A fine granularity load balancing technique for MMOG servers using a kd-tree to partition the space

Carlos Eduardo B. Bezerra, João L. D. Comba, Cláudio F. R. Geyer
Instituto de Informática
Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500, Porto Alegre

Abstract

MMOGs (massively multiplayer online games) are applications that require high bandwidth connections to work properly. This demand for bandwidth is specially critical on the servers that host the game. This happens because the typical number of simultaneous participants in this kind of game varies from a few hundreds to several tens of thousands, and the server is the one responsible for mediating the interaction between every pair of players connected to it. To deal with this problem, decentralized architectures with multiple servers have been proposed, where each server manages a region of the virtual environment of the game. Each player, then, connects only to the server that manages the region where he is playing. However, to distribute the load among the servers, it is necessary to devise an algorithm for partitioning the virtual environment. In order to readjust the load distribution during the game, this algorithm must be dynamic. Some work has already been made in this direction, but with a geometric algorithm, more appropriate than those found in the literature, it should be possible to reduce the distribution granularity without compromising the rebalancing time, or even reducing it. In this work, we propose the use of a kd-tree for dividing the virtual environment of the game into regions, each of which being designated to one of the servers. The split coordinates of the regions are adjusted dynamically according to the distribution of avatars in the virtual environment. We compared our algorithm to some approaches found in the literature and the simulation results show that our algorithm performed better in most aspects we analyzed.

Keywords: MMOGs, load balancing, distributed server, kd-trees.

Author's Contact:

{carlos.bezera, comba, geyer}@inf.ufrgs.br

1 Introduction

The main characteristic of MMOGs is the large number of players interacting simultaneously, reaching the number of tens of thousands [Schiele et al. 2007]. When using a client-server architecture for the players to communicate with one another, the server intermediates the communication between each pair of players.

To allow the interaction of players, each one of them sends his commands to the server, which calculates the resulting game state and sends it to all the players to whom the state change is relevant. We can see that the number of state update messages sent by the server may grow proportionally to the square of the number of players, if all players are interacting with one another. Obviously, depending on the number of players, the cost of maintaining a centralized infrastructure like this is too high, restricting the MMOG market to large companies with enough resources to pay the upkeep of the server.

In order to reduce this cost, several decentralized solutions have been proposed. Some of them use peer-to-peer networks, such as [Schiele et al. 2007; Rieche et al. 2007; Hampel et al. 2006; El Rhalibi and Merabti 2005; Iimura et al. 2004; Knutsson et al. 2004]. Others propose the use of a distributed server composed of low-cost nodes connected through the Internet, as in [Ng et al. 2002; Chertov and Fahmy 2006; Lee and Lee 2003; Assiotis and Tzanov 2006]. Anyway, in all these approaches, the "world", or virtual environment of the game is divided into regions and for every region is assigned a server – or a group of peers to manage it, when using peer-to-peer networks. Each of these regions must have a content such that the load imposed on the corresponding server is not greater than its capacity.

When an *avatar* (representation of the player in the virtual environment) is located in a region, the player controlling that avatar connects to the server associated to that region. That server, then, is responsible for receiving the input from that player and for sending, in response, the update messages. When a server becomes overloaded due to an excessive number of avatars in its region and, therefore, more players to be updated, the division of the virtual environment must be recalculated in order to alleviate the overloaded server.

Usually, the virtual environment is divided into relatively small cells, which are then grouped into regions and distributed among the servers. However, this approach has a severe limitation in its granularity, since the cells have fixed size and position. Using a more appropriate geometric algorithm, it should be possible to achieve a better player distribution among different servers, making use of traditional techniques that are generally used for computer graphics.

In this work, we propose the utilization of a kd-tree to perform the partitioning of the virtual environment. When a server is overloaded, it triggers the load balancing, readjusting the limits of its region by changing the split coordinates stored in the kd-tree. A prototype has been developed and used in simulations. The results found in these simulations have been compared to previous results from approaches which use the cell division technique.

The text is organized as follows: in section 2, some related works are described; in section 3, the algorithm proposed here is presented in detail; in the sections 4 and 5, we present, respectively, the simulation details and its results and, in section 6, the conclusions of this work are presented.

2 Related Work

Different authors have tried to address the problem of partitioning the virtual environment in MMOGs for distribution among multiple servers [Ahmed and Shirmohammadi 2008; Bezerra and Geyer 2009]. Generally, there is a static division into cells of fixed size and position. The cells are then grouped into regions (Figure 1), and each region is delegated to one of the servers. When one of them is overwhelmed, it seeks other servers, which can absorb part of the load. This is done by distributing one or more cells of the overloaded server to other servers.

[Ahmed and Shirmohammadi 2008], for example, propose a cell-oriented load balancing model. To balance the load, their algorithm finds, first, all clusters of cells that are managed by the overloaded server. The smallest cluster is selected and, from this cluster, it is chosen the cell which has the least interaction with other cells of the same server – the interaction between two cells A and B is defined by the authors as the number of pairs of avatars interacting with each other, one of them in A and the other one in B. The selected cell is then transferred to the least loaded server, considering "load" as the bandwidth used to send state updates to the players whose avatars are positioned in the cells managed by that server. This process is repeated until the server is no longer overloaded or there is no more servers capable of absorbing more load – in this case, one option could be to reduce the frequency at which state update messages are sent to the players, as suggested by [Bezerra et al. 2008].

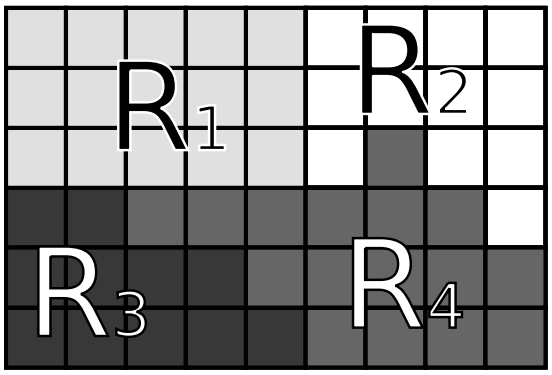


Figure 1: *Division into cells and grouping into regions*

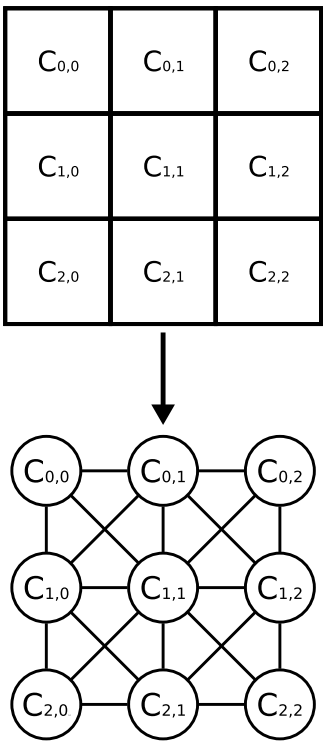


Figure 2: *Graph representation of the virtual environment*

In [Bezerra and Geyer 2009], it is also proposed the division into cells. To perform the division, the environment is represented by a graph (Figure 2), where each vertex represents a cell. Every edge in the graph connects two vertices representing neighboring cells. The weight of a vertex is the server's bandwidth occupied to send state updates to the players whose avatars are in the cell represented by that vertex. The interaction between any two cells define the weight of the edge connecting the corresponding vertices. To form the regions, the graph is partitioned using a greedy algorithm: starting from the heaviest vertex, at each step it is added the vertex connected by the heaviest edge to any of the vertices already selected, until the total weight of the partition of the graph – defined as the sum of the vertices' weights – reaches a certain threshold related to the total capacity of the server that will receive the region represented by that partition of the graph.

Although this approach works, there is a serious limitation on the distribution granularity it can achieve. If a finer granularity is desired, it is necessary to use very small cells, increasing the number of vertices in the graph that represents the virtual environment and, consequently, the time required to perform the balancing. Besides, the control message containing the list of cells designated to each server also becomes longer. Thus, it may be better to use another approach to perform the partitioning of the virtual environment, possibly using a more suitable data structure, such as the kd-tree [Bentley 1975].

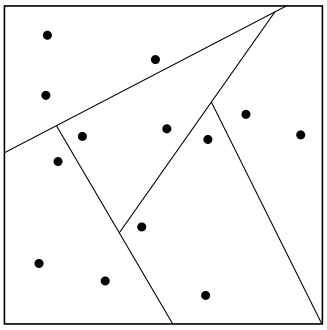


Figure 3: *Space partitioning using a BSP tree*

This kind of data structure is generally used in computer graphics. However, as in MMOGs there is geometric information – such as the position of the avatars in the environment –, space partitioning trees can be used. Moreover, we cand find in the literature techniques for keeping the partitions defined by the tree with a similar "load". In [Luque et al. 2005], for example, it is sought to reduce the time needed to calculate the collisions between pairs of objects moving through space. The authors propose the use of a BSP (binary space partitioning) tree to distribute the objects in the scene (Figure 3). Obviously, if each object of a pair is completely inserted in a different partition, they do not collide and there is no need to perform a more complex test for this pair. Assuming an initial division, it is proposed by the authors a dynamic readjustment of the tree as objects move, balancing their distribution on the leaf-nodes of the tree and, therefore, minimizing the time required to perform the collision detection. Some of the ideas proposed by the authors may be used in the context of load balancing between servers in MMOGs.

3 Proposed approach

The load balancing approach proposed here is based on two criteria: first, the system should be considered heterogeneous (i.e. every server may have a different amount of resources) and, second, the load on each server is *not* proportional to the number of players connected to it, but to the amount of bandwidth required to send state update messages to them.

This choice is due to the fact that every player sends commands to the server at a constant rate, so the number of messages received by the server per unit time grows linearly with the number of players, whereas the number of state update messages sent by the server may be quadratic, in the worst case.

As mentioned in the introduction, to divide the environment of the game into regions, we propose the utilization of a data structure known as kd-tree. The vast majority of MMOGs, such as World of Warcraft [Blizzard 2004], Ragnarok [Gravity 2001] and Lineage II [NCsoft 2003], despite having three-dimensional graphics, the simulated world – cities, forests, swamps and points of interest in general – in these games is mapped in two dimensions. Therefore, we propose to use a kd-tree with k = 2.

Each node of the tree represents a region of the space and, moreover, in this node it is stored a split coordinate. Each one of the two children of that node represents a subdivision of the region represented by the parent node, and one of them represents the sub-region before the split coordinate and the other one, the sub-region containing points whose coordinates are greater than or equal to the split coordinate. The split axis (in the case of two dimensions, the axes x and y) of the coordinate stored alternates for every level of the tree – if the first level nodes store x-coordinates, the second level nodes store y-coordinates and so on. Every leaf node also represents a region of the space, but it does not store any split coordinate. Instead, it stores a list of the avatars present in that region. Finally, each leaf node is associated to a server of the game. When a server is overloaded, it triggers the load balancing, which uses the kd-tree to readjust the split coordinates that define its region, reducing the amount of content managed by it.

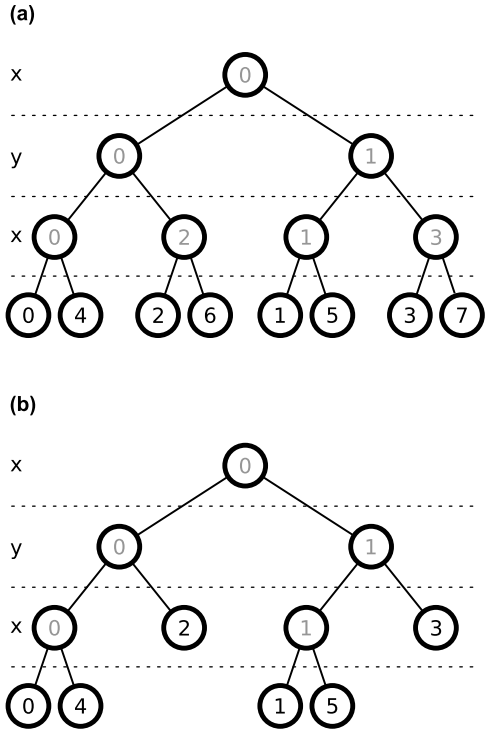


Figure 4: *Balanced kd-trees built with the described algorithm*

Every node of the tree also stores two other values: capacity and load of the subtree. The load of a non-leaf node is equal to the sum of the load of its children. Similarly, the capacity of a non-leaf node is equal to the sum of the capacity of its children nodes. For the leaf nodes, these values are the same of the server associated to each one of them. The tree root stores, therefore, the total weight of the game and the total capacity of the server system.

In the following sections, it will be described the construction of the tree, the calculation of the load associated with each server and the proposed balancing algorithm.

3.1 Building the kd-tree

To make an initial space division, it is constructed a balanced kd-tree. For this, we use the recursive function shown in Algorithm 1 to create the tree.

Algorithm 1 node::build_tree(id, level, num_servers)

if id + $2^{level} \geq num\_servers$ **then**
  $left\_child \leftarrow right\_child \leftarrow NIL$;
  return;
else
  $left\_child \leftarrow$ new_node();
  $left\_child.parent \leftarrow$ **this**;
  $right\_child \leftarrow$ new_node();
  $right\_child.parent \leftarrow$ **this**;
  $left\_child$.build_tree$(id, level + 1, num_servers)$;
  $right\_child$.build_tree$(id+2^{level}, level+1, num_servers)$;
end if

In Algorithm 1, the id value is used to calculate whether each node has children or not and, in the leaf nodes, it determines the server associated to the region represented by each leaf of the tree. The purpose of this is to create a balanced tree, where the number of leaf nodes on each of the two sub-trees of any node differs, in the maximum, by one. In Figure 4 (a), we have a full kd-tree formed with this simple algorithm and, in Figure 4 (b), an incomplete kd-tree with six-leaf nodes. As we can see, every node of the tree in (b) has two sub-trees whose number of leaf nodes differs by one in the worst case.

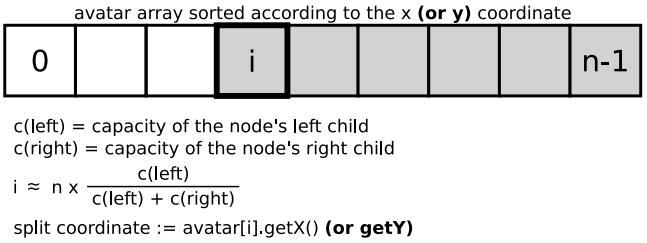


Figure 5: *A load splitting considering only the number os avatars*

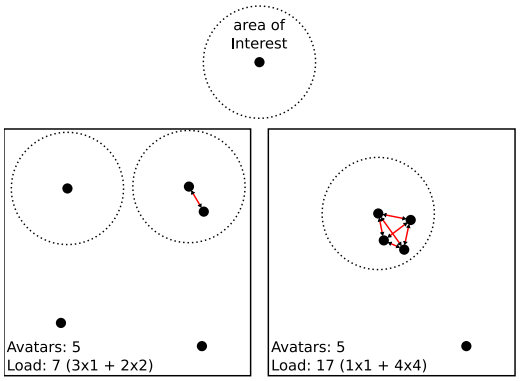


Figure 6: *Relation between avatars and load*

3.2 Calculating the load of avatars and tree nodes

The definition of the split coordinate for every non-leaf node of the tree depends on how the avatars will be distributed among the regions. An initial idea might be to distribute the players among servers, so that the number of players on each server is proportional to the bandwidth of that server. To calculate the split coordinate, it would be enough to simply sort the avatars in an array along the axis used (x or y) by the tree node to split the space and, then, calculate the index in the vector, such that the number of elements before this index is proportional to the capacity of the left child and the number of elements from that index to the end of the array is proportional to the capacity of the right child (Figure 5). The complexity of this operation is $O(nlogn)$, due to the sorting of avatars.

However, this distribution is not optimal, for the load imposed by the players depends on how they are interacting with one another. For example, if the avatars of two players are distant from each other, there will be probably no interaction between them and, therefore, the server will need only to update every one of them about the outcome of his own actions – for these, the growth in the number of messages is linear with the number of players. On the other hand, if the avatars are close to each other, each player should be updated not only about the outcome of his own actions but also about the actions of every other player – in this case, the number of messages may grow quadratically with the number of players (Figure 6). For this reason, it is not sufficient only to consider the number of players to divide them among the servers.

A more appropriate way to divide the avatars is by considering the load imposed by each one of them on the server. A brute-force method for calculating the loads would be to get the distance separating each pair of avatars and, based on their interaction, calculate the number of messages that each player should receive by unit of time. This approach has complexity $O(n^2)$. However, if the avatars are sorted according to their coordinates on the axis used to divide the space in the kd-tree, this calculation may be performed in less time.

For this, two nested loops are used to sweep the avatars array, where each of the avatars contains a $load$ variable initialized with zero. As the vector is sorted, the inner loop may start from an index before which it is known that no avatar a_j has relevance to that being referenced in the outer loop, a_i. It is used a variable $begin$, with initial value of zero: if the coordinate of a_j is smaller than that of

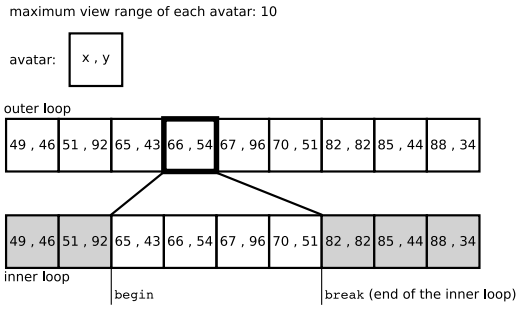


Figure 7: *Sweep of the sorted array of avatars*

a_i, with a difference greater than the maximum view range of the avatars, the variable $begin$ is incremented. For every a_j which is at a distance smaller than the maximum view range, the $load$ of a_i is increased according to the relevance of a_j to a_i. When the inner loop reaches an avatar a_j, such that its coordinate is greater than that of a_i, with a difference greater than the view range, the outer loop moves immediately to the next step, incrementing a_i and setting the value of a_j to that stored in $begin$ (Figure 7).

Let $width$ be the length of the virtual environment along the axis used for the splitting; let also $radius$ be the maximum view range of the avatars, and n, the number of avatars. The number of relevance calculations, assuming that the avatars are uniformly distributed in the virtual environment is $O(m \times n)$, where m is the number of avatars compared in the internal loop, i.e. $m = \frac{2 \times radius \times n}{width}$. The complexity of sorting the avatars along one of the axes is $O(nlogn)$. Although it is still quadratic, the execution time is reduced significantly, depending on the size of the virtual environment and on the view range of the avatars. The algorithm could go further and sort each set of avatars a_j which are close (in one of the axes) to a_i according to the other axis and, again, perform a sweep eliminating those which are too far away, in both dimensions. The number of relevance calculations would be $O(p \times n)$, where p is the number of avatars close to a_i, considering the two axes of coordinates, i.e. $p = \frac{(2 \times radius)^2 \times n}{width \times height}$. In this case, $height$ is the extension of the environment in the second axis taken as reference. Although there is a considerable reduction of the number of relevance calculations, it does not pay the time spent in sorting the sub-array of the avatars selected for each a_i. Adding up all the time spent on sort operations, it would be obtained a complexity of: $O(nlogn + n \times mlogm)$.

After calculating the load generated by each avatar, this value is used to define the load on each leaf node and, recursively, on the other nodes of the kd-tree. To each leaf node a server and a region of the virtual environment are assigned. The load of the leaf node is equal to the server's bandwidth used to send state updates to the players controlling the avatars located in its associated region. This way, the load of each leaf node is equal to the sum of the weights of the avatars located in the region represented by it.

3.3 Dynamic load balancing

Once the tree is built, each server is associated to a leaf node – which determines a region. All the state update messages to be sent to players whose avatars are located in a region must be sent by the corresponding server. When a server is overloaded, it may transfer part of the load assigned to it to some other server. To do this, the overloaded server collects some data from other servers and, using the kd-tree, it adjusts the split coordinates of the regions.

Every server maintains an array of the avatars located in the region managed by it, sorted according to the x coordinate. Also, each element of the array stores a pointer to another element, forming a chained list that is ordered according to the y coordinated of the avatars (Figure 8). By maintaining a local sorted avatar list on each server, the time required for balancing the load is somewhat reduced, for there will be no need for the server performing the rebalance to sort again the avatar lists sent by other servers. It will need only to merge all the avatars lists received from the other servers in

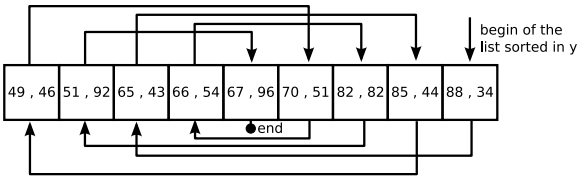


Figure 8: *Avatar array sorted by x, containing a list sort by y*

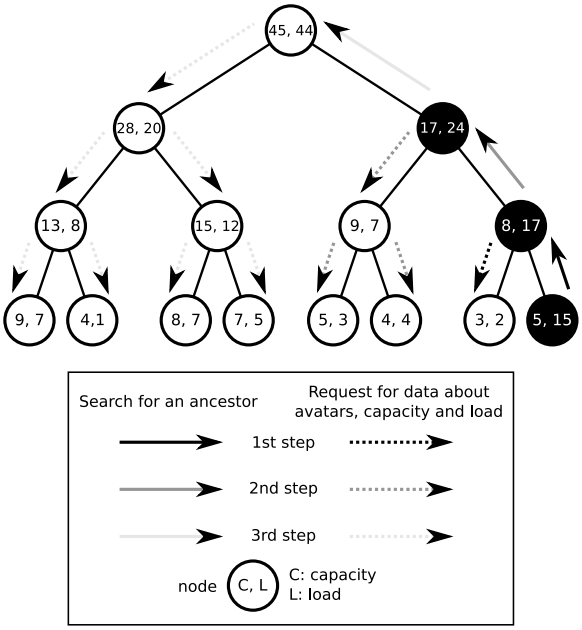


Figure 9: *Search for an ancestor node with enough resources*

an unique list, used to define the limits of the regions, what is done by changing the split coordinates which define the space partitions.

When the overloaded server initiates the rebalance, it runs an algorithm that traverses the kd-tree, beginning from the leaf node that defines its region and going one level up at each step until it finds an ancestor node with a capacity greater then or equal to the load. While this node is not found, the algorithm continues recursively up the tree until it reaches the root. For each node visited, a request for the information about all the avatars and the values of load and capacity is sent to the servers represented by the leaf nodes of the sub-tree to the left of that node (Figure 9). With these data, and its own list of avatars and values of load and capacity, the overloaded server can calculate the load and capacity of its ancestral node visited in the kd-tree, which are not known beforehand – these values are sent on-demand to save up some bandwidth of the servers and to keep the system scalable.

Reaching an ancestral node with capacity greater than or equal to the load – or the root of the tree, if no such node is found – the server that initiated the balance adjusts the split coordinates of the kd-tree nodes. For each node, it sets the split coordinate in a way such that the avatars are distributed according to the capacity of the node's children. For this, it is calculated the load fraction that should be assigned to each child node. The avatar list is then swept, stopping at the index i such that the total load of the avatars before i is approximately equal to the value defined as the load to be designated to the left child of the node whose split coordinate is being calculated (Figure 10). The children nodes have also, in turn, their split coordinates readjusted recursively, so that they are checked for validity – the split coordinate stored in a node must belong to the region defined by its ancestors in the kd-tree – and readjusted to follow the balance criteria defined.

As the avatar lists received from the other servers are already sorted along both axes, it is enough to merge these structures with the avatar list of the server which initiated the rebalance. Assuming that each server already calculated the weight of each avatar man-

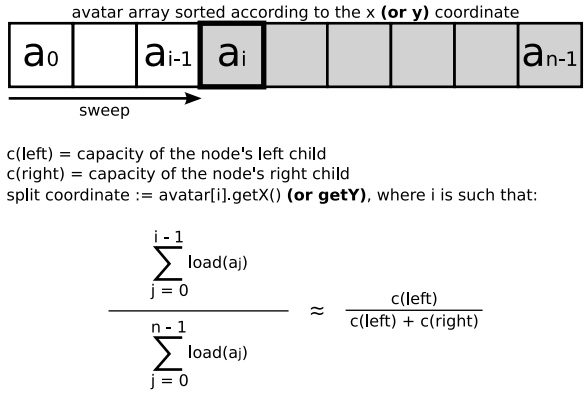


Figure 10: *Division of an avatar list between two brother nodes*

aged by it, the rebalance time is $O(nlogS)$, where n is the number of avatars in the game and S is the number of servers. The communication cost is $O(n)$, caused by the sending of data related to te n avatars. The merging of all avatar lists has $O(n)$ complexity, for the avatars were already sorted by the servers. At each level of the kd-tree, $O(n)$ avatars are swept in the worst case, in order to find the i index whose avatar's coordinate will be used to split the regions defined by each node of the tree (Figure 10). As this is a balanced tree with S leaf nodes, it has a height of $\lceil logS \rceil$.

4 Simulations

To evaluate the proposed dynamic load balancing algorithm, a virtual environment across which many avatars moved was simulated. Starting from a random point in the environment, each avatar moved according to the random waypoint model [Bettstetter et al. 2002]. To force a load imbalance and stress the algorithms tested, we defined some *hotspots* – points of interest to which the avatars moved with a higher probability than to other parts of the map. This way, a higher concentration of avatars was formed in some areas. Although the movement model used is not very realistic in terms of the way the players move their avatars in real games, it was only used to verify the load balance algorithms simulated. For each algorithm tested, we simulated two situations: one with the presence of hotspots and one without hotspots.

The proposed approach was compared to the ones presented in section 2, from other authors. However, it is important to observe that the model employed by [Ahmed and Shirmohammadi 2008] considers hexagonal cells, while in our simulations we used rectangular cells. Furthermore, the authors considered that there is a transmission rate threshold, which is the same for all servers in the system. As we assume a heterogenous system, their algorithm was simulated considering that each server has its own transmission rate threshold, depending on the upload bandwidth available in each one of them. However, we kept what we consider the core idea of the authors' approach, which is the selection of the smallest cell cluster managed by the overload server, then choosing that cell with the lowest interaction with other cells of the same server, and finally the transferring of this cell to the least loaded server. Besides Ahmed's algorithm, we also simulated some of the ones proposed in [Bezerra and Geyer 2009] – Progrega and BFBCT.

The simulated virtual environment consisted of a two-dimensional space, with 750 moving avatars, whose players were divided among eight servers ($S_1, S_2, ..., S_8$), each of which related to one of the regions determined by the balancing algorithm. For the cell-oriented approaches simulated, the space was divided into a 15×15 cell grid, or 225 cells. The capacity of each server S_i was equal to $i \times 20000$, forming a heterogeneous system. This heterogeneity allowed us to evaluate the load balancing algorithms simulated according to the criterion of proportionality of the load distribution on the servers.

In addition to evaluate the algorithms according to the proportionality of the load distribution, it was also considered the number of player migrations between servers. Each migration involves a

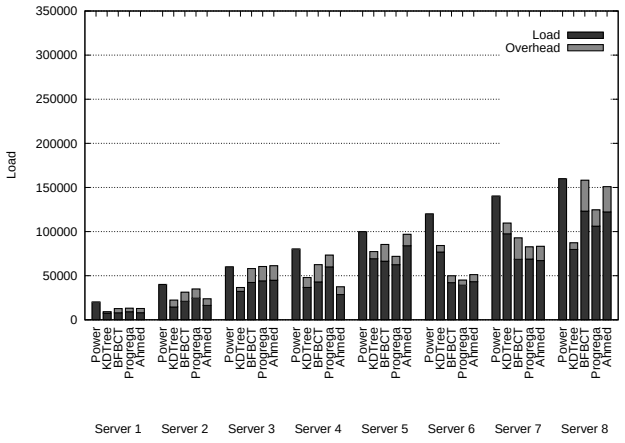


Figure 11: *Average load on each server (by algorithm, without hotspots)*

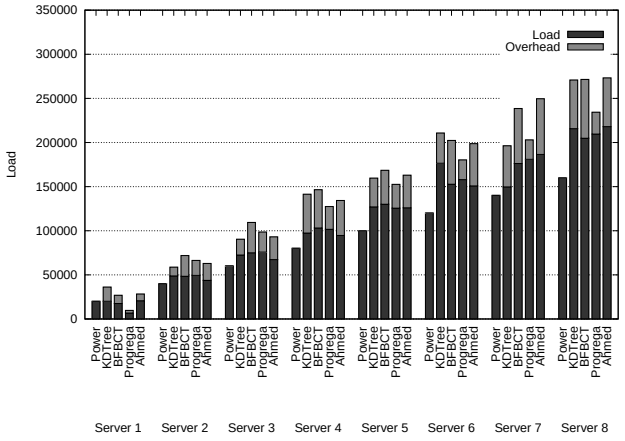


Figure 12: *Average load on each server (by algorithm, with hotspots)*

player connecting to the new server and disconnecting from the old one. This kind of situation may occur in two cases: the avatar moved, changing the region in which it is located and, consequently, changing the server to which its player is connected; or the avatar was not moving and still its player had to migrate to a new server. In the latter case, obviously the player's transfer was due to a rebalancing. An ideal balancing algorithm performs the load redistribution requiring the minimum possible number of player transfers between servers, while keeping the load on each server proportional to its capacity.

Finally, the inter-server communication overhead will also be evaluated. It occurs when two players are interacting, but each one of them is connected to a different server. Although the algorithm proposed in this work does not address this problem directly, it would be interesting to evaluate how the load distribution performed by it influences the communication between the servers.

5 Results

Figures 11 and 12 present the average load (plus the inter-server communication overhead) on each server, for each algorithm tested. The first figure shows the values in a situation without hotspots and, therefore, a smaller total load. The second, in turn, presents the load distribution when the server system is overloaded. We can see, in Figure 11, that all algorithms have met the objective of keeping the load on each server less than or equal to its capacity, when the system has sufficient resources to do so. In Figure 12, it is demonstrated that all the algorithms managed to dilute – in a more or less proportional manner – the load excess on the servers. It is important to observe, however, that the load shown in Figure 12 is only theo-

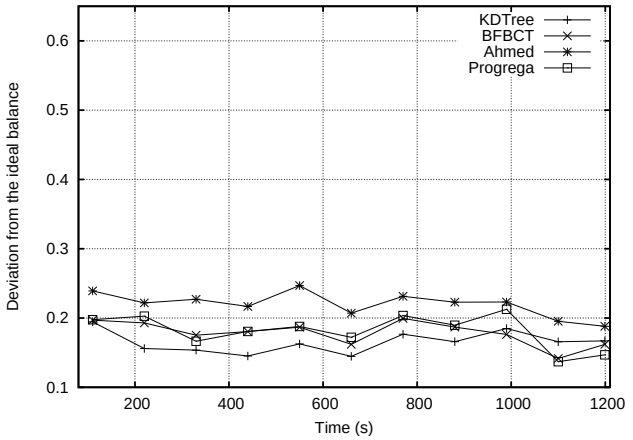


Figure 13: *Average deviation of the ideal balance of the servers (without hotspots)*

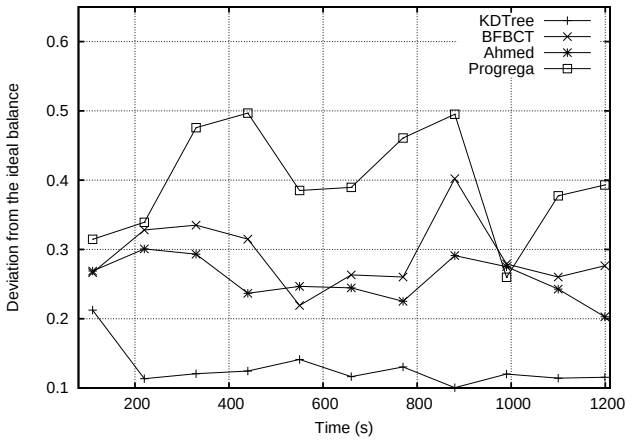


Figure 14: *Average deviation of the ideal balance of the servers (with hotspots)*

retical. Each server will perform some kind of "graceful degradation" in order to keep the load under its capacity. For example, the update frequency might be reduced and access to the game could be denied for new players attempting to join, which is a common practice in most MMOGs.

In figures 13 and 14, it is shown how much the balance generated by each algorithm deviates from an ideal balance – that is, how much, on the average, the load on the servers deviate from a value exactly proportional to the capacity of each one of them – over time. It is possible to observe that, in both situations – with and without hotspots – the algorithm that uses the kd-tree has the least deviation. This is the due to the fine granularity of its distribution, which, unlike the other approaches tested, is not limited by the size of a cell. In the situation with hotspots, the algorithm that uses the kd-tree is particularly effective, because rebalance is needed. In a situation where the system has more resources than necessary, the proportionality of the distribution is not as important: it is enough that each server manages a load smaller than its capacity.

Regarding player migrations between servers, all the algorithms – except BFBCT – had a similar number of user migrations in the absence of hotspots (Figure 15). This happens because the load of the game is less than the total capacity of the server system, which required less rebalancing and, thus, caused less migrations of players between servers. Figure 16, however, demonstrates that the algorithm that uses the kd-tree had a significantly lower number of user migrations than the other approaches. This is due, in the first place, to the fact that the regions defined by the leaf nodes of the kd-tree are necessarily contiguous, and each server was linked to only one leaf node. An avatar moving across the environment divided into very fragmented regions constantly crosses the borders

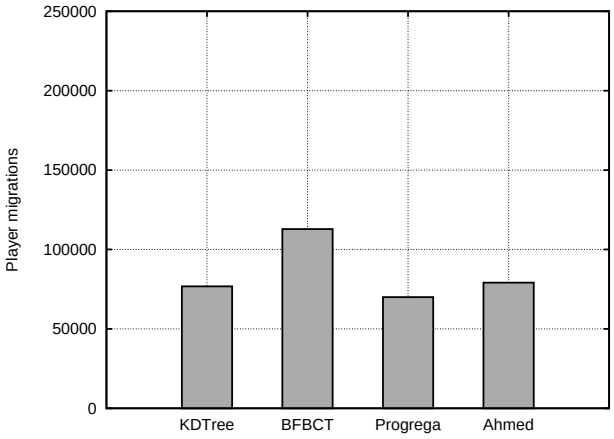


Figure 15: *Player migrations between servers (without hotspots)*

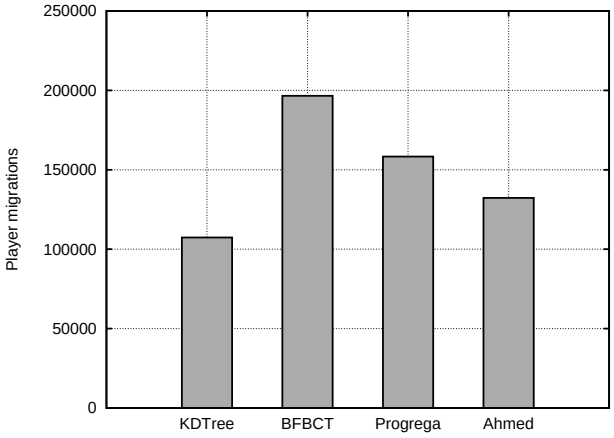


Figure 16: *Player migrations between servers (with hotspots)*

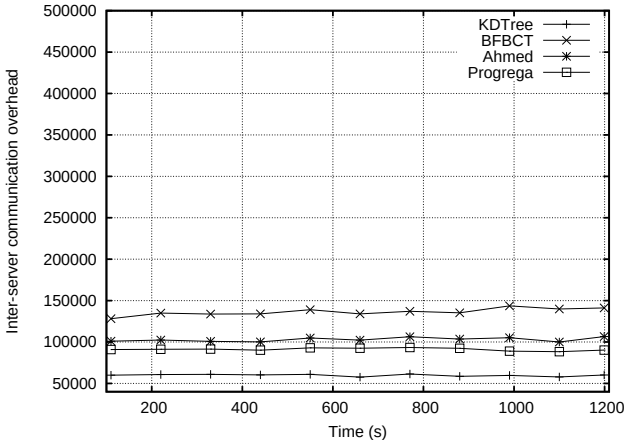


Figure 17: *Inter-server communication for each algorithm over time (without hotspots)*

between these regions and causes, therefore, its player to migrate from server to server repeatedly. Another reason for this result is that each rebalancing executed with the kd-tree gets much closer to an ideal distribution than the cell-based algorithms – again, thanks to the finer granularity of the kd-tree based distribution –, requiring less future rebalancing and, thus, causing less player migrations.

Finally, it is shown the amount of communication between servers for each simulated algorithm, over time. In Figure 17, all algorithms have similar results, and the one which uses the kd-tree is slightly better than the others. This is also explained by the fact that the regions are contiguous, minimizing the number of bound-

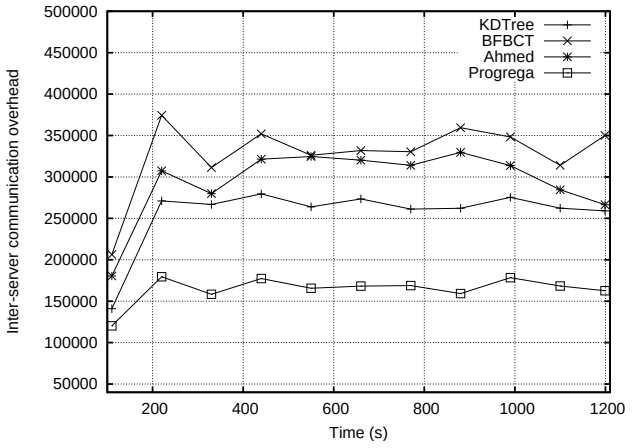


Figure 18: *Inter-server communication for each algorithm over time (with hotspots)*

aries between them and, consequently, reducing the probability of occurring interactions between pairs of avatars, each one in a different region. In Figure 18, it is possible to see that the inter-server communication caused by Progrega was considerably lower than all the others in a situation of system overload. The reason for this is that its main goal – besides balancing the load – is precisely to reduce the communication between servers. However, even not considering the additional cost, the algorithm that uses the kd-tree got second place in this criterion.

6 Conclusions

In this work, we proposed the use of a kd-tree to partition the virtual environment of MMOGs and perform the load balancing of servers by recursively adjusting the split coordinates stored in its nodes. One of the conclusions reached was that the use of kd-trees to make this partitioning allows a fine granularity of the load distribution, while the readjustment of the regions becomes simpler – by recursively traversing the tree – than the common approaches, based on cells and/or graph partitioning.

The finer granularity allows for a better balancing, so that the load assigned to each server is close to the ideal value that should be assigned to it. This better balance also helped to reduce the number of migrations, by performing less rebalancing operations. The fact that the regions defined by the kd-tree are necessarily contiguous was one of the factors that contributed to the results of the proposed algorithm, which was better than the other algorithms simulated in most of the criteria considered.

In conclusion, it was possible to use methods that can reduce the complexity of each rebalancing operation. This is due, first, to the reduction of the number of operations for calculating the relevance between pairs of avatars by sweeping a sorted avatar list and, secondly, to keeping at each server an avatar list already sorted in both dimensions, saving the time that would be spent on sorting the avatars when they were received by the server executing the rebalance.

Acknowledgements

The development of this work has been supported by the National Research Council (CNPq) and by the Coordination for Improvement of the Higher Education Personnel (CAPES).

References

AHMED, D., AND SHIRMOHAMMADI, S. 2008. A Microcell Oriented Load Balancing Model for Collaborative Virtual Environments. In *VECIMS*, 86–91.

ASSIOTIS, M., AND TZANOV, V. 2006. A distributed architecture for MMORPG. In *Proceedings of the ACM SIGCOMM workshop on Network and system support for games, NetGames, 5.*, New York: ACM, Singapore, 4.

BENTLEY, J. 1975. Multidimensional binary search trees used for associative searching.

BETTSTETTER, C., HARTENSTEIN, H., AND PÉREZ-COSTA, X. 2002. Stochastic Properties of the Random Waypoint Mobility Model: epoch length, direction distribution, and cell change rate. In *Proceedings of the ACM international workshop on Modeling analysis and simulation of wireless and mobile systems, 5.*, New York: ACM, Atlanta, GA, 7–14.

BEZERRA, C. E. B., AND GEYER, C. F. R. 2009. A load balancing scheme for massively multiplayer online games. *Massively Multiuser Online Gaming Systems and Applications, Special Issue of Springer's Journal of Multimedia Tools and Applications*.

BEZERRA, C. E. B., CECIN, F. R., AND GEYER, C. F. R. 2008. A3: a novel interest management algorithm for distributed simulations of mmogs. In *Proceedings of the IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, DS-RT, 12.*, Washington, DC: IEEE, Vancouver, Canada, 35–42.

BLIZZARD, 2004. World of warcraft. 2004. Available at: <http://www.worldofwarcraft.com/>. Last time accessed: 24 jul. 2009.

CHERTOV, R., AND FAHMY, S. 2006. Optimistic Load Balancing in a Distributed Virtual Environment. In *Proceedings of the ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV, 16.*, New York: ACM, Newport, USA, 1–6.

EL RHALIBI, A., AND MERABTI, M. 2005. Agents-based modeling for a peer-to-peer MMOG architecture. *Computers in Entertainment (CIE) 3*, 2, 3–3.

GRAVITY, 2001. Raganarök online. 2001. Available at: <http://www.ragnarokonline.com/>. Last time accessed: 24 jul. 2009.

HAMPEL, T., BOPP, T., AND HINN, R. 2006. A peer-to-peer architecture for massive multiplayer online games. In *Proceedings of the ACM SIGCOMM workshop on Network and system support for games, NetGames, 5.*, New York: ACM, Singapore, 48.

IIMURA, T., HAZEYAMA, H., AND KADOBAYASHI, Y. 2004. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In *Proceedings of the ACM SIGCOMM workshop on Network and system support for games, NetGames, 3.*, New York: ACM, Portland, USA, 116–120.

KNUTSSON, B., ET AL. 2004. Peer-to-peer support for massively multiplayer games. In *Proceedings of the IEEE Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM, 23.*, [S.l.]: IEEE, Hong Kong, 96–107.

LEE, K., AND LEE, D. 2003. A scalable dynamic load distribution scheme for multi-server distributed virtual environment systems with highly-skewed user distribution. In *Proceedings of the ACM symposium on Virtual reality software and technology*, New York: ACM, Osaka, Japan, 160–168.

LUQUE, R., COMBA, J., AND FREITAS, C. 2005. Broad-phase collision detection using semi-adjusting BSP-trees. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM New York, NY, USA, 179–186.

NCSOFT, 2003. Lineage ii. 2003. Available at: <http://www.lineage2.com/>. Last time accessed: 24 jul. 2009.

NG, B., ET AL. 2002. A multi-server architecture for distributed virtual walkthrough. In *Proceedings of the ACM symposium on Virtual reality software and technology, VRST*, New York: ACM, Hong Kong, 163–170.

RIECHE, S., ET AL. 2007. Peer-to-Peer-based Infrastructure Support for Massively Multiplayer Online Games. In *Proceedings of the 4th IEEE Consumer Communications and Networking Conference, CCNC, 4.*, [S.l.]: IEEE, Las Vegas, NV, 763–767.

SCHIELE, G., ET AL. 2007. Requirements of Peer-to-Peer-based Massively Multiplayer Online Gaming. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, CCGRID, 7.*, Washington, DC: IEEE, Rio de Janeiro, 773–782.

A Generative Programming Approach for Game Development

Victor T. Sarinho* Antônio L. Apolinário

State University of Feira de Santana, Brazil

Abstract

Nowadays, due to the great distance between design and implementation worlds, different skills are necessary to create a game system. To solve this problem, a lot of strategies for game development, trying to increase the abstraction level necessary for the game production, were proposed. In this way, a lot of game engines, game frameworks and others, in most cases without any compatibility or reuse criteria between them, were developed. This paper presents a new generative programming approach, able to increase the production of a digital game by the integration of different game development artifacts, following a system family strategy focused on variable and common aspects of a computer game. As result, high level abstractions of games, based on a common language, can be used to configure metaprogramming transformations during the game production, providing a great compatibility level between game domain and game implementation artifacts.

Keywords: generative game development, game feature models, game specification language, game metaprogramming.

Authors' contact:
*vsarinho@gmail.com
apolinario@ecomp.uefs.br

1. Introduction

During a game development project, according the design and implementation diversity, the game production becomes highly complex and expensive [Sarinho and Apolinário 2008].

To change this scenario, new approaches for game production, able to organize the game development process, using available engines and frameworks, are necessary [Furtado and Santos 2006].

An interesting approach able to perform these objectives in a software production is the generative programming. According Czarnecki and Eisenecker [2000], generative programming is "the process to generate programs where automated source code creation is done through code generators to improve programmer productivity".

The difference of generative programming by other approaches is because it aims to automate the software development process using a wide range of static and dynamic technologies, including metaprogramming, reflection, program and model analysis, for example [Czarnecki and Kim 2005]. It aims to model and implement "system families" in such a way that a given system can be automatically generated from a specification written in one or more textual or graphical domain specific languages [Czarnecki 2004].

Nevertheless, to define a generative programming approach for game development, a lot of available development techniques and domain aspects used during the game production must be considered. Therefore, questions like: "How the generative programming can be applied on game development?", "How many game software artifacts are necessary to achieve an automatic source code generation for a game development?", and "How the commonality and variability can be represented and used during the game development?" are raised.

This paper presents a generative programming proposal for generic game development. It is based on game feature models, able to represent variable and common implementation aspects of computer games, and metaprogramming resources, able to represent and generate compatible source code for available game engines and game frameworks.

It is organized as follows: Section 2 presents some generative software artifacts, and their related game works, necessary for the game development. Section 3 describes the ***G**ameSystem*, ***D**ecisionSupport* and ***S**ceneView* (**GDS**) feature model; the **G**ame **S**pecification **L**anguage (**GSL**); and the metaprogramming approach necessary for the game development. Section 4 presents the Pac-Man case study using the generative programming proposal. Finally, Section 5 presents some conclusions about the paper.

2. Related Work

Next subsections will present important artifacts for generative programming that will be used in this paper. In addition, some game researchs for each type of generative artifact will be described.

2.1 Feature Model Artifacts

According Czarnecki and Kim [2005], the feature modeling has several applications in generative software development, including domain analysis, product-line scoping, and feature-based product specification.

Feature modeling is a technique for managing software commonalities and variabilities. It can be used to: capture the results of domain analysis; facilitate scoping of product lines, domain-specific language families, components, platforms, and other reusable assets; and provide a basis for automated configuration of concrete products, languages, components, platforms, etc [Czarnecki and Kim 2005].

In a game development perspective, Zhang and Jarzabek [2005] proposed the application of feature models, defining similarities and differences among four different RPGs, to configure a RPG product line architecture (RPG-PLA).

To represent a generic game design, Sarinho and Apolinário [2008] proposed the **NESI** feature model, able to represent conceptual game aspects in four main views: **N**arrative, **E**ntertainment, **S**imulation and **I**nteraction.

Another interesting work was presented by Furtado [2006] where a game ontology to set game implementations was proposed. It was not a feature model, but some similar characteristics were presented in this research.

Unfortunately, none of them are able to represent common and variable aspects of implementation in a generic computer game.

2.2 Domain-Specific Language Artifacts

According Fowler [2005], a Domain-Specific Language (DSL) is a limited form of computer language designed for a specific class of problems. It is a small, usually declarative, language that offers expressive power focused on a particular problem domain [Furtado and Santos 2006]. In many cases, DSL programs are translated to calls to a common subroutine library, and the DSL can be viewed as a way to hide the details of that library [Furtado and Santos 2006].

An interesting research about DSL for games was presented by Moreno-Ger et al. [2005], who described a suitable Domain-Specific Language to build educational games. To support this language, an abstract syntax and its operational semantics, and a specific game engine were presented.

In this work, DSLs will be usefull to realize the generic game feature models in a reusable way, providing an optimal support for application developers by the available and compatible software development environments.

2.3 Metaprogramming Artifacts

According Azanza et al. [2007], "Generative programming is about developing metaprograms that synthesize other programs", and according Batory [2006], "metaprogramming is the concept that program synthesis is a computation".

To exemplify, Batory [2006] asserts that Model-Driven Development (MDD) is a metaprogramming paradigm, where models are program values and transformations are program functions that map these values. In fact, "the main difference between MDD and generative software development is the focus of the latter on system families" [Czarnecki 2004].

For Trujillo et al. [2007], Scripts that transform models into executables are also metaprograms. They are "programs that manipulate values that themselves are programs".

Looking for game researchs, Cutumisu et al. [2007] presented a three-step process allowing game authors (non-programmers) to generate the necessary procedural scripts to implement meaningful interactions between the PC and game objects.

In this work, scripts that transform the DSLs models, able to represent generic characteristics of computer games, into executables will be used.

3. The Generative Approach

The main objective of the generative approach for games presented in this paper is to integrate feature models, domain-specific languages and metaprogramming scripts for games as a unique and organized approach for game implementation.

In this way, this section will present a feature model able to represent game implementation aspects, a domain-specific language able to represent the feature model as a concrete syntax, and a collection of metaprogramming scripts able to translate the DSL for game engines, frameworks and similars.

3.1 The GDS Model

The objective of the GDS model (Figure 1) is define a game as a combination of three main standard features: *GameSystem*, *DecisionSupport* and *SceneView*, where each main feature describes generic configurations and behavioral aspects of a game. It is organized as a collection of features depicting various resources of game implementation found in several related works (see reference section).

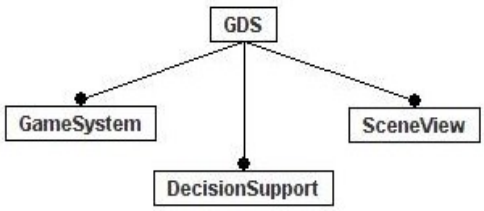


Figure 1: GDS feature model.

The *GameSystem* feature (Figure 2) is the main joint point of the game. It is responsible to control the game execution, describing available *GameBehaviors*, *GameContext* and *GameObservers* of the game. By the execution of *GameBehaviors*, activated by *GameObservers* or not, the current *Player* in the *GameSession* can trigger actions that can affect all defined data in the game, like *DecisionEntities* and *SceneNodes* for example. Some game subsystems, like *FileSystem* and *Networking* for example, can also execute some synchronous and asynchronous actions, triggering in the same way specific behaviors in the game.

The *DecisionSupport* feature (Figure 3) is an effort to integrate some AI game strategies used by different digital games. It presents *DecisionEntities*, like *Scenario* and *Agent*, the *ContextState* of each *DecisionEntity*, represented as *FiniteState* or *NeuralNet* for example, and predefined

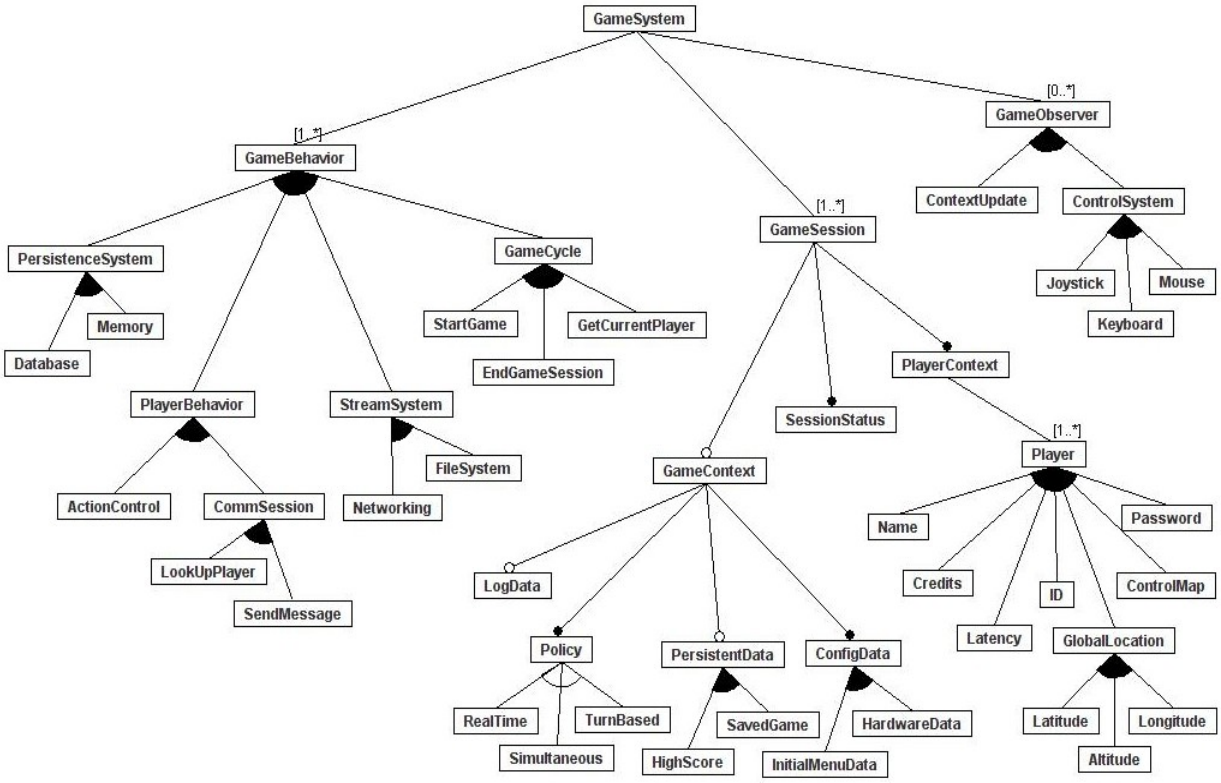


Figure 2: GameSystem feature diagram.

DecisionBehaviors, able to read and change *DecisionEntities* and *ContextStates* values during their executions.

The *SceneView* (Figure 4) feature is a collection of *SceneNodes* distributed by *Spatial* sessions with a lot of *SceneBehaviors* and *SceneObservers* to execute scene actions. Each *Spatial* session is composed by some *SceneNodes*. A *SceneNode* represents a hierarquical information about the scene, with a specific *Location* and a *BoudingVolume* for collision detection. Each *SceneNode* can represent at the same time *AudioNode*, *GraphicNode* and *PhysicsNode* informations. *Sorting* informations about the *Spatial*, like *Portals*, *BSPs* and others can also be defined.

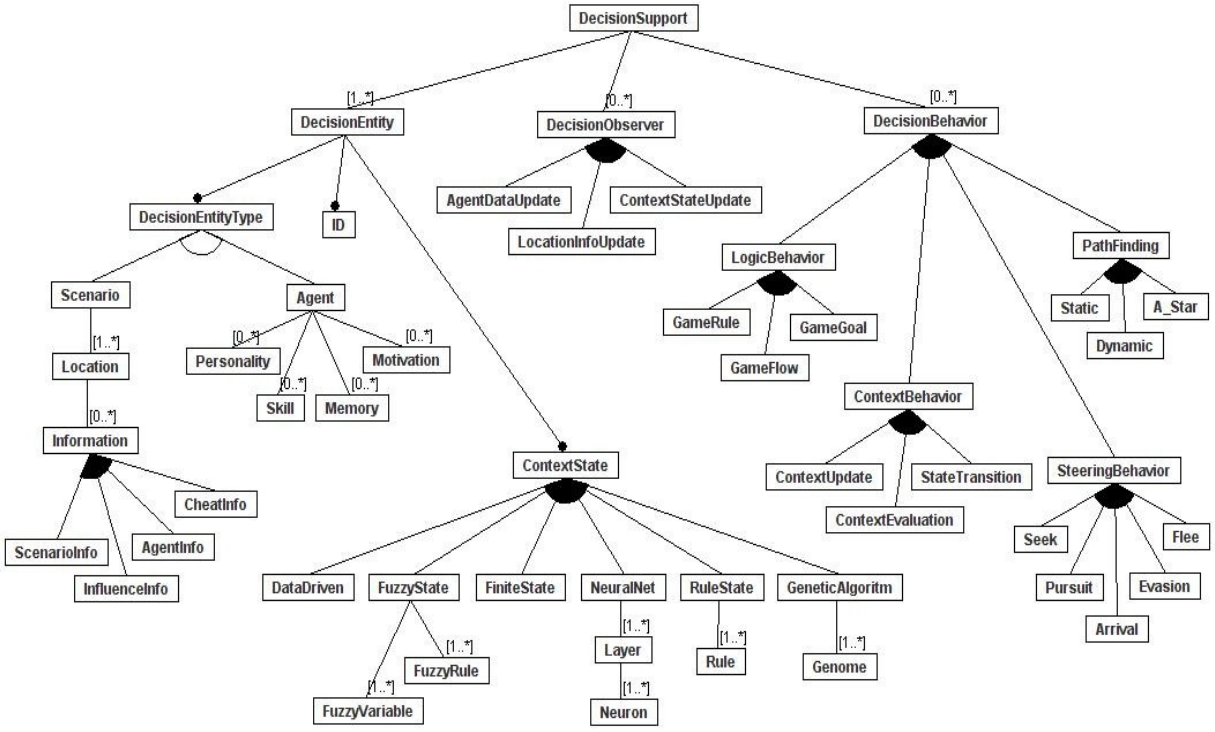


Figure 3: DecisionSupport feature diagram.

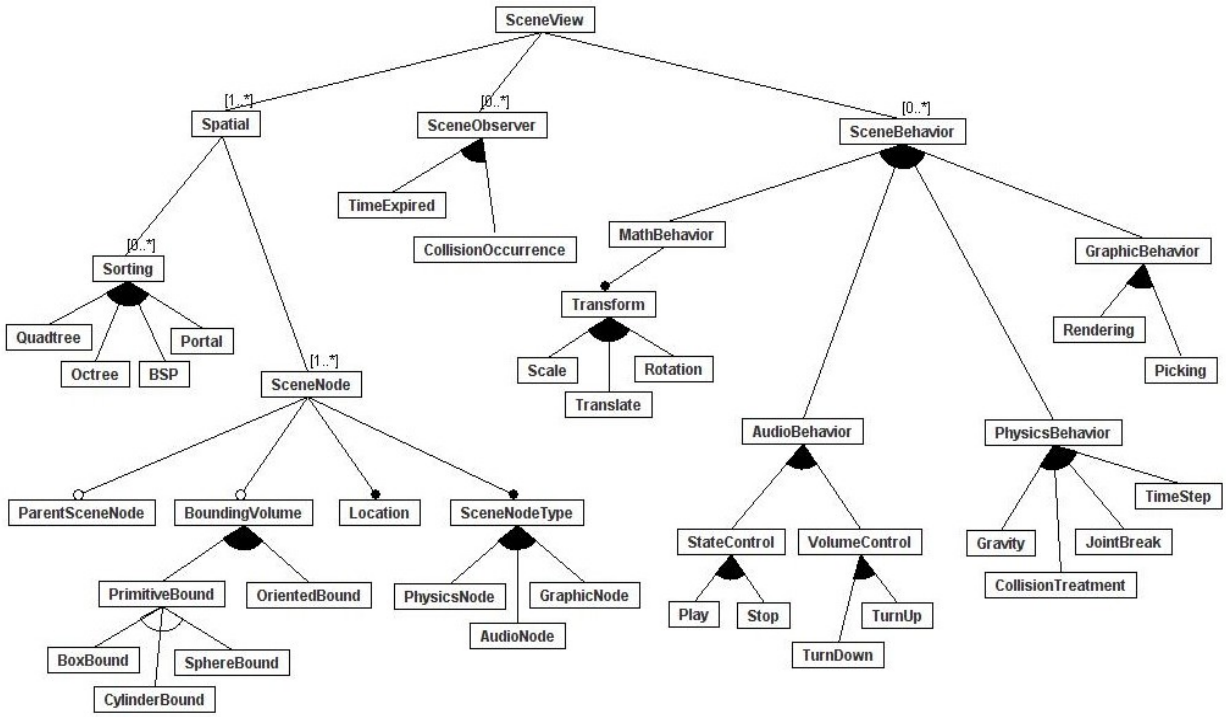


Figure 4: SceneView feature diagram.

3.2 The Game Specification Language

According [Czarnecki 2004], feature models are the starting point in the development of both DSLs and system-family architectures. DSL presents a lot of benefits: it is concise, self-documenting to a large extent, and can be reused for different purposes; it enhances productivity, reliability, maintainability and portability; it represents domain knowledge, enabling the conservation and reuse of this knowledge; and it allows validation and optimization at the domain level [Furtado and Santos 2006].

DSLs are usually declarative, offering only a restricted set of notations and abstractions. Consequently, they can be viewed as specification languages, like SQL, HTML, TeX, BNF and XML derivations, for example [Furtado and Santos 2006].

According W3C [2009], the Extensible Markup Language (XML) is a "simple, very flexible text format derived from SGML (ISO 8879)". It was originally designed to meet the challenges of large-scale electronic publishing, allowing the exchange of a wide variety of data on the Web and elsewhere [W3C 2009].

In this way, using the GDS model as a commonality and variability guide for game implementations, and the XML as a standard text format, a Game Specification Language (GSL), which can be caracterized as a textual DSL for games, is defined.

GSL presents a textual representation of the GDS model structure (taking advantage of the available XML tools), organized by the root tag *GSL* and its three main subtags: *GameSystem*, *DecisionSupport* and *SceneView*, with detailed values of attributes and subtags when necessary.

For each main subtags, a lot of tags can be defined to support the context data of a game. This context data can be represented using individual tags, like *GameContext* and *PlayerContext*, or using multiple tags, like *Players*, *DecisionEntities* and *SceneNodes*.

Subtags of *GameBehavior*, *GameObserver*, *DecisionBehavior*, *DecisionObserver*, *SceneBehavior* and *SceneObserver*, representing behavior and observer characteristics in a game, can also be defined, according the game project.

To illustrate a complete organization of these tags and subtags, section 4.1 shows an example of a GSL script, describing a simplified representation of a Pac-Man game [Iwatani 1980].

3.3 Metaprogramming Scripts for Games

Based on GDS model and XML text format, a GSL specification is a structured game description, focused on common and variable characteristics of a game implementation, where source code elements for real games can be created using a generative approach.

In fact, by the application of some translations in the GSL specification, using a specific game engine or game framework as the final target, the production of a real game based on a declarative resource (the GSL specification) is possible.

Unfortunately, different game development artifacts can be used during the game production (game engines, game makers, game frameworks, etc). As result, to support the logic and structure of a GSL specification, a specific translation process must be defined for each game development artifact used in the game production.

To solve this problem, a game metaframework, composed by a set of *core* classes able to support the

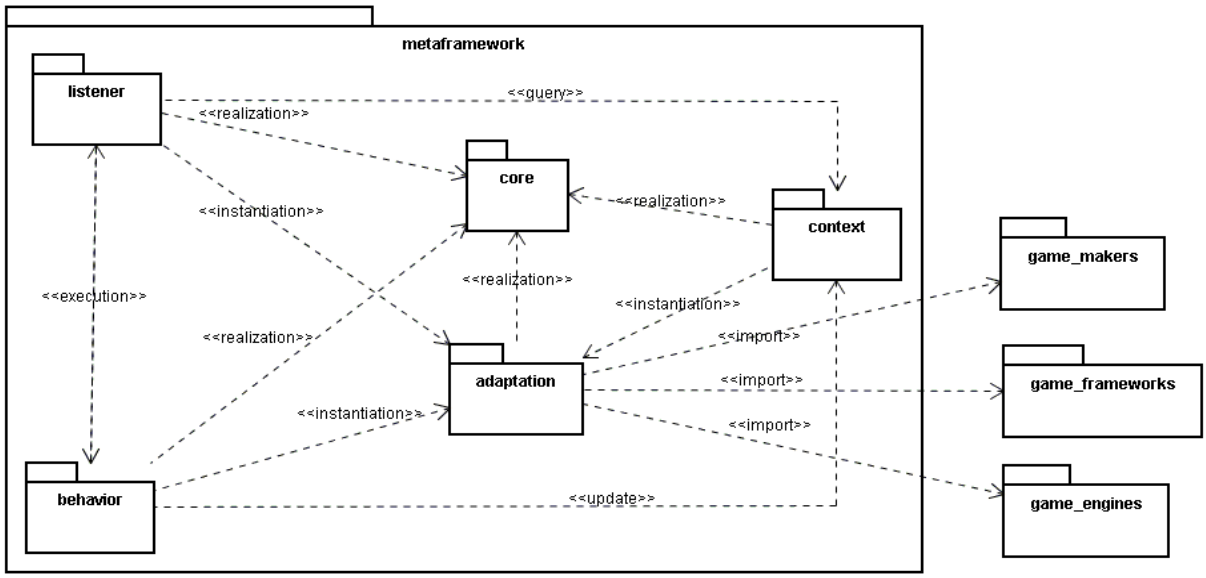


Figure 5: The metaframework package diagram.

GSL structure, that becomes the main target of the generative scripts based on GSL specifications, is proposed.

The metaframework objective is to create a communication layer between the generated code from GSL specifications with available game development artifacts. So, a direct consequence of the *adaptation* package (composed by classes that realize the *adapter* pattern [Gama et al. 1994]), necessary to maintain the compatibility between metaframework *core* classes and similars classes available in the respective game development artifact that will be reused.

Figure 5 illustrates the metaframework package diagram, showing the necessary structures to support GSL specifications: *core* (abstract structures to support GSL games), *adaptation* (collection of adapters necessary to support a new game engine or game framework), *behavior* (actions to be executed by a GSL game), *listener* (game observers that will be triggered by game subsystems or game behaviors) and *context* (game data structures that will be instantiated during the game play). External game development artifacts are also presented (*game engines*, *game frameworks* and *game makers*), showing their *import* relationships with *adaptation* structures, allowing the game portability with different game development artifacts.

Figure 6 presents a class diagram [Booch et. al. 1998] with an initial proposal for the *core* package. It describes a *GameSession* abstract class composed by three types of *GenericObservers* (*GameObserver*, *DecisionObserver* and *SceneObserver*), and some collections of game elements, like *spatials*, *decisionEntities* and *players*. A *Spatial* is composed by a collection of *sceneNodes* (which can be an *AudioNode*, a *GraphicNode* or a *PhysicsNode*). A *DecisionEntity* can be an *Agent* (with some collections of *skills*, *memories*, *motivations* and *personalities*) or a Sc*enario* (with a collection of *locations* and some *informations* for each *Location*). Each *DecisionEntity* contains informations about its *ContextState*, and each *GenericObserver* contains a collection of *GenericListener* able to execute a *GenericBehavior* when necessary.

For the metaprogramming scripts, these are described by Extensible Stylesheet Language (XSL) files [XSL 2009], able to translate GSL specifications in a compatible source code with the proposed metaframework. According Harold [1998], XSL includes both a transformation language and a formatting language. The transformation language "provides elements that define rules for how one XML document is transformed into another document". "They are purely about moving data from one computer system or program to another".

To illustrate the application of a XSL script to transform a GSL game to a real game, section 4.3 presents some examples of XSL rules able to produce a simplified version of the Pac-Man game (after translate the GSL specification presented in section 4.1).

4. Case Study: A Simplified Pac-Man

To exemplify an application of the proposed generative approach, a simplified version of the Pac-Man game will be designed.

This simplified project will ommit ghosts (only one ghost will be available), big pills (excluding the ghost vunerability), difficult levels (same skills for the ghost during the game play), score, fruits, ghost house, and others.

Only the default characteristics of the Pac-Man game will be present, like: the pac-man character, one ghost character, pills, walls, collision treatment between these game elements, lifes and a treatment for the completion of the game (game over or game victory).

Initial menu, game presentation, high scores, multiplayer support and other "*GameSystems*" characteristics were also ommited.

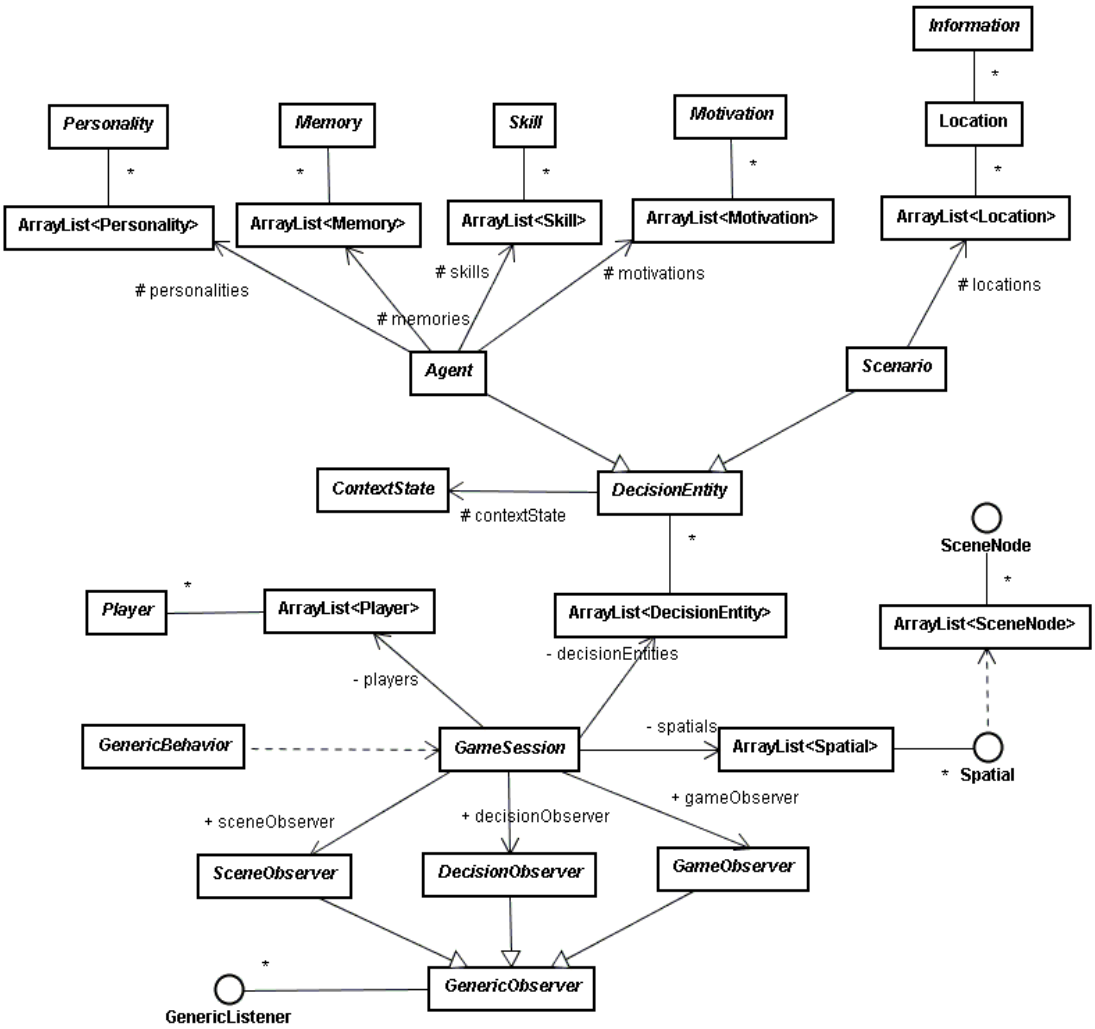


Figure 6: Class diagram for the core package.

4.1 Modeling the GSL Script

To represent the Simplified Pac-Man game, Figure 7 shows its GSL specification proposal, describing the necessary information to model and configure this game, allowing its realization according the generative approach.

For behaviors tags (*MoveGhost*, *RestoreInitialPosition*, for example) and observers tags (*UpDirection*, *LoseLifeListener*, for example), in this case, the existence of them is sufficient to represent the desired actions. Metaprogramming scripts will be responsible to decide what kind of source code will be generated by the existence of these tags.

For context tags (*DecisionEntities* and *SceneNodes*, for example), the structure and some parameters (*imageFile*, *X* and *Y* coordinates, for example) to configure the final source code, were described.

For the *Model* tag, used in all *SceneNodes*, it works with special files describing rendering and animation informations. As result, different ways to represent data file or image files are presented in the GSL specification for the *Model* tag: one image file, multiples image files, one data file, one data file and one image file, etc. Therefore, some special and different treatments are expected in the metaprogramming scripts for each *Model* tag.

Due to space limitations, the specification of *Spatial* tags was simplified (and highlighted using different font styles) to show only the *Location* and the *Model* tags.

4.2 New Classes and Adapters

According section 3.3, the proposed metaframework is the main target for generative scripts based on GSL specifications. In this way, to allow the realization of these GSL specifications, some adaptations and new structures will be applied to the metaframework, during the execution of those generative scripts.

To illustrate these metaframework updates, Figure 8 presents a class diagram showing some new classes added to the following metaframework packages: *adaptation*, *behavior*, *listener* and *context*. These new classes and adapters were developed according the GTGE game framework [GTGE 2009], allowing the communication between them.

As result, adaptations and new classes for *Player* (DefaultPlayer), *GameSession* (GameSessionGTGE adapter), *GenericObserver* (SceneObserverGTGE and GameObserverGTGE adapters), *Spatial* (SpatialField

```
<GSL>
  <GameSystem>
    <GameBehavior>
      <GameCycle>
        <GameOver/> <GameVictory/>
      </GameCycle>
      <PlayerBehavior> <ActionControl/> </PlayerBehavior>
    </GameBehavior>

    <GameObserver>
      <ControlSystem>
        <Keyboard>
          <UpDirection/> <DownDirection/>
          <LeftDirection/> <RightDirection/> <ESC/>
        </Keyboard>
      </ControlSystem>
    </GameObserver>

    <GameSession>
      <SessionStatus status="GamePlay"/>
      <PlayerContext>
        <Player>
          <ID value="1"/>
          <Name value="Player1"/>
          <ControlMap>
            <LeftDirection_MoveLeft/>
            <RightDirection_MoveRight/>
            <UpDirection_MoveUp/>
            <DownDirection_MoveDown/>
            <ESC_ButtonPress/>
          </ControlMap>
        </Player>
      </PlayerContext>
    </GameSession>
  </GameSystem>

  <DecisionSupport>
    <DecisionBehavior>
      <LogicBehavior>
        <GameRule> <LoseLife/> </GameRule>
        <GameFlow>
          <RestoreInitialPosition/>
        </GameFlow>
      </LogicBehavior>
    </DecisionBehavior>

    <DecisionObserver>
      <AgentDataUpdate>
        <LoseLifeListener/>
      </AgentDataUpdate>
    </DecisionObserver>

    <DecisionEntity>
      <ID value="PacMan"/>
      <DecisionEntityType>
        <Agent>
          <PacMan>
            <Skill> <Life value="2"/> </Skill>
          </PacMan>
        </Agent>
      </DecisionEntityType>
    </DecisionEntity>

    <DecisionEntity>
      <ID value="CornerScenario"/>
      <DecisionEntityType>
        <Scenario>
          <Location file="corner.dat">
            <Information>
              <ScenarioInfo> <Ways/> </ScenarioInfo>
            </Information>
          </Location>
        </Scenario>
      </DecisionEntityType>
    </DecisionEntity>
  </DecisionSupport>

  <SceneView>
    <SceneBehavior>
      <MathBehavior>
        <Translate>
          <MoveGhost/> <MovePacMan/>
        </Translate>
      </MathBehavior>
      <GraphicBehavior>
        <Rendering> <UpdateHUD/> </Rendering>
      </GraphicBehavior>
      <PhysicsBehavior>
        <CollisionTreatment>
          <PacManPillCollisionTreatment/>
          <PacManWallCollisionTreatment/>
          <GhostWallCollisionTreatment/>
          <PacManGhostCollisionTreatment/>
        </CollisionTreatment>
      </PhysicsBehavior>
    </SceneBehavior>

    <SceneObserver>
      <CollisionOcurrence>
        <PacManPillCollision/>
        <PacManWallCollision/>
        <GhostWallCollision/>
        <PacManGhostCollision/>
      </CollisionOcurrence>
    </SceneObserver>

    <Spatial ID="GamePlay">
      <SceneNode>
        <HUDNode> ...
<Location X="X_HUD_INIT" Y="Y_HUD_INIT"/> ...
<Model imageFile="twoLifes.png"/> ...
        </HUDNode>
        <WallNode> ...
<Location X="X_INIT" Y="Y_INIT"/> ...
<Model dataFile="scenario.dat" imageFile="wall.png"/> ...
        </WallNode>
        <PillNode> ...
<Location X="X_INIT" Y="Y_INIT"/> ...
<Model dataFile="pills.dat" imageFile="pill.png"/> ...
        </PillNode>
        <GhostNode> ...
<Location X="X_GHOST_INIT" Y="Y_GHOST_INIT"/>
<Model>
  <Image fileName="ghost1.png"/>
  <Image fileName="ghost2.png"/>
</Model> ...
        </GhostNode>
        <PacManNode> ...
<Location X="X_PACMAN_INIT" Y="Y_PACMAN_INIT"/> ...
<Model>
  <Image fileName="pacman0.png"/>
  <Image fileName="pacman1.png"/>
  <Image fileName="pacman2.png"/>
  <Image fileName="pacman3.png"/>
  <Image fileName="pacman4.png"/>
</Model> ...
        </PacManNode>
      </SceneNode>
    </Spatial>

    <Spatial ID="GameVictory"> ...
<Location align="center"/> ...
<Model imageFile="victory.png"/> ...
    </Spatial>

    <Spatial ID="GameOver"> ...
<Location align="center"/>
<Model imageFile="gameover.png"/>
    </Spatial>

  </SceneView>
</GSL>
```

Figure 7: GSL specification of the Simplified Pac-Man.

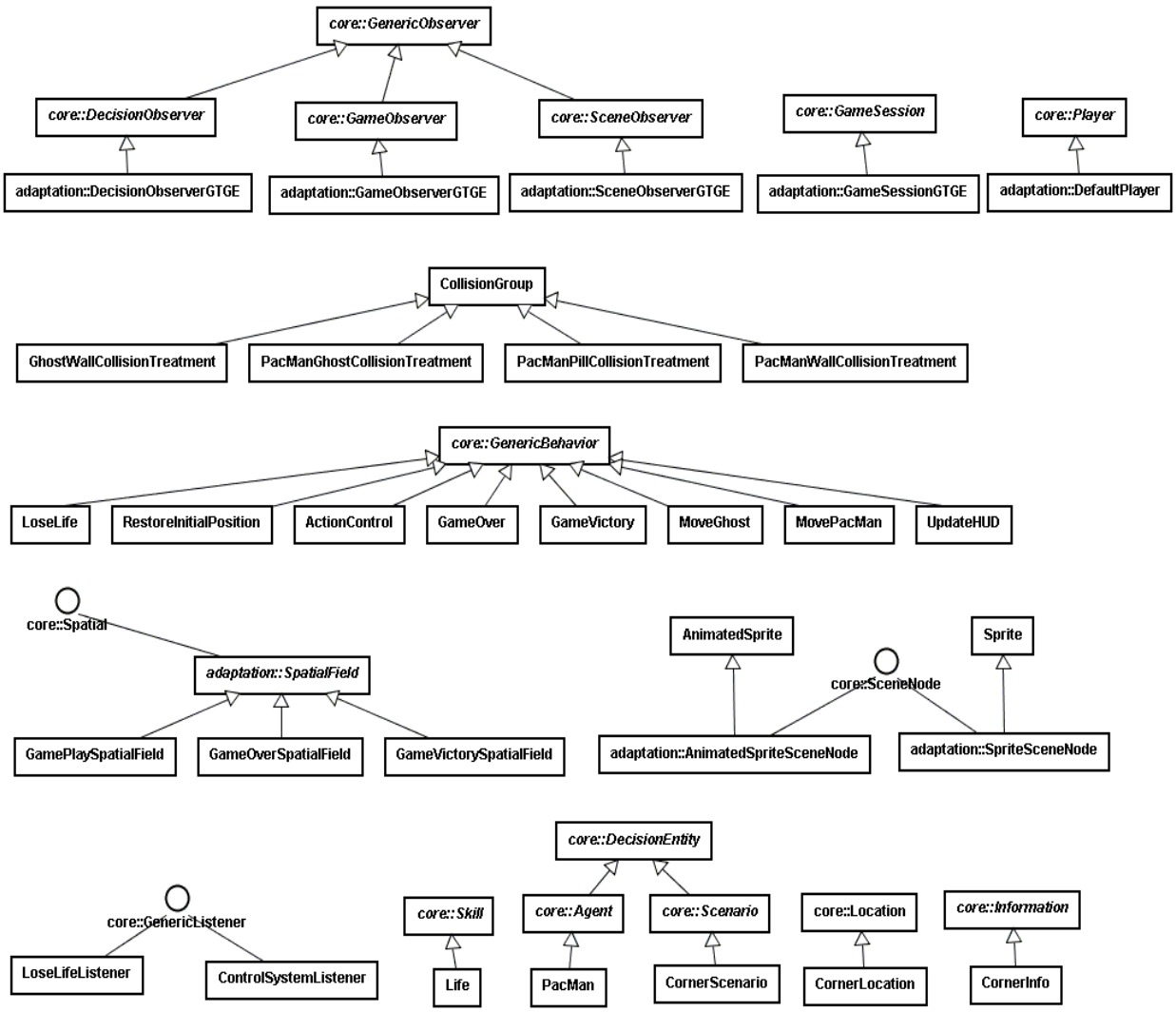


Figure 8: Class diagram with new classes for adaptation, behavior, listener and context packages.

adapter and some context classes like GamePlaySpatialField), *SceneNode* (SpriteSceneNode adapter), *DecisionEntity* (CornerLocation, Life), *GenericListener* (ControlSystemListener), and a lot of subclasses of *CollisionGroup* and *GenericBehavior*, were presented.

4.3 Building Games with XSL Rules

To create these game structures described above (classes and adapters) for the proposed metaframework, a set of metaprogramming scripts, able to translate the GSL specification in a compatible source code, must be created.

These metaprogramming scripts will be defined using the Extensible Stylesheet Language (XSL) format, which can define rules that transform one GSL specification to a desired source code (according section 3.3), like classes and adapters for example.

Some examples of XSL rules, able to translate the GSL specification of the Simplified Pac-Man to a real game, were illustrated in Figure 9. These rules are validated by templates, where, for each template validation, one of these transformations is expected: direct code introduction inside a class for each identified element, internal template validation for each identified element to introduce a code, direct code introduction inside a class, direct introduction of GSL values inside a class, and a complete class introduction in the final code.

As result, after the XSL rules execution described in Figure 9, the following resources will be available to the Simplified Pac-Man game: creation of all declared *Players*, *Keyboard* event treatment, introduction to decision support code for *PacMan*, source code inclusion of the *SessionStatus* value, and the creation of a class to support the *GameOver* behavior.

4.4 Metaframework Integration

During the integration process between a game engine or a game framework with the proposed metaframework, a lot of special resources of the game development artifact can be used, instead of the metaframework structures.

For example, the GTGE framework gives a special collection of classes to work with collision between game objects. In this way, subclasses of the

```
#### Direct code introduction inside a class for each identified element:
<xsl:for-each select="/GSL/GameSystem/GameSession/PlayerContext/*">
    gameSession.addPlayer(new DefaultPlayer(<xsl:value-of select="ID/@value"/>,"<xsl:value-of select="Name/@value"/>"));
</xsl:for-each>

#### Internal template validation for each identified element to introduce a code:
<xsl:for-each select="/GSL/GameSystem/GameObserver/ControlSystem/Keyboard/*">
      <xsl:choose>
        <xsl:when test="name(.)='UpDirection'">
    if (keyDown(KeyEvent.VK_UP)) {
      String[] values = {"UpDirection"};
      gameSession.gameObserver.fire(gameSession, "GameSystem&#47;GameObserver&#47;ControlSystem", values);
    }
        </xsl:when>
        <xsl:when test="name(.)='DownDirection'">
  . . .
</xsl:for-each>

#### Direct code introduction inside a class:
<xsl:if test="/GSL/DecisionSupport/DecisionEntity/ID/@value='PacMan'">
    gameSession.addDecisionEntitiy(GameDecisionData.createPacMan());
</xsl:if>

#### Direct introduction of GSL values inside a class:
gameSession.setStatus("<xsl:value-of select="/GSL/GameSystem/GameSession/SessionStatus/@status"/>");

#### Complete class introduction in the final code:
<xsl:template match="/GSL/GameSystem/GameBehavior/GameCycle/GameOver">
  class GameOver extends GenericBehavior {
    public void execute(GameSession gameSession, Object[] params){
      gameSession.setStatus("GameOver");
    }
  }
</xsl:template>
```

Figure 9: Some XSL rules to transform the GSL specification of the Simplified Pac-Man game.

CollisionGroup like *PacManGhostCollisionTreatment* are presented in the class diagram (Figure 8). But, according the GSL specification (Figure 7), *PacManGhostCollisionTreatment* is a *SceneBehavior*. So, it should be a subclass of the *GenericBehavior* class (defined in the *core* package of the metaframework - Figure 6), like *UpdateHUD* and *MoveGhost* (Figure 8), instead of a GTGE class.

In the same way, none decision information about the Pac-Man position were specified in the GSL specification (Figure 7). This information is directly extracted by available GTGE resources, avoiding the creation of *Information/AgentInfo/PacManPosition* tags for each *Location* in the *Scenario*.

These decisions about what type of game logic or game structure to be used during the generative script production are, in most cases, defined by the "game programmer". The "game programmer" is the person who knows the available game engine or game framework resources. The "game programmer" is the best person who can define the best solution of what types of game structures must be reused or not by game engines or game frameworks. The "game programmer" is the responsible to decide what types of *GameSystem*, *DecisionSupport* and *SceneView* structures will be excluded during the generative process.

4.5 The Final Game

To show the final result of the XSL execution, transforming the proposed GSL specification (Figure 7) in a real game, Figure 10 presents a captured image of the Simplified Pac-Man during its game play.

Although different techniques had been used to develop this final game, like the GDS feature model and the proposed metaframework, some game updates can be performed in a faster way, according the generative game artifact.

For example, to change the initial quantity of lifes available to the player in the Simplified Pac-Man, two simple updates in the GSL specification (Figure 7) are necessary: a new value to the *Life* tag attribute, and a new initial image to the *HUDNode* tag.

As result, the maintainability of the Simplified Pac-Man can be described by single updates on a specific generative artifact used to produce it.

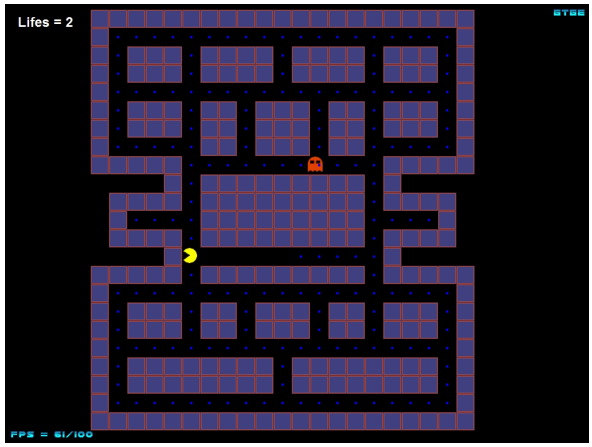


Figure 10: Simplified Pac-Man during the game play.

5. Conclusions

This paper presented a generative attempt to develop computer games in general, using different technologies to complete this task, like: the GDS feature model, the GSL format, the metaframework proposal and some metaprogramming scripts based on XSL format.

The great objective of this work is to describe a different game development experiment, able to increase the level of software reuse and maintainability, allowing the creation and the adaptation of new games by declarative specifications, metaprogramming definitions and metaframework updates.

As a result of this approach, new development strategies can be applied during the game production, according the final game environment (engines, frameworks or platforms, for example), resulting in a low coupling development strategy between game domain and game implementation software artifacts.

For future works, two main researchs will be developed. First, the creation of collections of metaframework adaptations for different game artifacts to be used in a metagenerative approach for games. Second, the definition of collections of game domain resources by the analysis of different game topics, game genres, game platforms, etc, allowing the creation of a repository of game dynamics [Hunicke et al. 2004], using feature models and GSL specifications as a common language.

References

AZANZA, M., TRUJILLO, S. AND DIAZ, O., 2007. Towards Generative Metaprogramming. *In: 2nd Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE 2007), Braga, Portugal, 2-7 July 2007.*

BATORY, D., 2006. Multi-Level Models in Model Driven Development, Product-Lines, and Metaprogramming. IBM Systems Journal, 45(3).

BOOCH, G., RUMBAUGH, J. AND JACOBSON, I., 1998. The Unified Modeling Language User Guide. Addison-Wesley. ISBN-10: 0201571684.

CUTUMISU, M., ONUCZKO, C., CRAWFORD, C., MCNAUGHTON, M., ROY, T., SCHAEFFER, J., SCHUMACHER, A., SIEGEL, J., SZAFRON, D., WAUGH, K., CARBONARO, M., DUFF, H., AND GILLIS, S., 2007. ScriptEase: A generative/adaptive programming paradigm for game scripting. *In: Science of Computer Programming, Volume 67, Issue 1 (June 2007), pp. 32-58. ISSN:0167-6423, Elsevier North-Holland, Inc. Amsterdam, The Netherlands.*

CZARNECKI, K., 2004. Overview of generative software development. *In UPP 2004, volume 3566 of LNCS, pages 326--341. Springer.*

CZARNECKI, K. AND EISENECKER, U., 2000. Generative Programming. Addison-Wesley, ISBN: 0201309777.

CZARNECKI, K. AND KIM, C., 2005. Cardinality-Based Feature Modeling and Constraints: A Progress Report. *In OOPSLA'05 International Workshop on Software Factories (online proceedings).*

FOWLER, M., 2005. Language Workbenches: The Killer-App for Domain Specific Languages?. Available from: http://www.martinfowler.com/articles/languageWorkbench.html [Accessed 07 July 2009].

FURTADO, A AND SANTOS, A., 2006. Applying Domain-Specific Modeling to Game Development with the Microsoft DSL Tools. Tutorial (in English). *In: 3rd Brazilian Symposium on Computer Games and Digital Entertainment (SBGames2006).*

FURTADO, A., 2006. Defining and Using Ontologies as Input for Game Software Factories. *In: Proceedings of the 3rd Brazilian Symposium on Computer Games and Digital Entertainment.*

GAMMA, E., HELM, R., JOHNSON, R. AND VLISSIDES, J., 1994. Design Patterns: Elements of Reusable Object-Oriented Software. *Addison-Wesley Professional*. ISBN 0-201-63361-2.

GTGE, 2009. *Golden T Game Engine (GTGE)*. Golden Studios. Available from: http://www.goldenstudios.or.id/products/GTGE/ [Accessed 07 July 2009].

HAROLD, E., 1999. XML Bible. Wiley. ISBN-10: 0764532367

HUNICKE, R., LEBLANC, M., AND ZUBECK, R., 2004. MDA: A formal approach to game design and game research. *In: Proceedings of the AAAI-04 Workshop on Challenges in Game AI, July 2004., pp. 1-5.*

IWATANI, T. 1980. Pac-Man. Namco.

MORENO-GER, P., MARTÍNEZ-ORTIZ, I., SIERRA, J. AND FERNÁNDEZ-MANJÓN, B., 2005. Language-Driven Development of Videogames: the <e-Game> Experience. *In: Proceedings of the 5th International Conference on Entertainment Computing (ICEC 2005), Cambridge, UK. Lecture Notes in Computer Science 4161, 153-164.*

SARINHO, V. AND APOLINÁRIO, A., 2008. A Feature Model Proposal for Computer Games Design. *In: Proceedings of the VII Brazilian Symposium on Computer Games and Digital Entertainment, Belo Horizonte, p. 54-63.*

TRUJILLO, S., AZANZA, M. AND DIAZ, O., 2007. Generative Metaprogramming. *In: 6th International Conference on Generative Programming and Component Engineering (GPCE'07), October 1–3, Salzburg, Austria.*

W3C, 2009. *World Wide Web Consortium*. Available from: http://www.w3.org/ [Accessed 07 July 2009].

XSL, 2009. *The Extensible Stylesheet Language Family*. Available from: http://www.w3.org/Style/XSL/ [Accessed 07 July 2009].

ZHANG, W. AND JARZABEK, S., 2005. Reuse without Compromising Performance: Experience from RPG Software Product Line for Mobile Devices. *In: Proceedings of the 9th Int. Software Product Line Conf., SPLC'05, Sept. 2005, Rennes, France, pp. 57-69.*

A Model for Interactive TV Storytelling

Marcelo M. Camanho[1] Angelo E. M. Ciarlini[1]
Antonio L. Furtado[2] Cesar T. Pozzer[3] Bruno Feijó[2]

[1]UNIRIO, Dep. Informática Aplicada, Brazil
[2]PUC-Rio, Dep. Informática, Brazil [3]UFSM, Dep. Eletrônica e Computação, Brazil

Abstract

Interactive storytelling systems are applications to generate and dramatize interactive stories. The main challenge to such systems is the generation of coherent stories while users are watching and interfering with what is happening. In an interactive TV environment, quality and diversity of narratives are crucially important objectives. In addition, new requirements related to comfort in user interaction, responsiveness and scalability have to be taken into account. In this paper, we present a model for interactive TV storytelling to cope with these requirements. The model was implemented in a new version of the planning-based interactive storytelling system Logtell.

Keywords: Interactive TV, Interactive Storytelling, Modeling and Simulation, Planning, Multimedia

Authors' contact:
marcelo.camanho@uniriotec.br
angelo.ciarlini@uniriotec.br
furtado@inf.puc-rio.br
pozzer@inf.ufsm.br
bfeijo@inf.puc-rio.br

1. Introduction

Interactive storytelling systems are computer applications for telling stories that can be modified to some extent by their users. While in conventional games stories are essentially used to create challenges for the player, in interactive storytelling applications stories are expected to surprise and entertain. As a consequence, the quality of the stories in terms of coherence and dramatic content must be regarded as a prime concern. Different approaches for interactive storytelling have been proposed and implemented with different goals. Some of them are more directed to games and others to filmmaking and literature. One of the main challenges for the implementation of such systems continues to be the conciliation of a good level of interactivity with the coherence of the stories.

In recent years, traditional Analog TV is gradually being replaced by Digital TV, with higher sound and image quality, and with new interaction possibilities. Besides that, we have seen an ample dissemination of new communication media, such as broadband Internet and 3G mobile phones, which offer competing alternatives for the exhibition of shows that would, in principle, seem naturally fit for TV.

The phrase interactive TV (iTV) has been used both in the context of open Digital TV and other media featuring some kind of interactivity for TV programs. As TV is one of the classical media for telling stories, new possibilities of interaction open great opportunities for the creation of interesting applications. In this context, however, certain requirements tend to become even more essential, such as quality, coherence and diversity of stories, as well as the need for comfortable and simple interaction methods. It is necessary to find hybrid means for presenting stories on iTV, mixing features of games and conventional TV. On the one hand, the appeal of TV programs for regular spectators has to be maintained, but, on the other hand, various ways of interacting with the medium should be provided. And one must consider not only the case in which users want to actively intervene in a story, but also the case in which they just want to watch TV without being called to interact by any means.

When spectators watch a film on TV, their satisfaction is directly related to the quality of the story, and coherence is a crucial issue for a good story. Interaction methods should not violate coherence and, at the same time, should facilitate variation, so that the user does not get tired of watching the same story over and over again, as happens in a number of games.

TV is a medium which demands high responsiveness, that is, the satisfaction of users' expectations without compromising the quality of service. When watching TV one is not pleased, for instance, with an excess of interruptions during the presentation of a movie. In addition, programs are watched by a huge amount of people. Hence, adequate responsiveness requirements related to presentation flow and scalability have to be taken into account when we think about interactive TV storytelling.

In this paper, we describe a new model for controlling interactive TV storytelling processes which has been incorporated in the implementation of Logtell 2, a second version of the logic-based interactive storytelling system Logtell [Ciarlini et al 2005]. In the model, plot generation, user interaction and dramatization occur in parallel, and we strive to conciliate the requirements of different levels of interaction with coherence and diversity of stories, aiming at the high responsiveness demanded by the medium.

The paper is organized as follows. Section 2 gives a brief state of the art survey of storytelling systems and iTV. Section 3 presents the proposed model, and section 4 describes its implementation. Section 5 contains concluding remarks.

2. TV and Interactive Storytelling

Since the 1990's we have seen more and more a process of digital convergence, which has brought

together many improvements and innovations in different areas, such as the infra-structure of communication networks, compression software and hardware, data transmission and broadcasting services [Furht 1996]. As a result of this convergence, new possibilities for TV broadcasting emerged such as open, satellite and cable Digital TV, Mobile TV, Web TV and IPTV. These developments have enhanced the experience of watching conventional TV. The first obvious benefit has been the improvement in image and sound quality, but changes have also been seen in the content that is available to spectators [Driscoll 2000]. Interactivity has scarcely begun to be explored, but it is already clear that it can radically change the way people watch TV. In particular, in developing countries, where TV has a much wider penetration than the Internet [CPqD 2005], interaction in an open digital TV environment has practical relevance, because it can significantly increase the access of a large part of the population to Education, Culture and Entertainment [Sancrini 2005].

Many opportunities for interactivity with users are possible, making it a topic of growing interest for research and development, both in industry and academia. There is, however, no consensus on how interaction with TV should happen. Possibilities of interaction depend on the computer power of the set-top boxes that receive and process the signal. Some specialists support the idea of *lazy interactivity*, with simple set-top boxes, by means of which the user has more limited options, but minimal effort and attention are demanded. Some others favor more powerful set-top boxes, but with easy and intuitive interfaces, so that users are able to watch TV and interact in a more active way. The specific features of set-top boxes that will prevail are not yet clear. It is quite possible however that there is space for both approaches. Anyway, it can be expected that the degree of interactivity will increase as interactive TV environments evolve. Interactive TV, or simply iTV, is a generic term that covers an ample set of possibilities of increasing sophistication, including:

- a weaker interactivity that corresponds to watching shows at a desired schedule, skipping ads, executing VCR commands, obtaining more information about what is shown (e.g. movies and news), etc.;
- directed and individualized advertising, together with sales and marketing;
- direct interaction with the presented content, changing, for instance, the ending of a story that is being watched; and
- the continuous interaction of a group of users with a shared content.

The first two items correspond to most of the new applications we have seen in recent years. The last two items are closely related to storytelling and tend to demand more research, but they have a strong appeal for the development of new applications that focus on Entertainment and/or Education. TV is a classical medium for telling stories in various formats, such as films, soap operas, cartoons and documentaries. Many possibilities for adapting these kinds of TV programs can be tried in order to incorporate interactivity in effective ways. Some experiments have already been made, but finding engaging formats that allow users to fully explore interactivity remains an open issue.

Some experiences of interactivity with TV content have already been carried out, even in conventional analog TV, although in an improvised and rigid way. An example is provided by reality shows, where spectators can make decisions by means of votes submitted by phone or via Internet.

Interaction with the content that is being presented is more complex than the other possibilities of interaction. It demands more sophisticated interaction methods and some kind of standardization of set-top boxes. Moreover, practical business concerns have to be addressed, because the user will have much more control on what is presented.

Projects like *ShapeShifting Media* [Ursu et al 2008] propose new forms of interactivity with TV content in opposition to the forms that have already been incorporated in interactive Digital TV environments. The project works with narrative models and some interesting applications have been developed as part of the project, such as *My News & Sports My Way*, in which the content of a continuous presentation of news is recombined in accordance with users' interest, and the romantic comedy *Accidental Lovers*, in which users can watch at real-time and influence a couple's relationship.

Despite the existence of some projects that tackle the interaction with TV content up to a certain extent, the dynamic creation of interactive stories at real-time for TV is still an open research issue. The mass production of coherent, diversified and engaging stories that can be influenced by users, in a comfortable way, is not a trivial task.

Interactive Storytelling has evolved as an interdisciplinary research area, involving Games, Filmmaking, Literature, Psychology, Cognitive Science and various fields of Computer Science, such as Computer Graphics and Artificial Intelligence.

Some approaches for Interactive Storytelling are classified as character-based [Cavazza 2002] because they focus on modeling characters as autonomous agents. In these approaches, stories emerge from the interaction between the characters. When this approach is adopted, it is easier to implement direct interaction with the characters, but harder to keep the story coherent. Other approaches are classified as plot-based [Grasbon and Brown 2001; Paiva et al 2001; Spierling et al 2002], since they focus on plot structure. They are directly influenced by the work of the Russian literary theoretician Vladimir Propp in his seminal work on the fairy-tales genre [Propp 1968]. In plot-based approaches, keeping the coherence of stories is easier, but opportunities of interaction are rather limited. Few attempts exist to combine both approach. Façade [Mateas and Stern 2003], for instance, keeps the characters' autonomy most of the time, but their goals and their behavior can be changed by a drama manager

to move the plot forward. Genres that stress realism typically demand more coherence; on the other hand, in various genres, a free direct interaction between characters might result in a more engaging experience. In general, the right approach depends on the goal of each application and the genre of stories to be generated and told.

To create stories, a promising strategy is the use of automated planning algorithms, as in [Riedl and Young 2004], allowing to explore alternative ways whereby a logically connected chain of events could achieve the goals of the characters and/or those of the story. Diversity and coherence can be thus conciliated, but interaction must be constrained so as to limit the stories to those acceptable to the algorithms. In [Cavazza 2002], hierarchical task network (HTN) planning is used to control the way characters achieve their goals in accordance with user intervention. HTN planning tends to be efficient but less general, requiring the previous construction of a task hierarchy and methods to perform each task. In Façade**,** a reactive planning language is used to emulate the personality of believable agents. In Mimesis [Young 2001], a planner combining HTN and partial-order planning is used to create a storyline beforehand. Techniques of mediation are used at run time to guarantee coherence, including the adoption of alternative story lines or interventions for forcing the failure of users' actions.

Logtell is an interactive storytelling system based on logical modeling and planning-based simulation, which also tries to conciliate plot-based and character-based features. The main difference of the approach adopted in Logtell from other planning-based interactive storytelling systems is the goal of the system. Instead of working on alternatives for an entire story line, Logtell seeks to generate a maximum of different and coherent stories of a certain genre along multiple *simulation stages*, combined with user intervention. A formal model for capturing the logics of the story genre is specified to determine the scope of coherent alternatives. When plots are fully or partially generated, they can be dramatized via an animation of virtual actors in a 3D scenario.

Research on Interactive Storytelling may hopefully provide the basis for creating good models. The model proposed in this paper for interactive TV storytelling processes uses the original model of Logtell as a starting point, but also takes into account the specificities of a medium where high responsiveness is demanded and a majority of users may still prefer to assume a more passive behavior.

3. Proposed Model

The approach for interactive storytelling we have adopted assumes a third person viewpoint. We also assume that the conventions of the story genre can be logically modeled. The basic idea is to let the user interact with the story as if he or she were a "deus-ex-machina", with the power to choose alternatives for the future, cause the story to backtrack to previous points and try to force the occurrence of events and situations. User interventions have however to preserve coherence with the logical model. Interventions that do not make sense are rejected, but those that are found to be logically compatible can be incorporated, generating consequences to the rest of the story. In this way, the approach can be seen as an extension of the experience of watching a film on TV.

Interactions can vary from a level in which the user just watches a story as in conventional TV to a level of strong interventions in which the user is enabled to explore the possibilities allowed by the story genre. Although this ability was already provided by the first version of Logtell, the system still remained essentially a tool for logically modeling and simulating a story world obeying the rules of a genre. Varied and coherent plots could be generated with user intervention, but dramatization occurred only after the generation of the plot and there was no user intervention during the dramatization. The model proposed here uses the original model of Logtell as a starting point but modifies it in several ways to make it compatible with an iTV context. In order to do that, the model seeks to fulfill the following requirements:

I. It should be possible to create diverse stories, all of them coherent and resulting from interactions with the users.
II. Presentation flow must be continuous, that is, plot generation, user interaction and dramatization should occur in parallel without delays.
III. Comfortable and simple interaction methods at various levels should be provided so that different kinds of users can enjoy the experience.
IV. Users should be allowed both to interact with stories as single users and to share the control of stories with other users.
V. The underlying architecture should be scalable, in view of the massive nature of the medium.

In this section we present the basic architecture proposed by the model and then we discuss how the requirements listed above can be fulfilled by a system implementing this architecture.

3.1 Architecture

Figure 1 presents the client-server architecture proposed by the model. The client-side is responsible for user interaction and dramatization of stories. At the application server side there is a pool of servers sharing the responsibility of creating and controlling multiple stories, which are presented in different clients. This takes care of the case wherein multiple users are simultaneously sharing the same story. If clients are set-top boxes for interactive TV, their computational resources are limited, making it hard to perform CPU-intensive tasks such as automated planning. By concentrating simulation tasks in application servers, it is easier to achieve higher scalability. In addition, it is also easier to exert control when a single story is shared by many users.

The access of all modules to the context of the stories, specified in the Context Database, is always performed via the Context Control Module (CCM), which runs in

the server. The context contains the description of the genre according to which stories are to be generated, and also the intended initial state specifying characters and the environment at the beginning of the story. The genre is basically described by: (a) a set of parameterized basic operations, with pre- and post-conditions, logically specifying the predetermined repertoire of events that can occur; (b) a set of goal-inference rules, specified in a temporal modal logic formalism, which define situations that lead characters to pursue the achievement of goals; (c) a library of typical plans, corresponding to typical combinations of operations for the achievement of specific goals, which is organized in "part-of" and "is-a" hierarchies; (d) logical descriptions of initial situations for the stories, introducing characters and their current properties; (e) a nondeterminisc automaton for each operation, specifying alternative ways whereby the event associated with the operation can be dramatized; and (f) graphical models of 3D virtual actors.

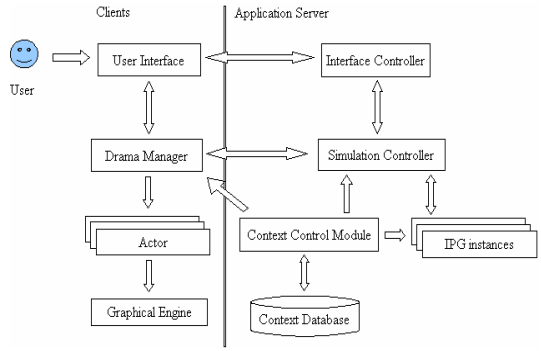


Fig. 1. Basic Architecture for iTV Storytelling

Plot generation is performed by the Interactive Plot Generator (IPG) [Ciarlini and Furtado, 2002]. IPG generates plots as a sequence of chapters. Each chapter corresponds to a cycle in which, subject to user interference, goals are inferred and planning is used to achieve the goals. IPG is controlled by the Simulation Controller. Multiple stages, each one corresponding to a chapter, usually occur in order to generate a plot. In case there is no user intervention, goals are inferred and events are planned continuously. Logical coherence of a requested user intervention is always checked before being incorporated, or else discarded if inconsistent.

The Simulation Controller is responsible for informing the Drama Manager, at the client side, the next events to be dramatized; receiving interaction requests and incorporating them in the story; selecting viable and hopefully interesting suggestions for users who are intent on performing strong interactions; controlling a number of instances of the Interactive Plot Generator, in order to obtain the next events to be dramatized; and controlling the time spent during simulation.

The Simulation Controller also keeps snapshots of the state of the simulation after the generation of each chapter, so that simulation can be resumed from any previous chapter of the story. In this way, intervention can be used to force a story to return to a previous point, from which alternative continuations can be tried. Snapshots are also important to allow different servers from a pool to deal with the same story, thus enhancing scalability. When the simulation of a story in about to be continued, the previous snapshot can be recovered from the database and the process can be resumed by any server available.

On the client side, the user interacts with the system via the User Interface, which informs the desired interactions to the Interface Controller placed at the server side. The Drama Manager requests the next event to be dramatized from the Simulation Controller, and controls actor instances for each character in a 3D environment running on the Graphical Engine. On the server side, the Interface Controller centralizes suggestions made by the various clients. When multiple users share the same story, interactions are selected according to the number of clients that requested them. When there is only one client, suggestions are automatically sent to the Simulation Controller.

The architecture in Figure 1 avoids the transmission of video on demand during the storytelling process, a precaution that ought to be taken in order to allow many simultaneous stories without compromising bandwidth. All data necessary for 3D dramatization on the client side can be transmitted before starting the simulation. During the process, the transmitted data is restricted to information about user interaction, the indication of events to be dramatized and synchronization commands. This strategy assumes however that there is enough computational power at the client side to generate a 3D animation. In another scenario, with many users sharing a limited number of stories but having very little computational power, the architecture can be modified to have a Drama Manager for each story running on the server-side. Each Drama Manager would then generate video to be broadcasted to the users that share the corresponding story.

In the sequel, we discuss the main strategies to control interactive storytelling processes in the described architecture.

3.2 Coherence and Diversity of Stories

Coherence and diversity of narratives are key factors to the success of any interactive storytelling application for iTV. If stories do not seem plausible and coherent with respect to the genre, the user may lose interest for the experience. If generated stories have little variation, their ability to entertain and surprise tends also to be reduced. After a few trials, users would be ready to discard the application.

Balancing coherence and diversity of stories with interactivity is a difficult challenge. Too much interactivity can easily hinder the coherence of the story. Too little interactivity on the other hand would reduce the variation in stories and the impact of the experience.

Conciliation of coherence, diversity and interactivity can be achieved by means of hard-coding various coherent alternatives. Façade, for instance, as

previously mentioned, is among the most successful interactive storytelling applications, allowing users to enjoy an interesting interactive experience for about 20 minutes. The structure of the system demanded however a huge authorial effort (with thousands of lines of code) to model a single dramatic situation with different possible outcomes. Another problem with this kind of solution is that authors lacking the required kind of programming expertise may find difficult to directly model situations as part of the application code. For the mass production of interactive storytelling contexts, as would be necessary for iTV, other solutions have to be sought.

The strategy that we propose is the construction of a logical model for the genre and the use of planning and inference of goals to guarantee coherence while exploring diversity. Plot generation starts by inferring goals for the various characters (and for the story as a whole) from the initial situation. Given this initial input, the system uses a planner that inserts events in the story-plot in order to fulfill the goals. When the planner detects that all goals have been either achieved or abandoned, the first chapter of the story is finished. If the user does not like the story, IPG can be asked to generate a different alternative for a chapter and to develop the story from this point on. If the user does not interfere in the process, chapters are continuously generated by inferring new goals from the situations generated in the previous chapter. If new goals are inferred, the planner is activated again to fulfill them.

The process thus alternates goal-inference and planning until the moment the user decides to stop or no new goal is inferred. Users can also interfere in the process by choosing alternatives and forcing the occurrence of events and situations as described in section 3.4. Notice that, in this process, we mix forward and backward reasoning. In the goal-inference phase, we adopt forward reasoning: past situations generate goals to be fulfilled in the future. In the planning phase, an event inserted in the plot for the achievement of a goal may have unsatisfied pre-conditions, to be handled through backward reasoning. To establish a pre-condition, the planner can insert previous events with further unfulfilled pre-conditions, and so on. The planner used in IPG is a partial-order planner, adopting a least-commitment strategy to more easily accomplish the conciliation of different goals. Constraints (including the order of events) are established only when necessary, and all possibilities for solving conflicts between events in the establishment of pre-conditions are considered.

IPG provides a base for virtually creating any plot compatible with the rules of the genre. At each stage, the user can reject the alternative being currently presented and ask for another, or may opt for a direct intervention. And whenever the user intervention is compatible with the genre, IPG provides means for adapting the story so that the user's contribution can be incorporated. In this way, the adopted approach aims to provide coherence and diversity by construction. Authorial effort is still necessary to formally model the interactive storytelling context, but this is inevitable in the creation of any interactive or conventional story. The difference is that, thanks to the plan-based support described here, plots need not be devised beforehand by the author.

3.3. Continuous Presentation Flow

When spectators watch movies on TV, events are presented continuously. This is not a difficult task because the whole story is generated and filmed beforehand. In iTV this is not the case, but, if iTV storytelling purports to be an extension of the experience of watching conventional films, the presentation flow should likewise be continuous.

A premise of our model is that users should feel that they can change, to a significant extent, the story being presented according to their will. The continuous presentation of a story that is modified by user interventions is a challenge, even if dramatization is performed by means of 3D animations instead of real actors. If we want to reach the same level of coherence of a conventional story, plots have to be continuously adapted to incorporate user interventions. Checking the coherence of an intervention at real-time and computing the possible consequences of the intervention to the rest of the story is not trivial and may become excessively time-consuming. Some kind of synchronization between plot generation, user interaction and dramatization is then mandatory.

In order to synchronize the processes, narratives are divided into chapters. While a chapter is being presented to the user, IPG can already start generating the future chapters. When user interventions are coherent, they are incorporated in the next chapter. In this way, we try to keep plot generation some steps ahead of the dramatization, so that chapters are continuously generated and dramatized. The main problem occurs when a user intervenes in the story, trying to force a situation or the occurrence of an event. Since user interaction affects the situation of the chapter currently being presented, future chapters previously generated without taking the intervention into consideration would no longer be useful. The difficulty is that, as the next chapter has to be ready before the end of the dramatization of the current chapter, there is a risk of interruption in the presentation flow. The following strategy is applied, with two options. When the Simulation Controller detects that more time is needed for generating the next chapter, a message is sent to the Drama Manager to the effect that the duration of the remaining events in the current chapter will be extended, as detailed in [Doria et al 2008]. If there is no way to extend the chapter being presented until the next chapter is ready, the user intervention is discarded, as if it were inconsistent. In this case, the chapter that had been generated without incorporating the user intervention is used.

An alternative to reduce the number of times when coherent user interventions are rejected is the use of other instances generated by IPG, besides those corresponding to the current flow of the stories. Such instances can be used to try to anticipate the effects of possible user interventions, so that future chapters will

be ready when necessary. In a continuous presentation of the story, strong user interventions are based on suggestions given by the system. The Simulation Controller is then aware of possible user interventions, and IPG instances that incorporate each one of them can be started.

Due to the combinatorial complexity of automatic planning, the most time-consuming part of the simulation corresponds to planning events to reach inferred goals or goals imposed by the user. In the simple scenario used to test our model, the current planning algorithm adopted by IPG has been able to generate chapters in due time. However, in order to maintain responsiveness in more complex scenarios, it may be necessary to enhance the performance of our planner. The possibilities envisaged include combining the current planner with graph planning techniques [Geverini and Serina 2002], and resorting to heuristics and control strategies [Hoffman 2001] and/or HTN techniques [Nau et al 2003].

3.4. Interaction Methods

In an iTV storytelling application, the user should be able to easily interact with the story. Interaction must never disrupt the user's immersion in the story, exactly as one expects from conventional TV. In opposition to what occurs in various games, the user's ability to quickly react is not so critical, because the effort involved in interacting with a story is basically intellectual, rather than physical.

Interaction methods have also to take into consideration the different kinds of prospective users of the application. Some users may want to essentially remain as spectators, willing to interact very little with the story. Others would be prepared to continuously intervene in the story, actively determining the way the plot unfolds. It is then necessary to provide more than one method, to accommodate different levels of intervention in the stories.

Our model offers the possibility of both weak and strong interventions in the story. By means of weak interventions, the user can select alternatives that are automatically generated by IPG. Strong interventions are used to try to force the occurrence of events or specific situations. The window in Figure 2 shows the current version of our tool's interface, through which users interact with the story being dramatized as displayed in the main window. The interface has certainly to be improved and adapted to devices other than a desktop computer, but has already served to check the viability of our initial set of interaction mechanisms.

Chapters are continuously generated and presented in the main window. When a chapter is being presented, a new line corresponding to that chapter is inserted into the list box *Chapters*. The description in natural language of a selected chapter appears in a text box. Weak interventions occur by means of the commands *Rewind* and *Another*. In order to execute such commands, the user has only to select a chapter and press the corresponding button.

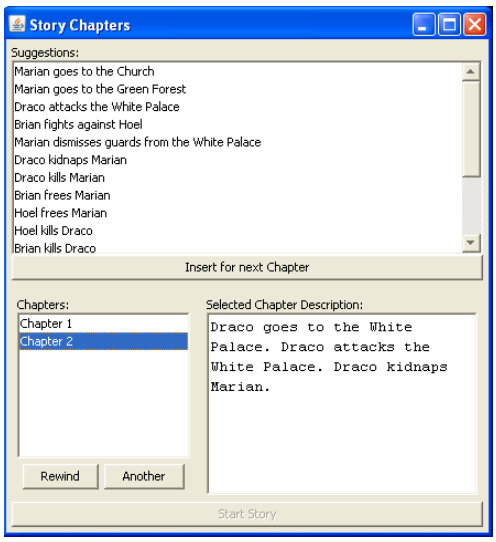


Figure 2: Window for continuous interaction.

The *Rewind* command was so named by analogy with the rewinding fuction of Video Cassette Recorders, but, in fact, it is considerably more powerful. By executing this command, we allow the user to "return" until the time the selected chapter was being presented. The chapter is presented again and the user has the opportunity of interacting with the system and checking alternatives for the following chapters. When this command is executed, the Simulation Controller retrieves the snapshot corresponding to the storytelling process at the time the chapter was presented and resumes the simulation from this point, discarding the snapshots of the next states, which will be generated again in accordance with the user's interactions.

Command *Another* is used to ask the system to provide an alternative for the selected chapter. It is similar to command *Rewind*, in that it also involves a return until the time the selected chapter was being presented. In response to the command, IPG generates another solution for the goals that were reached. In this way a different combination of events can be generated for the chapter, whereby a completely different continuation of the story can be developed.

Commands *Rewind* and *Another* demand a moderate mental effort from the user and are quite intuitive. They are useful when users want to explore other alternatives for the story without having to assume a more active participation in the plot generation process.

In contrast, strong interventions correspond to the specification of events and situations that should occur in the next chapter. Situations are considered as goals to be achieved at a certain time, and events can have unfulfilled preconditions that might demand the insertion of more events. In such cases, IPG has to plan a chapter with additional events and constraints that make the user intervention consistent with the plot and the rules of the genre. If this is not possible, the user intervention is simply rejected. In a continuous presentation, the specification of events and situations from scratch might impose an excessive burden to the user. To make the process simpler, the model proposes

a mechanism in which viable strong interventions are *suggested* to the users, so that they can simply select the one that better suits their taste and press a button. The list box *Suggestions* in the interface contains suggestions from which the user can select at a given time. The list is updated whenever the presentation of a new chapter starts, so that only meaningful suggestions are presented.

The Simulation Controller is responsible for generating suggestions of strong interventions. Suggestions should be consistent and lead the plot towards different outcomes. The methods below are proposed by the model to obtain meaningful suggestions:

- The first method corresponds to the specification by the author of rules for the inference of suggestions. Such rules are specified in the same temporal modal logic used to describe goal-inference rules. The Simulation Controller evaluates all rules in accordance with the context of the current chapter and collects suggestions that would make sense. Suggestions obtained in this way can be quite helpful to create an interesting set of options for the users. This method demands however an additional authorial effort.
- A second method uses a library of typical plans organized in a hierarchy of events. Typical plans usually consist of certain combinations of events whereby the various characters pursue their goals, but they can also correspond to *motifs*, i.e. recurring structures compiled in the course of critical studies on the genre [Polti 1945]. IPG contains a procedure for the recognition of plans, based on an algorithm specified by Kautz [Kautz 1991]. The procedure is able to discover that some given events are compatible with a motif for which we have a typical plan, enabling the Simulation Controller to suggest the inclusion of additional events contained in the plan.
- A third method corresponds to an analysis of goal-inference rules in face of the current plot. If the system detects that a goal-inference rule will be triggered if a certain situation is verified in the next chapter, this situation can be selected as an interesting suggestion.

Interactive TV is still a novel environment. A model for iTV storytelling has then to be open enough to incorporate new kinds of interaction. With this in mind, the model considers the possibility of incorporating other methods for weak and strong interventions, such as letting users insert abstract events and situations in the story, which are automatically specialized by the simulation process; tune narrative tensions by means of numeric scales referring to levels of violence, romantic turns, etc.; and communicate with the system by entering phrases in (a restricted subset of) natural language.

Besides the support for real-time generation and dramatization of stories, it is important to support the authorial effort. The original version of Logtell already worked in a step-by-step mode, in which the partial plot after each chapter was presented as a graph and could be inspected in detail. In this mode, weak and strong interventions are also possible and dramatization for the plot generated so far has to be explicitly activated by the user. The user could even analyze the generation of the whole plot and activate dramatization only at the end. The present model considers that the continuous interaction mode previously presented should coexist with the step-by-step mode.

3.5. Sharing Stories

A most important iTV storytelling mode of application is the creation of stories that are influenced by a great number of users. Weak interventions, via commands like *Rewind* and *Another*, are not so appealing when multiple users are watching the same story. More attention should then be directed to interaction methods enabling strong interventions.

Criteria on how to deal with different interventions are necessary. A first possibility is to consider that, when users ask the system to incorporate a specific suggestion, they are only voting for the suggestion. The most voted suggestion would then be chosen to be incorporated.

The Interface Controller organizes the interaction with clients and interacts with the Simulation Controller as if there were a single user. In order to do that, it coordinates the simultaneous dramatization and the presentation of suggestions for strong interventions in the various clients. It also controls the time during which users' choices are considered. After computing the most voted suggestion, the Interface Controller checks whether the number of votes reaches a minimum threshold. If this is the case, the suggestion is sent to the Simulation Controller. However, the selection of strong interventions does not have to be limited to considering only the most voted intervention. Other possible strategies are:

- The number of votes can be weighted by the potential of each option to trigger goal-inference rules. In this way, options that generate more interesting situations tend to be chosen.
- Compatible interventions can be combined in the same chapter. In particular, different groups of users may have different options. They can, for instance, be distinguished by the characters the group components decide to support. IPG would then try to combine the choices of all groups.

When the possibility of influencing the story by tuning narrative tensions is admitted, the numeric scales can be controlled by the average of the values assigned by the users.

The way groups are formed to share a story is also an issue to be resolved as part of the implementation of the model. A simple solution is to assume that any user is allowed to schedule the start of a story based on a specific context at a certain time. As other users notice one such story in a list of scheduled stories, they may then be tempted to join the group. Users can either have equal rights to intervene in the story or not; in the

latter case, different criteria can be established to assign their rights and privileges. Methods for the communication among users who share the same story can also be devised, so that they would be able to discuss their interventions.

Other alternative methods and criteria for multiuser interaction can be examined. The model proposes a platform to experiment with them in order to test their feasibility, and determine which ones are more engaging for the audience.

3.6. Scalability

Due the massive aspect of the medium, an iTV storytelling system has to cope with the possibility that a huge number of people might be using the application at the same time. They can use the system simultaneously to generate stories in different contexts; they can share a same context to obtain different stories; and they can simply share stories. As TV spectators are not used to experiment unexpected delays, the scalability of an iTV storytelling application is crucial.

In our model, plot generation consumes considerable computational resources in terms of CPU time and memory. In order to avoid the creation of a bottleneck, IPG and the Simulation Controller can run in multiple servers. Snapshots of each story can be temporarily stored in a database, so that any available server can restore a specific snapshot and generate the next chapter for the corresponding story. In this way, responsiveness can be maintained by simply increasing the number of servers in the pool. Additional servers can also be considered to avoid bottlenecks in the access to the database and in the communication with clients.

Regarding scalability, another important issue to be taken into account is that users might want to interact with the application via different platforms and with different local computational resources. In particular, if local resources are very limited, the execution of the dramatization in the server tends to be mandatory. For this purpose, it may be necessary to adopt a hybrid architecture in which some clients can dramatize their own stories locally, and other clients can only share stories dramatized and broadcasted from the server.

4. Model Implementation

In this section, we explain the main issues and the options adopted for the implementation of Logtell 2, a new version of Logtell that incorporates the iTV storytelling model described in section 3.

4.1 Application Environment

Logtell 2 was built utilizing a modular architecture, employing different technologies appropriate to its intended functionalities. The User Interface, the Interface Controller and the Simulation Controller modules were implemented in Java. The Drama Manager maintains the original 3D engine implemented in the first version of Logtell, coded in C++. The Drama Manager communicates with the parts of the system implemented in Java via a DLL that provides an interface connection with the C++ code. The IPG planner is implemented in a version of SICStus Prolog supporting Constraint Programming. The Simulation Controller manages plot generation by accessing IPG via a Java bean that uses native C++ calls to communicate with the SICstus Prolog interpreter.

One of the main requirements for the implementation of Logtell 2 was the use of a client-server environment designed for availability and scalability. Since the application is mostly Java-based, given that both the interface and server side code is implemented in this language, a popular J2EE application server was used, namely JBoss [Marrs and Davis 2005]. JBoss provides a good set of facilities such as distributed services, security, support for asynchronous messages, remote proxies, database access, Web Services and HTTP servers, all desirable for a complete system for interactive TV. Regarding scalability in particular, JBoss makes easier the construction of a pool of servers to provide services to a great number of clients.

It is also important to notice that, since the JBoss architecture handles Web Services, different ways of accessing Logtell 2 can be provided. Since all services were codded using the Enterprise Java Beans 3.0 standard, under the form of Stateless Session Beans, their conversion into Web Services becomes practically automatic, thus making it possible to use mobiles among other ways of access.

4.2 Logtell 2's Modules

The effort for constructing a distributed iTV storytelling processor mainly involved the implementation of the User Interface, the Interface Controller and the Simulation Controller modules.

The services provided by the application servers in Logtell 2 follow the EJB standard. The Simulation Controller was implemented as a Stateless Session Bean. The logic control tasks for the creation of stories have been assigned to distinct submodules: generic services to manipulate the story are handled by StoryManagerService, while the generation of the story-plot via IPG is performed by subclasses of the AbstractStoryWriter: one for continuous stories (ContinuousStoryWriter) and one for stories generated in step-by-step mode (StepByStepStoryWriter)

The StoryManagerService centralizes the services of updating and retrieving stories and their respective chapter snapshots, using a database abstraction in the form of Database Abstract Objects (DAOs). Stories are iteratively generated. Whenever a new simulation cycle is requested, the service restores the previous story snapshot and then proceeds to write another part of the story. In continuous mode, the code that prompts story generation organizes the plot composition in chapters, wherein the total order of the events is established automatically, obeying partial-order constraints established by IPG. In step-by-step mode, the ordering is determined by the user on the graphic interface.

In the continuous mode, there is an instance of ContinuousStoryWriter for each story that is being

generated on the fly. The ContinuousStoryWriter works as if it were a "live screenwriter". For that, it manipulates a StepByStepStoryWriter instance, keeping control over the story that is being generated. The ContinuousStoryWriter implements the strategy of always being ahead of what is being watched by (one or more) users. When, in the continuous mode, a chapter is requested by the client, the server side checks whether this is the most recently created. In this case, to avoid an interruption to the story flow, a message is sent to the corresponding ContinuousStoryWriter of that story, requesting the generation of the next chapter.

The Simulation Controller is also responsible for providing the new forms of strong intervention in continuous mode, which are based on suggestions of strong interventions. Strong interventions are only incorporated if IPG validates them as coherent. In addition, the chapter incorporating the intervention has to be generated before the end of the dramatization in all clients that are watching the same story.

The multi-user story generation process is very similar to regular continuous mode. The main difference is that multi-user stories do not start instantaneously. They are scheduled to start at a certain time by one user. At any time, a user can inspect the set of scheduled stories and join the group of users that will watch and interact with the story selected. At the scheduled starting time, the story dramatizations start in all clients and are synchronized from this moment on. For the time being, strong interactions provided by different clients are chosen to be incorporated only on the basis of the most-voted strategy.

The User Interface contains submodules that implement the interface with users in continuous mode and in step-by-step mode. In step-by-step mode, the User Interface communicates directly with the Simulation Controller. In continuous mode, it communicates with the Interface Controller. The Interface Controller is in charge of centralizing the interaction (in case of multi-user interaction), redirecting user interventions to the Simulation Controller and synchronizing the list of suggestions of strong interventions in the clients.

In the current version of Logtell 2, the Drama Manager and the IPG planner have received relatively minor modifications with respect to their original versions. In IPG, modifications were introduced to allow the Simulation Controller to save and recover story snapshots after the generation of each chapter. The evaluation of rules for inferring promising strong interventions was the other extension of IPG incorporated in this version. The Drama Manager remains essentially the same module of Logtell's first version, but another version is presently under development, which uses a nondeterministic dramatization model to allow the system to control the duration of events and provide different dramatizations for the same event [Doria et al 2008]. In order to provide better quality in 3D animations, the current 3D engine is also being replaced by a module that controls characters in the UNITY 3D game engine.

4.3 Using the Prototype

In order to evaluate the prototype, we utilized the same storytelling context adopted in [Ciarlini et al 2005] where the first version of Logtell was described, with minor modifications. The context corresponds to a small sub-class of the popular *Swords and Dragons* genre. In this context, the events that can occur correspond to *attacks to the opponent's home, fights between characters, kidnapping of a victim, liberation of a victim, charms, weddings*, etc. Goal-inference rules used in this context establish relations between situations and goals, such as: *if the villain is strong, the hero wants to become even stronger*; *if a victim is kidnapped, a hero will want to rescue her*; etc.

The prototype was applied to generate stories in this context, using machines connected via a local network. When stories were generated and dramatized in continuous mode, without user intervention, there was no perceptible interruption (less than 200 ms) between the time a chapter is over on the client and the time the following chapter starts. Chapters were generated on the server with more than enough time left while the user was watching the previous chapters. In situations where network problems could arise, there might be a chance that the user's experience would be affected; fortunately this is not likely to happen, since the amount of information that is sent to the client is very small in the current way the system is designed.

Tests were also performed using the forms of interaction specific to the continuous mode. When executing the command *Rewind*, the time to resume the story at the indicated chapter was also insignificant (~ 350 ms). The execution of command *Another* takes a little more time (~ 5 seconds), because it demands not only the recovery of a previous context, but also the generation of another solution for the chapter, but the time consumed still seemed quite acceptable. In general, it was verified that, under ideal conditions, the results were satisfactory for the continuous presentation flow when using weak interventions. By "ideal conditions" we mean a situation where the computational resources in the application environment are not overloaded.

When using strong interventions, there were also no additional delays compared to the tests where the user watched passively. This happens because, in continuous mode, interventions are incorporated only when the server still has time to prepare the next chapter. In the worst case, the intervention is ignored but no interruption occurs. In most occasions, the dramatization time of a chapter showed to be long enough to allow the incorporation of coherent interventions.

Regarding the diversity of the stories, it was possible to obtain in continuous mode most of the stories that a user could obtain in step-by-step mode. The presentation of meaningful suggestions for strong interventions showed to be effective to allow users to actively intervene in the story with minimal effort.

We had no difficulty to generate different stories simultaneously. The process of sharing stories worked

equally well in the same circumstances. Tests were done with clients running both on the same and on different machines. As far as scalability is concerned, more tests have still to be performed, but preliminary results indicate that the model can work well with a large number of clients.

5. Concluding Remarks

The model of interactive storytelling, discussed in the present paper, aims at the generation and dramatization, by means of 3D animations, of interactive stories in iTV environments. The implementation and use of the Logtell 2 prototype, based on the model, served to confirm that the architecture and methods proposed are viable and able to cope with the requirements of coherence and diversity of story-plots, continuous presentation flow, comfort and ease in interaction, multiple user participation in stories, and scalability. The prototype currently runs on a local network. We are now preparing a version for the Web, with the ultimate purpose of reaching the open Brazilian Digital TV environment. Tests on an increasingly larger scale will be performed at each stage of the implementation.

In more detail, we are working on a new version of the Drama Manager with better 3D animations, additional dramatization possibilities, and with the ability to control the duration of each event. Another short-term objective of the project is the full implementation of features of the model for which a simplified solution was initially adopted. For instance, the coordination of a pool of servers will supersede the limited use of a single server to handle all stories. Also bulkier snapshots should no longer be kept in main memory, being transferred to database storage. More advanced methods for user interaction and evaluation of suggestions for strong interventions are still to be implemented. As extensions to the model itself, we are considering the use of nondeterministic planners for plot generation, and of frame-based schemes and methods to deal with the drives, attitudes and emotions of the acting characters.

Acknowledgements

The authors are grateful to Fundação CAPES for partially supporting this work.

References

CAVAZZA, M., CHARLES, F. AND MEAD, S. 2002. Character-based interactive storytelling. IEEE Intel-ligent Systems, special issue on AI in Interactive Entertainment, 17(4):17-24

CIARLINI, A.E.M., POZZER, C. T., FURTADO AND A.L., FEIJO, B., 2005. A Logic-Based Tool for Interactive Generation and Dramatization of Stories. In: Proc. ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, Valencia.

CIARLINI, A. AND FURTADO, A. 2002. Understanding and Simulating Narratives in the Context of Information Systems. In Proc. ER'2002 – 21st. International Conference on Conceptual Modelling, Tampere, Finland, Oct. 2002

CPQD. 2008. Modelo de Referência – Sistema Brasileiro de Televisão Digital Terrestre. Vers. PD.30.12.36A.0002A/RT-08-AB. 2006. <http://www.tvdi.inf.br/index.php?s=artigos>.

DORIA, T. R., CIARLINI, A. E. M. AND ANDREATTA, A., 2008. A Nondeterministic Model for Controlling the Dramatization of Interactive Stories. In: Proceedings of the ACM MM2008 - 2nd ACM Workshop on Story Representation, Mechanism and Context - SRMC'08.

DRISCOLL, G. 2000. The essential guide to digital set-top boxes and interactive TV. Prentice Hall, Upper Saddle River, NJ, 1st edition, Nov. 2000.

FURHT, B. 1996. Interactive television systems. In: Proc.1996 ACM Symposium on Applied Computing, p. 7-11, Philadelphia, Pennsylvania, February 1996. ACM Press.

GRASBON, D. AND BRAUN, N. 2001. A morphological approach to interactive storytelling. In: Proc. CAST01, Living in Mixed Realities. Special issue of Netzspannung.org/journal, the Magazine for Media Production and Intermedia Research, pp. 337-340, Sankt Augustin, Germany

GEVERINI, A. AND SERINA, I. 2002. LPG: A planner based on local search for planning graphs. In Proceedings of the International Conference on AI Planning Systems (AIPS), pp. 968-973.

HOFFMAN, J. 2001. FF: The Fast-Forward planning system. AI Magazine, v. 22, n. 3, pp. 57-62.

KAUTZ, H. A. 1991. A Formal Theory of Plan Recognition and its Implementation. In: Allen, J. F. et al (eds.): Reasoning about Plans. Morgan Kaufmann, San Mateo

MARRS, T. AND DAVIS, S. 2005. JBoss at Work: A Practical Guide. O'Reilly Media, Inc.

MATEAS, M. AND STERN, A. 2003. Façade: An Experiment in Building a Fully-Realized Interactive Drama. In: Proc. Game Developers Conference, p. 4-8, 2003.

NAU, D. S., AU, T.-C., ILGHAMI, O., KUTER, U., MURDOCK, W., WU, D. AND YAMAN, F. 2003. SHOP2: An HTN planning system. Journal of Artificial Intelligence Research, 20:379-404.

PAIVA, A., MACHADO, I. AND PRADA, R. 2001. Heroes, villains, magicians, ... Dramatis personae in a virtual story creation environment. In: Proc. Intelligent User Interfaces

POLTI, G. 1945. Thirty-Six Dramatic Situations. Whitefish, MT: Kessinger Publishing.

PROPP, V. 1968 Morphology of the Folktale, Laurence Scott (trans.), Austin: University of Texas Press.

RIEDL, M. AND YOUNG, R.M. 2004. An intent-driven planner for multi-agent story generation. In 3rd International Conference on Autonomous Agents and Multi Agent Systems, New York, 186-193

SANCRINI, M., 2005. O Uso da Televisão Digital no Contexto Educativo, Educação Temática Digital, Campinas, v. 7, n. 1, ISSN 1676-2592, pp. 31-44.

SPIERLING, U., BRAUN, N., IURGEL, I. AND GRASBON, D. 2002. Setting the scene: playing digital director in interactive storytelling and creation. Computers & Graphics, v. 26, n.1, pp. 31-44, 2002.

URSU, M.F., THOMAS, M., KEGEL, I., et al. 2008. Interactive TV Narratives: Opportunities, Progress, and Challenges. In: ACM Transactions on Multimedia Computing, Communications, and Applications, v. 4, n. 4. ISSN:1551-6857

YOUNG, R. 2001. An Overview of the Mimesis Architecture: Integrating Intelligent Narrative Control into an Existing Gaming Environment. The Working Notes of the AAAI Spring Symp. on Artificial Intelligence and Interactive Entertainment

A Neighborhood Grid Data Structure for Massive 3D Crowd Simulation on GPU

Mark Joselli
UFF, Medialab

Erick Baptista Passos
UFF, Medialab

Marcelo Zamith
UFF, Medialab

Esteban Walter Gonzalez Clua
UFF, Medialab

Anselmo Montenegro
UFF, Medialab

Bruno Feijó
PUC-RIO, ICAD Games

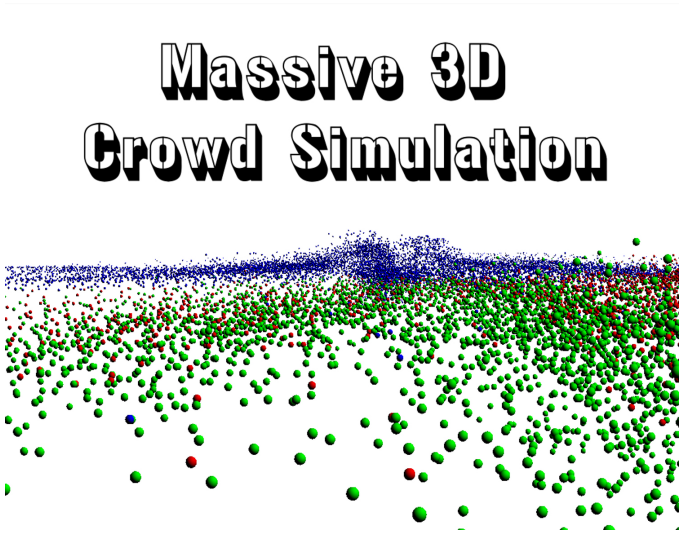


Figure 1: *A screenshot of the simulation.*

Abstract

Simulation and visualization of emergent crowd in real-time is a computationally intensive task. This intensity mostly comes from the $O(n^2)$ complexity of the traversal algorithm, necessary for the proximity queries of all pair of entities in order to compute the relevant mutual interactions. Previous works reduced this complexity by considerably factors, using adequate data structures for spatial subdivision and parallel computing on modern graphic hardware, achieving interactive frame rates in real-time simulations. However, the performance of existent proposals are heavily affected by the maximum density of the spatial subdivision cells, which is usually high, yet leading to algorithms that are not optimal. In this paper we extend previous neighborhood data structure, which is called neighborhood grid, and a simulation architecture that provides for extremely low parallel complexity. Also, we implement a representative flocking boids case-study from which we run benchmarks with simulation and rendering of up to 1 million boids at interactive frame-rates. We remark that this work can achive a minimum speeup of 2.94 when compared to traditional spatial subdivision methods with a similar visual experience and with lesser use of memory.

Keywords:: GPGPU, CUDA, Crowd Simulation, Cellular Automata, Flocking Boids

Author's Contact:

{mjoselli,epassos,mzamith,esteban,anselmo}@ic.uff.br
bruno@inf.puc-rio.br

1 Introduction

In a typical natural environment it is common to find a huge number of animals, plants and small dynamic particles. This is also the case in other densely populated systems, such as sport arenas, communities of ants, bees and other insects, or even streams of blood cells in our circulatory system. Computer simulations of these systems usually present a very limited number of independent entities, mostly with very predictable behavior. There are several approaches that aim to include more realistic behavioral models for crowd simulation such as [Reynolds 1987; Musse and Thalmann 1997; Shao and Terzopoulos 2005; Pelechano et al. 2007; Treuille et al. 2006].

Algorithms for massive crowd simulation are driven by the need to avoid the $O(n^2)$ complexity of the proximity queries between entities. Several approaches have been proposed to cope with this issue [Reynolds 2000; Chiara et al. 2004; Courty and Musse 2005] but none of them has reached an ideal level of scalability. As far as we know, no work until the present date has proposed a real time simulation of more than just a few thousands of complex entities interacting with each other. Applications for these computationally demanding algorithms range over crowd behavior prediction in emergency scenarios, street traffic simulation and enrichment of computer game worlds.

Non-graphics algorithms traditionally executed on the CPU, such as behavioral artificial intelligence algorithms, are sometimes suitable for parallel execution, which makes them appropriate to be implemented on the GPU [Joselli et al. 2008]. However, the first applications of GPUs performing general purpose computation (GPGPU) had to rely on the adaptation of graphics rendering APIs to different concepts, leading to a difficult learning curve and sometimes not very efficient data structures for the proposed solutions. The CUDA [NVidia 2009], CAL [AMD 2007] and OpenCL [Group 2009] technologies aim to provide a new abstraction layer on top of graphics hardware to facilitate its usage for non-graphics processing. Crowd simulation that explores this programming model on the GPU is a promising line of research.

Most of the research on crowd simulation tries to avoid the high complexity of proximity queries by applying some form of spatial subdivision to the environment and classifying entities among the cells based on their position. To accelerate data fetching in a parallel hardware (such as GPUs) the entities list must be sorted in such a way that all entities on the same cells are grouped together. This approach helps lowering the number of proximity queries but is very sensible to the maximum number of entities that can fit in a single cell. In this paper instead of using a similar approach,

we propose a novel simulation architecture that maintains entities into another kind of proximity based data structure, which we call "neighborhood grid". In this data structure, each cell now fits only one entity and does not directly represent a discrete spatial subdivision. The "neighborhood grid" is an approximate representation of the system of neighborhoods of the environment that maps the N-dimensional environment to a discrete map (lattice) with N dimensions, so that entities that are close in a neighborhood sense, appear close to each other in the map. Another approach is to think of it as a multi-dimensional compression of the environment that still keeps the original position information of all entities.

The entities are simulated and sorted as Cellular Automata with Extended Moore Neighborhood [Sarkar 2000] over the neighborhood grid, which is an ideal case for the memory model of GPUs. We argue and show that this approximate simulation technique brings a new bound to crowd simulation performance, maintainning the believability for entertainment contexts. The high performance and scalability are achieved by a very low parallel complexity of the model.

To keep the "neighborhood grid" aligned this work shows a previous implementation of a partial sorting mechanism, a partial odd-even sort, and a new sorting scheme, a bitonic sort, that can keep a much better visual experience with similar performance.

To illustrate and evaluate the "neighborhood grid", we implement a traditional emergent behavior model of flocking boids [Reynolds 1987] that has a minimum speedup of 2.94 over the tradition spatial hashing methods [Reynolds 2000; Reynolds 1999], with similar visual experience. The architecture can be further extended to any other simulation model that rely on dynamic autonomous entities and neighborhood information.

Summaryzing, this work is an extension of the work [Passos et al. 2008], with the following enhancements, which are the main contributions of these paper:

- Extension of the data structure for 3D environments;
- Presentation of a new sorting scheme that keeps a better visual experience with similar performance;
- Comparison of performance between our method and the tradition spatial hashing method [Reynolds 2000; Reynolds 1999] which was also implemented by this work.

The paper is organized as follows: Section 2 discusses related work on crowd simulation. Sections 3 explain the proposed "neighborhood grid", the data structures, the simulation steps in 3D and a simplification of the "neighborhood grid" for 2D systems. Section 4 describes the particular behavior model used to validate the proposed architecture. Section 5 brings the experimental results and analysis of the implemented simulation model. Finally, section 6 concludes the paper with a discussion on future work.

2 Related Work

The first known agent-based simulation for groups of interacting animals is the work proposed by Craig Reynolds [Reynolds 1987], in which he presented a distributed behavioral model to perform this task. His model is similar to a particle system where each individual is independently simulated and acts accordantly to its observation of the environment, including physical rules such as gravity, and influences by the other individuals perceived in the surroundings. The main drawback of the proposed approach is the $O(n^2)$ complexity of the traversal algorithm needed to perform the proximity tests for each pair of individuals. This was such an issue at the time that the simulation had to be run as an offline batch process, even for a limited number of individuals. In order to cope with this limitation, the author suggested the use of spatial hashing. This work also introduced the term *boid* (abbreviation for birdoid) that has been used to designate generic simulated flocking creatures ever since.

Musse and Thalmman [Musse and Thalmann 1997] propose a more complex modeling of human motion based on internal goal-oriented parameters and the group interactions that emerge from the simulation, taking into account sociological aspects of human relations. Others include psychological effects [Pelechano et al. 2007], social forces [Cordeiro et al. 2005] or even knowledge and learning aspects [Funge et al. 1999]. Shao and Terzopoulos [Shao and Terzopoulos 2005] extend the latter including path planning and visibility for pedestrians. It is important to mention that these proposals are mainly focused on the correctness aspects of behavior modeling. The data structures and algorithms used by these works are not suitable for real-time simulation of very large crowds, which is one of the goals of this work.

Reynolds further enhanced his behavioral model to include more complex rules and to achieve the desired interactive performance by the use of spatial hashing [Reynolds 2000; Reynolds 1999]. This implementation could simulate up to 280 boids at 60 fps in a Playstation 2 hardware. By using the spatial hash to classify the boids into a grid, the proximity query algorithm could be performed against a reduced number of pairs. For each boid, only those inside the same grid cell and at adjacent ones, depending on its position, were considered. This strategy leads to a sequential complexity that is closer to $O(n)$. This complexity, however, is highly dependent on the maximum density of each grid cell, which can be very high if the simulated environment is large and dense. We remark that the complexity of our neighborhood grid is not affected by the size of the environment or the distribution of the boids over it.

Quinn et al. [Quinn et al. 2003] used distributed multiprocessors to simulate evacuation scenarios up to 10,000 individuals at 45 fps on a cluster connected by a gigabit switch. More recently, a similar spatial hashing data-structure was used by Reynolds [Reynolds 2006] to render up to 15,000 boids in Playstation 3 hardware at interactive framerates, but with a reduced simulation frame rate of around 10 fps. Due to the distributed memory of both architectures, it is necessary to copy compact versions of the buckets/cells of boids to the individual parallel processors before the simulation step could run, copying them back at the end of it to perform the rendering, which leads to a potential performance bottleneck for larger sets of boids. This issue is evidenced in [Steed and Abou-Haidar 2003], where the authors span the crowd simulation over several network servers and conclude that moving individuals between servers is an expensive operation.

The use of the parallel power of GPUs in massive crowd simulation is very promising but brings another issue, related to its intrinsic dependency on data-locality to achieve high performance in this kind of hardware. For agent-based simulations that rely on spatial hashing, it is desired that the individuals should be sorted through the data-structure based on their cell indexes. The work by Chiara et. al. [Chiara et al. 2004] makes use of the CPU to perform this sorting. To avoid the performance penalty, this sorting task is triggered only when a boid departs from its group, which is detected by the use of a scattering matrix. This system could simulate 1,600 boids at 60 fps including the rendering of animated 2D models. Also the work by Silva et al. [Silva et al. 2008] implement a similar work, but it focus on the optimization of the algorithm by doing occlusion based on the vision of the boids. The FastCrowd system [Courty and Musse 2005] was also implemented with a mix of CPU and GPU computation to simulate and render a crowd of 10,000 individuals at 20 fps as simple 2D discs. Using this simple rendering primitive, the GPU was also capable of simultaneously computing the flow of gases on an evacuation scenario. A more recent work in the GPGPU field by Shopf et. al. [Shopf et al. 2008] presents an implementation that runs entirely on the GPU and can simulate and render 3,000 high detailed animated models or 65,000 simple primitives at real-time frame rates. Our implementation also runs entirely on the GPU and makes use of the fact that groups tend to move as blocks and uses a parallel sorting algorithm on the GPU to achieve even higher performance, as explained in the next sections.

The simulation architecture and data-structures proposed by [Treuille et al. 2006] depart from the agent-based models presented so far. These authors uses a 2D dynamic field to represent both the crowd density and the obstacles of the environment. The individuals navigate through and according to this continuum field. Treuille et al argue that locally controlled agents, while providing for complex emergent behavior, are not an appropriate model for goal-driven individuals, such as human pedestrians. The im-

plemented system could simulate up to 10,000 humans at 5 fps (without graphics) even with the inclusion of a dynamic environment such as traffic lights. The continuum field is an interesting approach but limits the environment to a predetermined size.

In the work [Passos et al. 2008], which this work extends, is implemented a crowd simulation system on the GPU where each boid is modeled as a cellular automaton [Sarkar 2000] in a 2D data structure. This work could achieve the simulation and renderization of up to 1 millions boids at interactive frame rates. The present paper extends that previous work to 3D environments keeping the same performance. As far as we are aware, no other work in the literature presents such a high performance.

This cellular automata model matches perfectly with the ideal locality for data fetching on graphics hardware but imposes that boids information have to be kept reasonably sorted over this data structure during simulation. Our proposal, such as most of the above work, is based on distributed agents to yield emergent behaviour, but the novel data-structures are suitable for unlimited environment size and better scalability over both the number of entities and neighbourhood reach.

3 Simulation Architecture

Individual entities in crowd behavior simulations depend on observations of their surrounding neighbors to decide which actions to take. The straightforward implementation of the neighborhood finding algorithm has a complexity of $O(n^2)$, for n entities, since it performs at least one proximity query for each entity pair in the crowd. Individuals are autonomous and can move during each frame, which leads to a very computationally intesive task.

Techniques of spatial subdivision have been used to group and sort these entities in order to accelerate the neighborhood finding task. Current implementations are usually based on variations of relatively coarse subdivisions techniques, such as a grid over the considered environment. After each update, all entities have their grid cell index calculated based on their latest locations. For GPU based solutions, some kind of sorting based on this index has to be performed in order to benefit from the read-ahead and caching mechanisms of such hardware. This way, neighbor entities in geometric space are stored near each other over the data structure. However, static subdivisions have some limitations when simulating large geometric spaces, where the size of each grid cell may fit a large number of entities. This issue limits the neighborhood finding problem by a hidden $O(n^2)$ complexity factor in the worst case scenario.

In this work we propose another approach for the neighborhood finding problem. This approach uses a grid data structure, which we call "neighborhood grid" that is used to store information about all the entities. In this "neighborhood grid", each entity is mapped in a individual cell (1:1 mapping) accordingly to its spatial location, so that entities that are close in a neighborhood sense, appear close to each other in the grid. In order to keep the "neighborhood grid" mapped accordingly to the spatial location, a sorting mechanism is needed. To fulfill that need, we present two sorting mechanism, one partial odd-even sort and one bitonic sort.

This simulation architecture can be described as a continuous loop with the following steps:

- Sorting pass (re-organizes the neighborhood grid);
- Simulation pass (updates position and orientation);
- Rendering pass (draws visible entities).

The following subsections describe the architecture. In the next subsection the "neighborhood grid" is explained. The role of sorting and the types of simulation algorithms suitable to the proposed architecture are also explained in following subsections. Also in the last part of this section we show a simplified version of the data structures for 2D simulations.

3.1 3D Proximity Data Structure: The Neighborhood Grid

The proposed architecture was developed with CUDA technology [NVidia 2009], and, in order to keep the processing entirely at the GPU, all information about entities is mapped as textures for the display-list and vertex shader rendering. The minimum information required for each entity are: position (a vector, representing the position of the entity), speed (a vector for storing the orientation and velocity in a single structure) and type (an integer that can be used to differentiate entity classes).

This information is stored in 3D arrays (grid), where each position holds the entire data for an individual entity. In this case two grids are required, one for the 3D position and another for the orientation with the entity type variable kept at a fourth value in one of these grids. The grid that contains the position vector for the entities is then used as a sorting structure. In this data structure, each cell fits only one entity. Figure 2 illustrates how a randomly distributed set of entities would be arranged in the "neighborhood grid" when correctly sorted. The smaller circles represent entities that are further away from the viewpoint.

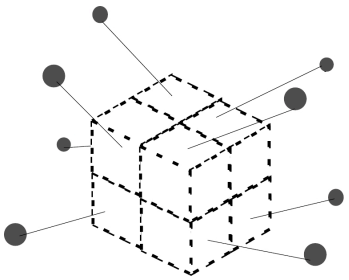


Figure 2: *An example of a distribution of entities in the neighborhood grid. Entities that are further away from the viewpoint are illustrated by small circles.*

In this work we use a form of neighborhood gathering that is known as Extended Moore Neighborhood [Sarkar 2000] in the Cellular Automata theory. Figure 3 illustrates this structure with a 2D matrix holding arbitrary information for 36 individual entities. To reduce the cost of proximity queries, each entity will only gather information about the entities surrounding its cell, based on a constant radius. In the example of Figure 3, this radius is 2, so the entity represented at cell (2,2) (in gray) would have access to the 24 highlighted surrounding cells/entities (in green) only.

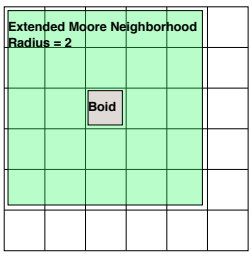


Figure 3: *Example of the Structure of the Extended Moore Neighborhood with 36 entities and radius = 2.*

Our work extends this matrix example to a 3D grid maintaining the same form of information gathering, only adding the extra dimension. Figure 4 illustrates our neighborhood grid with a neighborhood radius of 1.

This kind of spatial data structure and extremely regular information gathering enables a good prediction of the performance, since the number of proximity queries will always be constant over the simulation. This happens because instead of making these proximity queries over all entities inside a coarse grid bucket/cell (variable quantity), such as in traditional implementations, each entity would query only a fixed number of surrounding individual neighbors. However, this matrix has to be sorted continually in such a way that

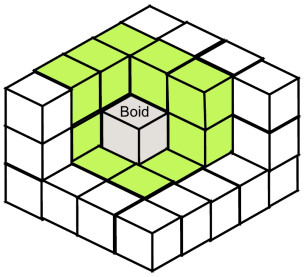


Figure 4: *Example of the neighborhood grid with radius = 1.*

those entities which are neighbors in geometric space are stored in individual cells that are close to each other. This guarantees that each entity should gather information about its closest neighbors. Depending on the simulation (and the sorting step), some misalignment may occur over the data structure, causing that some of the neighbor entities are missed by the gathering step. However, the larger the Moore radius is, less likely it is to happen such issue, which we could observe during the experiments.

3.2 Sorting Pass

The position information of each entity is used to perform a lexicographical sort based on the three dimensions of this vector. The goal is to store in the closer-bottom-leftmost cell of the grid the entity with the smaller values for Z, Y and X, and in the far-top-rightmost cell the entity with highest values of Z, Y and X respectively. Using these three values to sort the matrix, the farthest lines will be filled with the entities with the higher values of Z while the top lines will be filled with the entities with higher values of Y and the right columns will store those with higher values for X and so on. This kind of sorting provides for the approximate neighborhood query that is optimal in terms of data locality.

When performing a sorting over an one dimension array of float point values, the goal is that given an array **A**, the following rule must apply at the end:

- $\forall A[i] \in \mathbf{A},\ i > 0 \Rightarrow A[i-1] \leq A[i]$.

Extending this rule to a grid **G** where each cell has three float point values *X*, *Y* and *Z*:

1. $\forall G[i][j][k] \in \mathbf{G},\ k > 0, G[i][j][k].Z \leq G[i][j][k-1].Z$;
2. $\forall G[i][j][k] \in \mathbf{G},\ k > 0, G[i][j][k-1].Z = G[i][j][k].Z \Rightarrow G[i][j][k].X \leq G[i][j][k-1].X$;
3. $\forall G[i][j][k] \in \mathbf{G},\ k > 0, G[i][j][k-1].Z = G[i][j][k].Z\ AND\ G[i][j][k].X \leq G[i][j][k-1].X \Rightarrow G[i][j][k].Y \leq G[i][j][k-1].Y$;
4. $\forall G[i][j][k] \in \mathbf{G},\ j > 0, G[i][j][k].Y \leq G[i][j-1][k].Y$;;
5. $\forall G[i][j][k] \in \mathbf{G},\ j > 0, G[i][j-1][k].Y = G[i][j][k].Y \Rightarrow G[i][j][k].Z \leq G[i][j-1][k].Z$;;
6. $\forall G[i][j][k] \in \mathbf{G},\ j > 0, G[i][j-1][k].Y = G[i][j][k].Y\ AND\ G[i][j-1][k].Z \leq G[i][j][k].Z \Rightarrow G[i][j][k].X \leq G[i][j][k-1].X$;
7. $\forall G[i][j][k] \in \mathbf{G},\ i > 0, G[i][j][k].X \leq G[i-1][j][k].X$;
8. $\forall G[i][j][k] \in \mathbf{G},\ i > 0, G[i-1][j][k].X = G[i][j][k].X \Rightarrow G[i][j][k].Y \leq G[i-1][j][k].Y$;
9. $\forall G[i][j][k] \in \mathbf{G},\ i > 0, G[i-1][j][k].X = G[i][j][k].X\ AND\ G[i][j][k].Y \leq G[i-1][j][k].Y \Rightarrow G[i][j][k].Z \leq G[i-1][j][k].Z$;

The architecture is independent of the sorting algorithm used, as long as the rules above are always, eventually or even partially achieved during simulation, depending on the desired neighborhood precision. In this work we show a partial odd-even sort, which makes a partial sort in each dimension and a bitonic sort [Batcher 1968], which make a full sort in each dimension.

3.2.1 Partial Odd-Even Sorting

Here we present an inherently parallel (but not optimal) partial sort strategy: an odd-even transposition sort, with only one odd-even pass per update. The odd-even transposition sort is similar to the bubble sort algorithm and it is possible to complete a partial pass, traversing the whole data structure, in $O(n)$ sequential time or $O(1)$ parallel complexity when running on n CUDA threads (if available on the GPU). Because there are two steps, one for odd and other for even elements (for each axis), this algorithm is suitable for parallel execution.

This sorting pass must be spread into six steps, one for odd and one for even elements for each axis. The first step runs the sorting between each entity position vector of the even columns against its immediate neighbor in the subsequent odd column. If the rules described by Eq.1, Eq. 2 or Eq.3 are violated, the entities switch cells in the grids. The other six sorting steps perform the same operation for the odd column of the Z and the similar steps over the Y and X axis.

From tests we have seen that with this partial sort more than 10% of entities are in the wrong place on the "neighborhood grid" when comparing with a full sort on the entire grid. So this sorting mechanism only seems viable on simulation that does not need a lot of precision, or that the entities does not change position very often. Otherwise the use of the bitonic sort is advised, which is present next.

3.2.2 Bitonic Sorting

The bitonic sort [Batcher 1968] is simple parallel sorting algorithm that is very efficient when sorting small number of elements [Blelloch et al. 1998], which is our case since our sort strategy is divided by dimensions. Our implementation is an optimized and adapted version based on a demo from nVidia [nVidia 2008]. This sort is divided in 3 passes, one for each dimension (X,Y and Z).

The complexity of this algorithm is $O(nlog(n)^2)$ being n the number of elements to sort in sequential time. This comparisons are divided in n CUDA threads making the algorithm in this parallel implementation with a complexity of $O(log(n)^2)$, if there is n stream processors on the GPU.

This sorting does not make a full sort on the "neighborhood grid" only a full sort on each dimension (X,Y and Z) of the grid. So, for example, if a change in one entity position on the Y pass, another pass for the X would be needed in order to keep the "neighborhood grid" with an full sort. But from tests we have seen that this misaligned is very small, less than 1% of the entities changes place in one step of the simulation, and in the next step this error will be fixed, and the use of a full sort on the "neighborhood grid" would impose some lost in performance without visible gain in the simulation.

3.3 Simulation Pass

The simulation pass can perform any kind of emergent crowd behavior for entities that are constrained to the knowledge of data in their surrounds, such as flocking boids, swarms or pedestrian groups. This pass must be implemented as a CUDA kernel function that receives as arguments at least the position and orientation of each entity (double buffered as input and output) and the time elapsed since the last step. This kernel function is then executed in parallel with one CUDA thread for each entity. This function uses the data from the previous step for the respective entity and its neighbors and calculates new values for its entity only, which must be written to the same cell in the output grid.

In Section 4, an example of a flocking boids simulation pass is described. The implementation of such simulation in our architecture is evaluated in Section 5. The following subsection is dedicated to explain the 2D version of the presented data structures.

3.4 Simulation Architecture with 2D Proximity: The Neighborhood Matrix

The 2D proximity is a simplification of the 3D proximity with only the X and Y (or Z) dimension. In this case, the proximity data structure used is a "neighborhood matrix" instead of the 3D "neighborhood grid", but with similar extended Moore neighborhood as showed in figure 3. The sorting pass is a simplified one with just the X and Y passes.

We remark that the term 2D mentioned refers only to the spatial nature of the data structure, which is still suitable for a simple 3D simulation where the entities do not traverse the third dimension too much such as a pedestrian crowd. For more complex simulations where entities move freely over the third axis, such as swarm of bees, we recommend using the 3D version of the proximity data structure.

4 Case-Study: Flocking Boids

For the purpose of validating the proposed technique, we choose to implement a well known distributed simulation algorithm called flocking boids [Reynolds 1987]. This is a good algorithm to use because of its good visual results, proximity to real world behavior observation of animals and understandability. The implementation of the flocking boids model using our "neighborhood grid" enables real time simulation with up to one million animals of several types, with a corresponding visual feedback as shown in the experiments described the next section.

Our model simulates a crowd of animals interacting with each other and avoiding random obstacles around the continuous 3D space. This simulation can be used to represent from small bird flocks to huge and complex terrestrial animal groups. Boids from the same type (representing species) try to form groups and avoid staying close to the other types. The number of simulated entities/boids and types is limited only by technology but, as demonstrated in the next section, our method scales very well due to the data structures used. In this section we focus on the extension of the concepts of cellular automata in the simulation step, in order to represent emergent animal behavior.

To achieve a believable simulation we try to mimic what is observable in nature: many animal behaviors resemble that of cellular automata, where a combination of internal and external factors (from neighbor cells) defines which actions are taken and how they are done. With this approach, internal state is represented by position, speed (also orientation) and the boid type, and external information reffers to visible neighbors, depending on where the boid is looking at (orientation), and their relative distances.

Our simulation algorithm computes these influences for each boid: flocking (grouping, repulsion, and direction following); leader following; and repulsion from other types of boids (that can be used also for obstacle avoidance). Additionally, there are constant multiplier factors which dictate how each influence type may get blended with another. In order to enable a richer simulation, these factors are stored independently for each type of boid in separate arrays. More information about the behavior used in this work refer to [Passos et al. 2008].

5 Performance and Analysis

In this work, we implemented and tested the flocking boids case-study using the "neighborhood grid" and also evaluated the rendering of all boids. The rendering consists of a simple display list that is repeated for each entity/boid using the position and orientation information gathered from a texture that is bound from the output VBO of the CUDA simulation in a vertex shader as can be seen on Figure 5.

All tests in this work were performed on an Intel Core 2 Quad 2.4GHz CPU with 3GB of RAM and equipped with an NVidia 8800 GTS GPU (that has 96 stream processors) and the operating system is Windows Vista. Each instance of the test ran for 300 seconds. The average time to compute a frame (and subsequent frames per second) was recorded for each experiment. To assure the results are consistent, each test was repeated 10 times and the standard deviation of the average times confirmed to be within 3%. All tests results includes the simulation calculation and also the renderization to the screen.

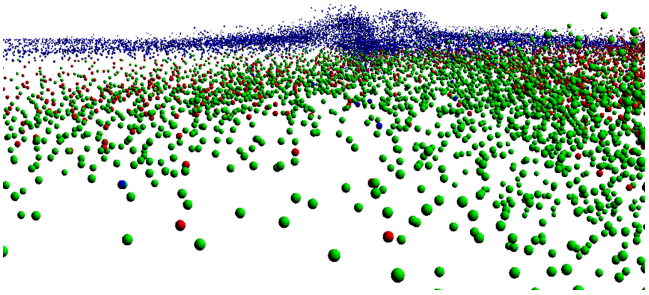


Figure 5: *Simulation with 32K boids.*

To evaluate the scalability of the architecture, we varied the number of entities/boids being simulated (from 1 thousand to 1 million) and the Moore neighborhood radius (from 1 to 4). At preliminary tests, we observed that the number of different boid types had no observable influence on the performance, so a fixed number of 4 types was used for all experiments. In order to fully evaluate the speedup of this architecture for crowd simulation over the traditional spatial hashing method based on [Reynolds 2000; Reynolds 1999], we have implemented the spatial hasing scheme in GPU with the use of CUDA. This implementation has the same flocking behavior as the one implemented in our architecure.

Table 1 shows the results of the simulation in frames per second, for all experiments in 3D with the partial odd-even sort compared with the traditional spatial hashing. We can notice that the simulation runs at interactive frame-rates even with 1 million boids. With the use of the partial odd-even sort, we see the best visual performance with radius 4, but we can still see some strange behavior some times, i.e, collision and behaviors that should not happen. With this radius we have a minimum speedup of 1.86 when compared with the traditional spatial hash method.

Table 2 shows the results of the simulation in frames per second, for all experiments in 3D with the bitonic sort compared with the traditional spatial hashing. With the use of the bitonic sort, we see the best visual performance with radius 2. With this radius we have a minimum speedup of 2.94 when compared with the traditional spatial hash method.

From this results we can see that the bitonic sort is faster than the partial odd-even sort when there are less than 32,768 entities. This happens mainly because the bitonic sort does 1 pass for each dimension while the partial odd-even sort does 2 passes for each dimension. Also using the best radius for visual experience (radius 2 for the bitonic sort and radius 4 for the partial odd-even sort), we can see that the bitonic sort have minimum speedup of 1.16 over the partial odd-even sort. We suggest that for the best visual and performance crowd simulation, to use the presented architecture with bitonic sort and the radius 2.

Table 3 shows how much memory for the presented architecture, which is the same for both sorting mechanisms and different neighborhood radius, and for the spatial hash. From this results we can see that this architecture spends much less memory since it does not needs a lot of memory to keep the data structure having consuming memory in a linear form, while the spatial hash does needs at least 2 MB for keeping the data structure.

Figure 6 shows how the time are spent in % with each step of the simulation (with the data structure, the behavior and the memory copy) during the simulation with 32,769 boids for the bitonic sort (with radius 2), odd-even sort (with radius 4) and spatial hash. This shows that the spatial hash uses 35 % of its time processing the its data structure while the bitonic sort spends 25% and the odd-even

Table 1: *Numerical results of the architecture running with a partial odd-even sort compared with the spatial hash.*

# Boids	Spatial Hash Fps	Partial Odd-Even Sort							
		Radius=1		Radius=2		Radius=3		Radius=4	
		FPS	Speedup	FPS	Speedup	FPS	Speedup	FPS	Speedup
1,024	370	860	2.35	710	1.92	695	1.87	688	1.86
32,768	72	222	3.08	200	2.78	185	2.57	166	2.30
131,072	18	68	3.78	63	3.50	57	3.17	51	2.83
524,288	4.00	19	4.75	17	4.25	15	3.75	12	3.00
1,048,576	0.50	9.72	19.55	8.60	17.20	7.41	14.84	6.25	12.50

Table 2: *Numerical results of the architecture running with a bitonic sort compared with the spatial Hash.*

# Boids	Spatial Hash Fps	Bitonic Sort							
		Radius=1		Radius=2		Radius=3		Radius=4	
		FPS	Speedup	FPS	Speedup	FPS	Speedup	FPS	Speedup
1,024	370	1,155	3.12	1,118	3.02	1,109	3.00	1,099	2.97
32,768	72	212	2.94	197	2.74	178	2.47	164	2.28
131,072	18	62	4.50	58	3.22	53	2.94	48	2.67
524,288	4.00	18	4.50	16	4.00	14	3.50	12	3.00
1,048,576	0.50	8.45	16.90	7.49	14.98	6.64	13.28	6.20	12.40

Table 3: *Use of the memory when using the Spatial Hash and this architecture.*

# Boids	Use of Memory	
	Spatial Hash	Neighborhood Grid
1,024	2.1 MB	5.6 KB
32,768	2.3MB	180 KB
131,072	3 MB	721 KB
524,288	5.5 MB	2.9 MB
1,048,576	9 MB	5.8 MB

only 14%.

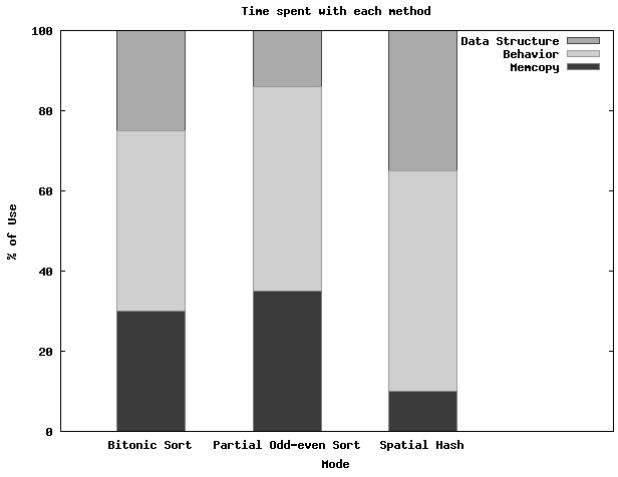


Figure 6: *Comparison of the % of use between the Bitonic Sort, Odd-even partial sort and Spatial Hash.*

Figure 7 shows comparison between the spatial hashing and the two sorting schemes, showing how the time to compute and render each frame grows with the number of boids using the the same radius 2 for the bitonic sort and radius 4 for the partial odd-even sort. These plot uses a logarithmic scale in both axis (due to the growth on the number of boids in the experiments) which shows that there is a linear (and not quadratic) relation between the number of boids and the computing cost, for our architecture.

Also we have tested the architecture with the simplified data structure for 2D simulations (the "neighborhood matrix"). The results show gains from 10 to 20 % in speedup when compared with the 3D simulation. The reason for this performance difference is not related to the simulation itself, but to the fact that in 3D the number of candidate neighbors is higher, leading to more memory reads for each boid. For instance, with a radius of 1 each boid agent in 2D reads its 8 immediate neighbors, while in 3D this number (with the same radius) grows to 26. The architecture is also implemented in the CPU so it can be used in computers that does not have GPU with CUDA. In this case it can only simulate and render up to 8,000 boids in real time.

6 Conclusion

In this paper we have shown an extension of a novel technique for simulating emergent behavior of dynamic entities in a densely populated environment. We have extended all of our data structure to higher dimension (3D) in order to deal with 3D scenes. We also have implemented and compared two sorting techniques to be used with the architecture, one partial odd-even sort and one bitonic sort. We have seen, from visual experience and numerical test, that for simulating flocking boids the partial odd-even sort is not the best approach when precision is needed, since it keeps a high error in the "neighborhood matrix" of more than 10 % on the grid and this error can be perceved visualy in the simulation by the boids' behavior and collisions. With the best radius for visual experience, radius 2 for the bitonic sort and radius 4 for the partial odd-even sort, we have a speedup of 1.16 with the use of the bitonic sort over the partial odd-even sort.

This architecture is capable of interactively simulating and rendering up to 1 million of individual flocking boids in real time, while the traditional spatial hashing methods expends 2 seconds for executing each frame. And with the use of our architecture with bitonic sort and a radius 2, we experience a similar visual simulation as with the spatial hashing method with expressive speedup. The authors of this work suggest using this configuration, the presented architecture with bitonic sort and the radius 2, to achieve best visual and performance crowd simulation.

The data structures developed for 3D and 2D approximate neighborhood queries lead to very low parallel complexity and are suitable for several different simulation algorithms, as long as they can be modeled as cellular automata with Extended Moore Neighborhood.

As future work we plan to extend this architecture to enable the representation of more complex geometric obstacles such as buildings, terrains or mazes. These augmented data structures and more complex algorithms are being designed in order to achieve more realistic simulations, and consequently providing for a even more believable virtual environment. This project is being developed as a crowd simulation framework where programmers can plug in their chosen sorting and simulation strategies.

While our performance evaluations do not render a complex geometry for each entity, we argue that the performance penalty for adding

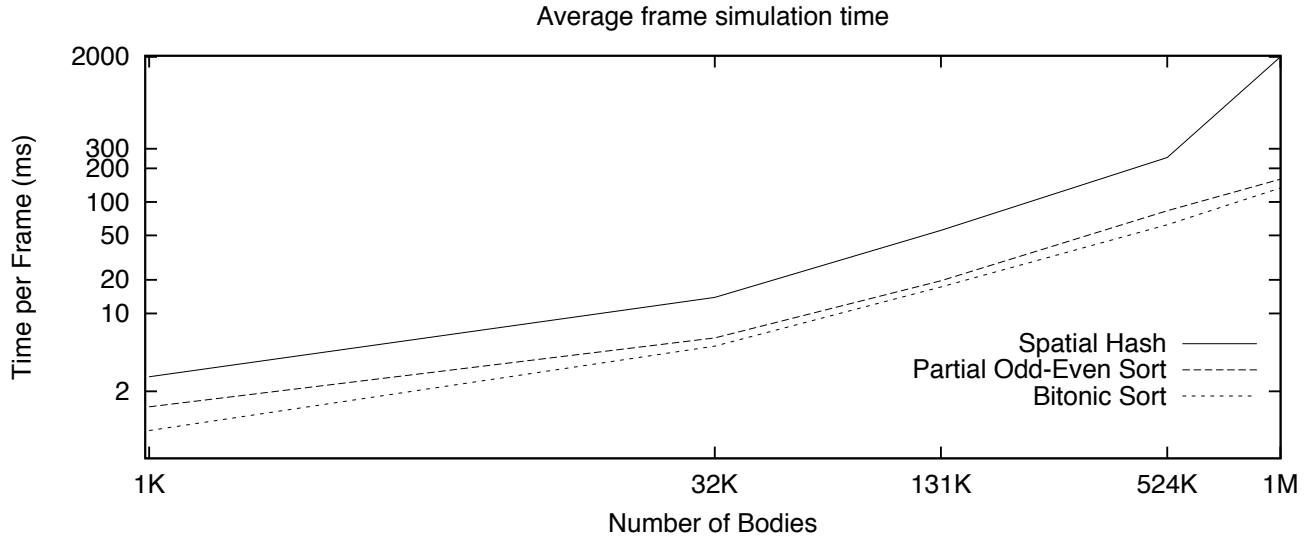


Figure 7: *Comparison of the evolution between the spatial hash and the bitonic and partial odd-even sort.*

such rendering at the end of the simulation can be easily predicted. To do so, we suggest to add the provided computational cost of the sorting and simulation passes (time spent in mili-seconds, with variable numbers of entities and neighborhood radius) to that of a VBO + vertex shader transforming and pixel shader lighting of several copies of the same display list, with more complex geometries, which can be found in published literature.

References

AMD, 2007. Amd stream computing. Available at: `http://ati.amd.com/technology/streamcomputing/firestream-sdk-whitepaper.pdf`. Acessed in 20/02/2009.

BATCHER, K. E. 1968. Sorting networks and their applications. In *AFIPS '68 (Spring): Proceedings of the April 30–May 2, 1968, spring joint computer conference*, ACM, New York, NY, USA, AFIPS, 307–314.

BLELLOCH, G. E., PLAXTON, C. G., LEISERSON, C. E., SMITH, S. J., MAGGS, B. M., AND ZAGHA, M., 1998. An experimental analysis of parallel sorting algorithms.

CHIARA, R. D., ERRA, U., SCARANO, V., AND TATAFIORE, M. 2004. Massive simulation using gpu of a distributed behavioral model of a flock with obstacle avoidance. In *Vision, Modeling, and Visualization (VMV)*, VMV, 233–240.

CORDEIRO, O. C., BRAUN, A., SILVEIRA, C. B., AND MUSSE, S. R. 2005. Concurrency on social forces simulation model. In *Proceedings of the First International Workshop on Crowd Simulation (V-CROWDS)*, V-CROWDS.

COURTY, N., AND MUSSE, S. R. 2005. Simulation of large crowds in emergency situations including gaseous phenomena. In *CGI '05: Proceedings of the Computer Graphics International 2005*, IEEE Computer Society, Washington, DC, USA, CGI, 206–212.

FUNGE, J., TU, X., AND TERZOPOULOS, D. 1999. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *Siggraph 1999, Computer Graphics Proceedings*, Addison Wesley Longman, Los Angeles, A. Rockwood, Ed., Siggraph, 29–38.

GROUP, K., 2009. Opencl - the open standard for parallel programming of heterogeneous systems. Avalible at: `http://www.khronos.org/opencl/`.

JOSELLI, M., ZAMITH, M., VALENTE, L., CLUA, E. W. G., MONTENEGRO, A., CONCI, A., AND FEIJÓ, PAGLIOSA, P. 2008. An adaptative game loop architecture with automatic distribution of tasks between cpu and gpu. *Proceedings of the VII Brazilian Symposium on Computer Games and Digital Entertainment*, 115–120.

MUSSE, S. R., AND THALMANN, D. 1997. A model of human crowd behavior: Group inter-relationship and collision detection analysis. In *Workshop Computer Animation and Simulation of Eurographics*, Eurographics, 39–52.

NVIDIA, 2008. Bitonic sort demo. Avalible at: `http://www.nvidia.com/content/cudazone/cuda_sdk/Data-Parallel_Algorithms.html#bitonic`.

NVIDIA, 2009. Cuda technology. http://www.nvidia.com/cuda. Acessed in 20/02/2009.

PASSOS, E., JOSELLI, M., ZAMITH, M., ROCHA, J., MONTENEGRO, A., CLUA, E., CONCI, A., AND FEIJÓ, B. 2008. Supermassive crowd simulation on gpu based on emergent behavior. In *Proceedings of the VII Brazilian Symposium on Computer Games and Digital Entertainment*, SBC, 81–86.

PELECHANO, N., ALLBECK, J. M., AND BADLER, N. I. 2007. Controlling individual agents in high-density crowd simulation. In *SCA 07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA, 99–108.

QUINN, M. J., METOYER, R. A., AND HUNTER-ZAWORSKI, K. 2003. Parallel implementation of the social forces model. In *Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics*, PED, 63–74.

REYNOLDS, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH, 25–34.

REYNOLDS, C. 1999. Steering behaviors for autonomous characters. In *Game Developers Conference 1999*, GDC.

REYNOLDS, C. 2000. Interaction with groups of autonomous characters. In *Game Developers Conference 2000*, GDC.

REYNOLDS, C. 2006. Big fast crowds on ps3. In *Sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, ACM, New York, NY, USA, Sandbox, 113–121.

SARKAR, P. 2000. A brief history of cellular automata. *ACM Comput. Surv. 32*, 1, 80–107.

SHAO, W., AND TERZOPOULOS, D. 2005. Autonomous pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM, New York, NY, USA, SCA, 19–28.

SHOPF, J., BARCZAK, J., OAT, C., AND TATARCHUK, N. 2008. March of the froblins: simulation and rendering massive crowds of intelligent and detailed creatures on gpu. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, ACM, New York, NY, USA, SIGGRAPH, 52–101.

SILVA, A. R., LAGES, W. S., AND CHAIMOWICZ, L. 2008. Improving boids algorithm in gpu using estimated self occlusion. In *Proceedings of SBGames'08 - VII Brazilian Symposium on Computer Games and Digital Entertainment*, Sociedade Brasileira de Computação, SBC, SBC, 41–46.

STEED, A., AND ABOU-HAIDAR, R. 2003. Partitioning crowded virtual environments. In *VRST '03: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, VRST, 7–14.

TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. 2006. Continuum crowds. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM, New York, NY, USA, SIGGRAPH, 1160–1168.

A Novel Algorithm to Verify the Solution of Geometric Puzzle Games

Manoel Siqueira Júnior Rafael Alves Esteban Clua Erick Passos
Clayton da Silva Anselmo Montenegro Júlio Cesar Oliveira*

UFF, Media Lab, Brazil *UFRJ Co., Brazil

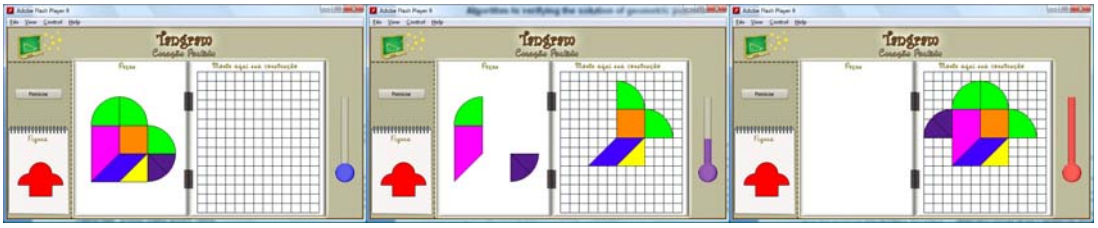


Figure 1: Example of a Tangram game, showing the algorithm verification of the player´s progress with a thermometer.

Abstract

In this paper, we present a novel algorithm to solve the problem of correctly verifying the solution for geometric puzzles. When compared with others, this approach covers a satisfactory amount of cases. The method comprises the use of sixteen possible relations between polygon edges, which are classified to eliminate those that are not necessarily part of the final figure. This method provides for a precise verification of an arranged set of polygons that must form the same image as the desired solution, without the need of extra meta-data. Only the vertexes themselves (also the edge concavity and center position, when circumference arcs are present) are used the algorithm.

Keywords: geometric puzzles, polygon contour, Tangram

Authors' contact:
manoeljr88@gmail.com
rafamachadoalves@yahoo.com.br
{creis,anselmo,epassos,esteban}@ic.uff.br
juliooceano@gmail.com

1. Introduction

Many games are related to puzzles and geometric piece arrangements. For this kind of games, it is necessary that algorithms validate the correctness of the arrangement and also give a feedback to the player, telling how close he is from the correct solution. Figure 1 shows the player's progress in a geometric jigsaw puzzle by the algorithm proposed in this work. All geometric jigsaw puzzles presented in this paper are based on the games proposed in Kaleff et al. [2002].

The algorithm that will be discussed in this paper is an alternative approach to the problem of identification if a determinate geometric jigsaw puzzle assembly represents the intended figure. This work is an extension of the algorithm presented by Scarlatos [1999], which is not suitable for jigsaw puzzle that have more than one possibility for a correct assembly. This work is also an extension of a preliminary work made by Siqueira et al. [2008].

The proposed algorithm use a different approach than that presented by Scarlatos [1999], in order to verify different arrangements of parts that lead to the same figure, considering just the outline formed by the pieces together and not how they are arranged within the contour. The presented approach also analyses that the final assembly has no holes.

When compared with pixel by pixel approaches, the proposed algorithm has the advantage of being easier to get the partial solution and to solve the problem with parts that have any rotations.

It is important to note that the pieces of the geometric jigsaw puzzle and any assembly composed with them can be represented by a polygon, as well as the solution figure (which indicates the solution to be achieved in the game). For this work, the possible outline of the polygons can be defined by edges composed of circular arcs and/or straight segments. If a puzzle is composed of figures that are not polygons, a bounding polygon will be required for algorithm usage.

The remainder of this paper is organized as follows: section 2 presents some works related to the design and implementation of methods for the verification of relations between edges on polygonal figures. Section 3 carries out an analysis of the problems to be handled, while section 4 explains the proposed algorithm. Nevertheless, section 5 talks about some special cases where the algorithm does not correctly detect the contour of the assembly made by the player. Finally, the conclusion and future work associated with the proposal are presented in section 6.

2. Related Work

In this section, we attempt to compare the proposed technique with other implementations of mathematical jigsaw puzzle, specifically in relation to recognition of the solution, and works about generic methods for the

recognition of figures through the classification of relations between edges.

There are few games of geometric jigsaw puzzle that make the recognition of solution, in other words the games indicate to the player if he succeeded or not in completing the assembly. Most implementations found, for example, shown by Jacob [2002], Martins [2002] and Lankin [2001], only allow a passive visual comparison by the player.

The Tangram (geometric jigsaw puzzle) presented by Ztor [2005] makes the recognition of a solution, but this approach does not allow assembly with rotating alternatives. Moreover, the previews work does not make the partial recognition of a solution, an important feature for showing the player how far or close he is from the final solution. The solution presented in this paper allows both arbitrary rotation and the estimation of partial solution, besides allowing the use of pieces with edges composed of circumference arcs.

In Flashkit [1999], it is possible to find a great repository of source code, tutorials and other materials for developing games in Flash, including many examples of Tangrans, as those available in Lankin [2001], Jacob [2002], Martins [2002], Kunovi [2001] and Texas [2000]. However, none of these examples presents algorithms for verification of solution, serving only as a reference for implementation of the basic mechanics.

In Scarlatos [1999], it is shown a method to recognize a figure formed by several polygons juxtaposed through exact relations formed by their edges. This solution, however, supports only edges composed of straight segments and only recognize individually each possible solution. A figure that can be formed by more than one combination of parts may have multiple representations, making it impossible to use for puzzles that may have thousands of possible solutions even for simple instances. Although not satisfactory, the work referenced served as the basis for the method developed.

A situation in which different arrangements of pieces form the same figure is illustrated in Figure 2. In this example, the square figure formed by the union of polygons 2 and 3 may be attached to the polygon 1 in different rotations. While the method referenced by Scarlatos [1999] generates two different representations, the mechanism that is proposed in this paper recognizes both figures as equals, and therefore is more appropriate to the problem of verification of solutions of geometric jigsaw puzzle.

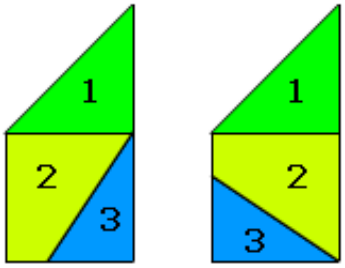


Figure 2: Two different arrangements that represent the same figure.

The authors did not found any other works that, at least using polygons with edges composed of straight segments, provide a general method for the recognition of solution of geometric jigsaw puzzle from the comparison with the figure solution.

3. Problem Considerations

For the arrangement of the pieces during the game to be verified, relations between the edges of different polygons, proposed in the work of Scarlatos (1999), are considered to assist in identifying the contours of the current figure mounted by the player. Besides the basic relations between edges, adjustments are made so that the final result of this assembly is adequately compared with the figure solution.

Both the representation of the solution and the assembly made by the player are composed of circular lists of edges, which describe appropriately the final design of each. It is important to note that the polygons formed during the game may contain holes and it does not interfere in the analysis of the solution proposed by the player.

For a correct usage of the algorithm it is necessary that:

- There is no overlap between the pieces of the geometric jigsaw puzzle, since the elimination of edges can not happen in some cases it was necessary
- Each of the pieces and the solution figure has to be designed either in a clockwise or anti-clockwise direction; in other words, all must be designed in the same direction at the definition of its vertices constituents. Thus, comparison of the assembly by the player with the correct solution is made with the same sense of drawing
- It must be kept in memory the coordinates of the vertices that form the polygons, and if the representation of edges is composed of circumference arcs, it is also necessary the center of the circumference arc and its concavity (concave or convex). Such as it is illustrated on Figure 3, the vertices A, B, C and D; the center and the concavity of circumference arc BC and DA should be stored in memory

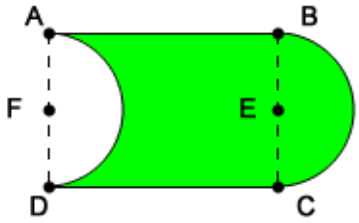


Figure 3: Example of a polygon with edges composed of straight segments and circumference arcs.

.

- The solution polygon can not have holes, since it is not taken into consideration by the algorithm the position in the figure that the hole is located. The presence of holes will only be checked by the player itself. A solution polygon with a hole is shown in Figure 4

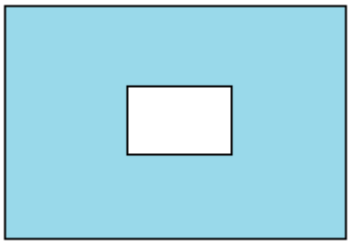


Figure 4: The solution can not contain holes, such as it is represented by the white rectangle.

- The game must be composed only of closed polygons without holes, and formed only by borders consisting of straight segments and circumference arcs, not supporting other types of curves

Despite these restrictions, infinity of forms of pieces is permitted so that the proposed algorithm properly analyzes the solution set by the player during the assembly of geometric jigsaw puzzle.

4. Heuristic for Verification

This heuristic for the verification of the solution of a geometric jigsaw puzzle uses the relations described in Scarlatos [1999]. Thus, it is added the verification of edges composed of circumference arcs, besides interpreting relations separately, removing the edges that are not part of the contour of the result obtained by the relations, so that only the edges of the contour remain. The edges of the contours are then rearranged so that the figure is represented in a unique way.

4.1 Relations

Only the relations between edges of different polygons will be considered. What characterizes each relation between a pair of edges is the verification of the vertices of each edge, checking if it belongs or not to the other analyzed edge.

Initially, it is supposed that only edges composed of straight segments are analyzed and that one of the edges is formed by the points a_1 and a_2 and the other by b_1 and b_2. For each of the four corners, if it belongs to the other edge, it will be assigned value 1, otherwise, value 0. Concatenated values of a_1, a_2, b_1 and b_2, in that order, has the representation of the relationship between two edges expressed in a binary system, converted into a decimal system.

Table 1 (attached at end of paper) shows the possible relations between edges, where the polygons were drawn in a clockwise direction. The arrows of one vertex to another indicate the direction of the edge they represent. The relations are presented in decimal form and are located just below the figure to which they belong.

To deal with edges composed of arcs, in addition to the vertices, it is also necessary the center of the circumference and its concavity (concave or convex) information. Therefore, different data structures are generated to be associated with two types of edges (straight segments and circumference arcs). Thus, relations are established for identifying the type of edge that each one has.

There are three groups of relations: those that split an edge, those that are null and those that totally or partially eliminate the edges. When it includes edges which are circumference arcs in the verification of relations, the interpretation obtained can be more comprehensive. Therefore, to simplify the algorithm, the relations are considered void, in cases where there are not explained in subsections 4.1.1, 4.1.2 and 4.1.3, without even checking the number of the result.

It is important to note that a relation may belong to different groups, since as it comes from the comparison between two edges. This can lead to an individually and differently treatment. For instance, the relation that has the number 12 in Table 1 has one of the edges eliminated completely and the other splitted.

In all cases presented in subsections 4.1.1, 4.1.2 and 4.1.3, the pair of edges can only be analyzed as follows:

- Both edges are straight segments
- Both edges are circumference arcs with equals radius and global center coordinate but with different concavities

Table 2 (attached at end of article) shows the possible relations between edges represented by circumference arcs in the described case immediately above, in which the polygons were drawn clockwise. The arrows from one vertex to another indicate the direction of the edge they represent and the point "c" indicates the center of both edges. The relations are presented in decimal form and are located just below

the figure to which they relate. As can be noted, not all relations are possible.

4.1.1 Split Relation

This type of relations occurs either when only one vertex is tangential to the edge examined or when two vertices that belong to one edge are contained in another edge but not touching in the extreme points of this. These cases are illustrated in Table 1: the relation 1 for the first case and the relation 12 for the second case. Table 2 illustrates the second case using the relation 12.

Based on the points a_1, a_2, b_1 and b_2, below a description of the relations in that group can be:

- Relation 1 ($a_1 = 0$; $a_2 = 0$; $b_1 = 0$; $b_2 = 1$): edge formed by the points a_1 and a_2 is subdivided to point b_2
- Relation 2 ($a_1 = 0$; $a_2 = 0$; $b_1 = 1$; $b_2 = 0$): edge formed by the points a_1 and a_2 is subdivided to point b_1
- Relation 3 ($a_1 = 0$; $a_2 = 0$; $b_1 = 1$; $b_2 = 1$): subdivides the edge formed by the points a_1 and a_2 in two: one whose extreme points are a_1 and b_2 and the other, b_1 and a_2, in this order, since that way the direction of edges is preserved
- Relation 4 ($a_1 = 0$; $a_2 = 1$; $b_1 = 0$; $b_2 = 0$): edge formed by the points b_1 e b_2 is subdivided in point a_2
- Relation 8 ($a_1 = 1$; $a_2 = 0$; $b_1 = 0$; $b_2 = 0$): formed by the points b_1 e b_2 is subdivided in point a_1
- Relation 12 ($a_1 = 1$; $a_2 = 1$; $b_1 = 0$; $b_2 = 0$): subdivides the edge formed by the points b_1 and b_2 in two: one whose extreme points are b_1 e a_2 and the other, a_1 e b_2, in this order, because that way the direction of edges is preserved

4.1.2 Null Relation

This type of relation occurs in the verified edge's vertex either when the vertex do not touch the other, or the intersection between these two edges result only in a vertex in common. Both cases are illustrated in Table 1. For example, the relation 0 for the first case and the relation 6 for the second case. Table 2 uses the relation 0 to illustrate the first case.

Similarly to the previous subsection, below there is a description of the relations contained in that group:

- Relation 0 ($a_1 = 0$; $a_2 = 0$; $b_1 = 0$; $b_2 = 0$): as they are disjoint edges, there is nothing to do
- Relation 5 ($a_1 = 0$; $a_2 = 1$; $b_1 = 0$; $b_2 = 1$): as only the points a_2 and b_2 touch each other, there is no need to change the two edges, so there is nothing to do
- Relation 6 ($a_1 = 0$; $a_2 = 1$; $b_1 = 1$; $b_2 = 0$): as only the points a_2 and b_1 touch each other, no need to change the two edges, so there is nothing to do
- Relation 9 ($a_1 = 1$; $a_2 = 0$; $b_1 = 0$; $b_2 = 1$): as only the points a_1 and b_2 touch each other, no need to change the two edges, so there is nothing to do
- Relation 10 ($a_1 = 1$; $a_2 = 0$; $b_1 = 1$; $b_2 = 0$): as only the points a_1 and b_1 touch each other, no need to change the two edges, so there is nothing to do
- Relation 15 (a1 = 1; a2 = 1; b1 = 1; b2 = 1): If the vertex a_1 has the same global coordinate as b_1, a_2 has the same global coordinate as b_2 and both edges are circumference arcs with same radius and global coordinate center (Figure 5), there is nothing to do

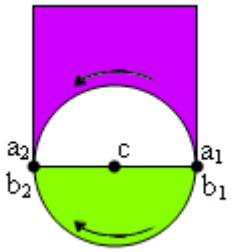


Figure 5: Case where relation 15 is null

4.1.3 Elimination relation

This type of relation occurs when the two vertexes of one edge touches the examined edge or when one of the vertices of each edge touches another, but these points do not have the same global coordinate. Both cases are also illustrated in Table 1 and Table 2. Relation 3 and 5 are examples for the first and second case, respectively.

Similarly to the previous two subsections, following there is description of the relations contained in that group:

- Relation 3 ($a_1 = 0$; $a_2 = 0$; $b_1 = 1$; $b_2 = 1$): eliminates the edge b_1b_2
- Relation 5 ($a_1 = 0$; $a_2 = 1$; $b_1 = 0$; $b_2 = 1$): if a_2 and b_2 have different coordinates, it is eliminated the parts that touch the two edges connected
- Relation 7 ($a_1 = 0$; $a_2 = 1$; $b_1 = 1$; $b_2 = 1$): it is eliminated the edge b_1b_2 and the part of the edge a_1a_2 that has intersection with the edge b_1b_2
- Relation 10 ($a_1 = 1$; $a_2 = 0$; $b_1 = 1$; $b_2 = 0$): if a_1 e b_1 have different coordinate, it is eliminated the parts that touch the two connected edges
- Relation 11 ($a_1 = 1$; $a_2 = 0$; $b_1 = 1$; $b_2 = 1$): it is eliminated the edge b_1b_2 and the part of the

edge a_1a_2 that has intersection with the edge b_1b_2;

- Relation 12 ($a_1 = 1$; $a_2 = 1$; $b_1 = 0$; $b_2 = 0$): it is eliminated the edge a_1a_2
- Relation 13 ($a_1 = 1$; $a_2 = 1$; $b_1 = 0$; $b_2 = 1$): it is eliminated the edge a_1a_2 and the part of the edge b_1b_2 that has intersection with the edge a_1a_2;
- Relation 14 ($a_1 = 1$; $a_2 = 1$; $b_1 = 1$; $b_2 = 0$): it is eliminated the edge a_1a_2 and the part of the edge b_1b_2 that has intersection with the edge a_1a_2;
- Relation 15 ($a_1 = 1$; $a_2 = 1$; $b_1 = 1$; $b_2 = 1$): it is eliminated the two edges if they are straight segments. If both edges are circumference arcs with same radius and global coordinate center, but have different concavities they will only be partial or total removed if one of the following conditions are met:
  - The vertex a_1 has the same global coordinate b_2 and $b_1$has the same global coordinate a_2 (Table 2, relation 15). In this case, there is a complete elimination of the edges;
  - The vertex a_1 has the same global coordinate b_1 and b_2 has the same global coordinate a_2 (Figure 6). In this case, there is an elimination of the edges parts which is between the vertexes b_2 and a_2;

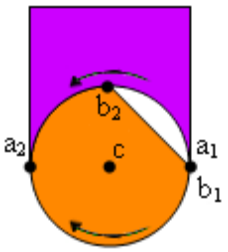


Figure 6: Relation 15 with elimination of the edges which are among the vertexes b_2 and a_2.

  - The vertex a_1 has not the same global coordinate b_1 and b_2 has the same global coordinate a_2 (Figure 7). In this case, there is elimination of the edges parts which is between the vertexes b_1 and a_1;

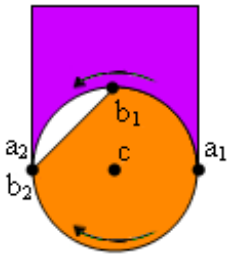


Figure 7: Relation 15 with elimination of the edges which are among the vertexes b_1 and a_1.

  - All vertexes have different global coordinates (Figure 8). In this case, there is elimination of the edges parts which are between the vertexes b_1 and a_1 and between b_2 and a_2.

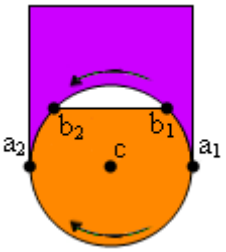


Figure 8: Relation 15 with elimination of the edges which are among the vertexes b_1 and a_1 and between b_2 and a_2.

4.2 Adjusting Relation Outcomes

Firstly, it's important to know that there is a data structure that represents the list of final figures and each figure is represented by one edge circular list. The final figures are the contours obtained with the assembly of player pieces.

After the relations were applied, contour edges are found, but it's needed to adjust them. Thus one or more final figures are identified. This adjusting is done selecting an edge to be added in initial position of the circular list that represents any of the final figures. After, the next edge to be added in this data structure is selected. The chosen edge is that among those with a global coordinate of initial vertex equal to the end vertex of the last edge, added to the circular list and has the smallest angle with the last edge inserted in the circular list. This is done until a closed polygon is found. During this phase are also unified adjacent straight segments that form a 180° angle between them and adjacent circumference arcs having the same global coordinate center. These operations are repeated until all contours are identified.

With this method, the contours are defined and it is possible to have representations of holes. For this, it is necessary that filled polygons have a direction (clockwise, for example) and holes have the opposite (in the case of Figure 9, counterclockwise).

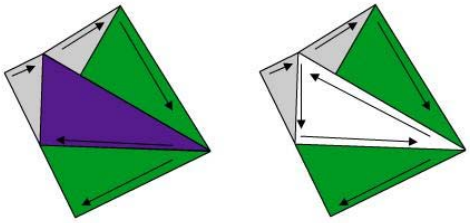


Figure 9: Differences between a filled polygon and a polygon with hole.

To identify the direction of a contour, it is simply necessary to be used the formula of area found in Bourke [1998], which returns positive if the figure is

drawn counterclockwise, and negative if the figure is drawn in another direction.

To verify if a filled polygon represents the solution, there is a circular list in which each element stored is represented by one of the following data structures: one containing the distance between the edge vertexes and the angle between this edge and the next on the list, while the other structure contains this information and the radius and the concavity of the circumference arc. These structures must follow the same order of the circular edge list that represents the solution submitted by the player. The same kind of list is made for solution figure of the game.

Once these lists are ready with the previous structures, the game solution is compared with the player solution. If any filled polygon forms the solution figure, the next step consists on verifying the existence of a hole in this polygon. One strategy suggested for this is presented in solution 2 described by Bourke [1987], which shows how to verify if a point is in a polygon. To identify if a hole belongs to a filled polygon, it is necessary only to apply this solution in all vertexes of the candidate hole.

If the filled polygon is equal to the solution figure and it doesn't have holes, it is possible to conclude that the player set the correct solution.

4.3 Completeness Level

For the calculation of the completeness level, it obtained the minimum between the completeness of the polygon contour assembled by the player that has most proximity to the polygon solution and the filled area of this contour. This measurement of progress is more a motivation of usability than of accuracy, since it allows a progress perception of the player.

5. Special Cases

There is infinity of geometric forms that the verifying heuristic presented in subsection 4 treats correctly. Although there are special cases that the contour isn't identified properly, depending on the manner that the pieces are designed, many of these cases are solved.

A possible special case would be a geometric jigsaw puzzle that has a solution figure illustrated in Figure 10. This solution is drawn in clockwise and composed of edges AB, BC, CE, EB, BF and FA, being AB and BF convexes and centered in point G; BC and EB are convexes and centered in point D.

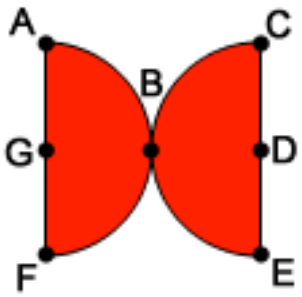


Figure 10: Solution figure of a special case.

It is supposed the available pieces for the solution assembly are two pieces shown in Figure 11. These two pieces were drawn clockwise. One of them is composed of edges AB, BA, being AB convex and centered in point C. The other is composed of edges DF, FD, being FD convex and centered in point E.

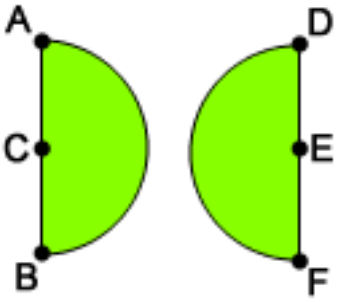


Figure 11: Available pieces for the solution figure assembly.

When the pieces of Figure 11 form the solution figure shown in Figure 10, the edges AB and FD won't be subdivided because the resulting relation between them will be null. Consequently, the contour of the solution figure won't be found.

Meanwhile, this problem can be solved, creating with a differently way the game pieces. Figure 12 shows an alternative of creation pieces that allows the correct assembly of the solution figure. The pieces shown in Figure 12 were drawn clockwise. One of them is composed of edges AB, BC and CA, being AB and BC convexes and centered in point D. The other is composed of edges EG, GH and HE, being GH and HE convexes and centered in point F.

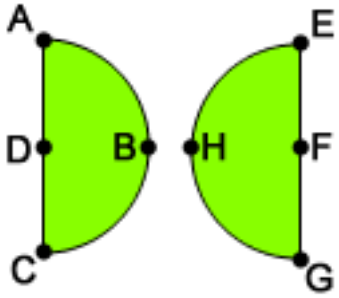


Figure 12: Alternative set of available pieces to assembly the solution figure.

When the pieces of Figure 12 intersect the vertexes B and H, forming the solution figure shown in Figure 10, the figure solution contour will be identified

because the responsible edges for the problem of the null relation of Figure 11 will already be subdivided. In other words, AC was subdivided in AB and BC and GE was subdivided in GH and HE.

Another special case example, similar to previous one, is a geometric puzzle that has the solution figure illustrated in Figure 13. This solution is drawn clockwise and composed of edges AB, BC, CD, DE, EG, GD, DH and HA, being DE and GH convexes and centered in point F.

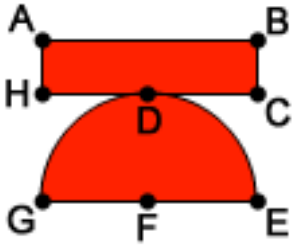


Figure 13: Solution figure of a special case.

Figure 14 shows a possible assembly for the pieces of this geometric puzzle. The pieces shown in this figure were drawn in clockwise. One of them is composed of edges AB, BC, CD and DA. The other is composed of edges EF and FE, being EF convex and centered in point G.

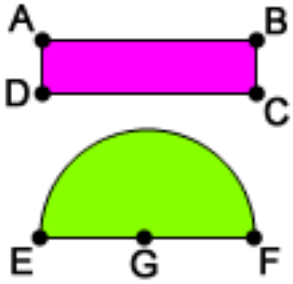


Figure 14: Available pieces for the assembly of the solution figure.

In this situation, the difference with the previous special case is the pair of edges that can originate the contour identification error. This pair is composed of a straight segment and a circumference arc.

Similar to the previous special case, there is a way of creating available pieces on game that is illustrated in Figure 15. Thus the purposed algorithm detects the solution figure contour. The pieces shown in Figure 15 were drawn clockwise. One of them is composed of edges AB, BC, CD, DE, EF and FA. The other is composed of edges GH, HI and IG, being GH and HI convexes and centered in point J.

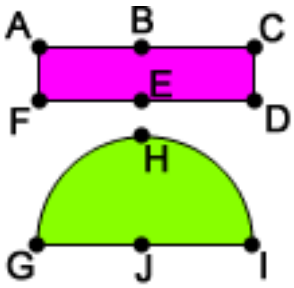


Figure 15: Alternative set of available pieces for the solution figure assembly.

6. Conclusion and Future Works

In this article, it was presented a new approach to the relations between the edges of adjacent polygons. Thus there are other ways of successfully verifying the solution of geometric puzzles other than the ones already established.

This approach comes along to mend faults of other approaches that do not consider only the contour of the solution assembled by the player, but all the arrangement of the pieces. Also the algorithm presented considers edges represented by circumference arcs.

In future, many adaptations must be made in this algorithm, such as:

- Adapting pieces and solutions with edges represented by different kind of curves
- Adding the calculation of intersection between edges to apply division relations of edges in special cases and thus giving more precision to this algorithm
- Including pieces and solutions with holes
- Adapting this algorithm for similar problem in three dimensions

Acknowledgements

This project is founded by the Brazilian ministery of Education and Science and Technology. Special thanks for professor Alexsandre Machado, who helped in the English review, and Ana Kaleff, who presented geometric puzzles to us.

References

LANKIN, A., 2001. *Tangram Implementation.* Available from: http://www.flashkit.com/movies/Games/Full_Game_Source/Tangram-Andrew_L-5966/index.php [Accessed 03 August 2008].

JACOB, E., 2002. *Tangram Implementation.* Available from: http://www.flashkit.com/movies/Games/Full_Game_Source/Tangram-Eduardo_-8137/index.php [Accessed 03 August 2008].

FLASHKIT, 1999. Section: *Movies*; Subsection: *Games*. Available from: http://www.flashkit.com [Acessed 03 August 2008].

MARTINS, I. F., 2002. *Tangram Implementation.* Available from: http://www.flashkit.com/movies/Games/Tangram-Ilclio_-6471/index.php [Acessed 03 August 2008].

KALEFF, A. M.; REI, D.M.; GARCIA S. S. *Quebra-cabeças geométricos e formas planas*. 3 ed. Niteroi: EdUFF, 2002.

DE KUNOVI, N., 2001. *Tangram Implementation.* Available from: http://www.flashkit.com/movies/Games/Full_Game_Source/Tangram-Nicola_d-3822/index.php [Acessed 03 August 2008].

JASP, T., 2000. *Tangram Implementation.* Available from: http://www.flashkit.com/movies/Games/Full_Game_Source/Tangram-Texas_Ja-2061/index.php [Acessed 03 August 2008].

ZTOR, 2005. *Tangram Implementation.* Available from: http://www.ztor.com/index.php4?ln=&g=game&d=tang [Acessed 03 August 2008].

SCARLATOS, L. L., 1999. *Puzzle piece topology: detecting arrangements in smart objects interfaces*.

BOURKE, P., 1987. *Determining if a point lies on the interior of a polygon*. Available from: http://local.wasp.uwa.edu.au/~pbourke/geometry/insidepoly/ [Acessed 21 June 2008].

BOURKE, P., *Determining whether or not a polygon (2D) has its vertices ordered clockwise or counterclockwise*, 1998. Available from: http://local.wasp.uwa.edu.au/~pbourke/geometry/clockwise/ [Acessed 21 June 2008].

SIQUEIRA, M. M. J.; MACHADO, R. A.; SANTOS, W. DOS; CARVALHO, C.; SILVA, C.; MONTENEGRO, A.; PASSOS, E.; CLUA, E. *Algoritmo para Verificação da Solução de Quebra-cabeças Geométricos*. In: SBGames, 2008, Belo Horizonte.

Representation of relations considering only straight segments

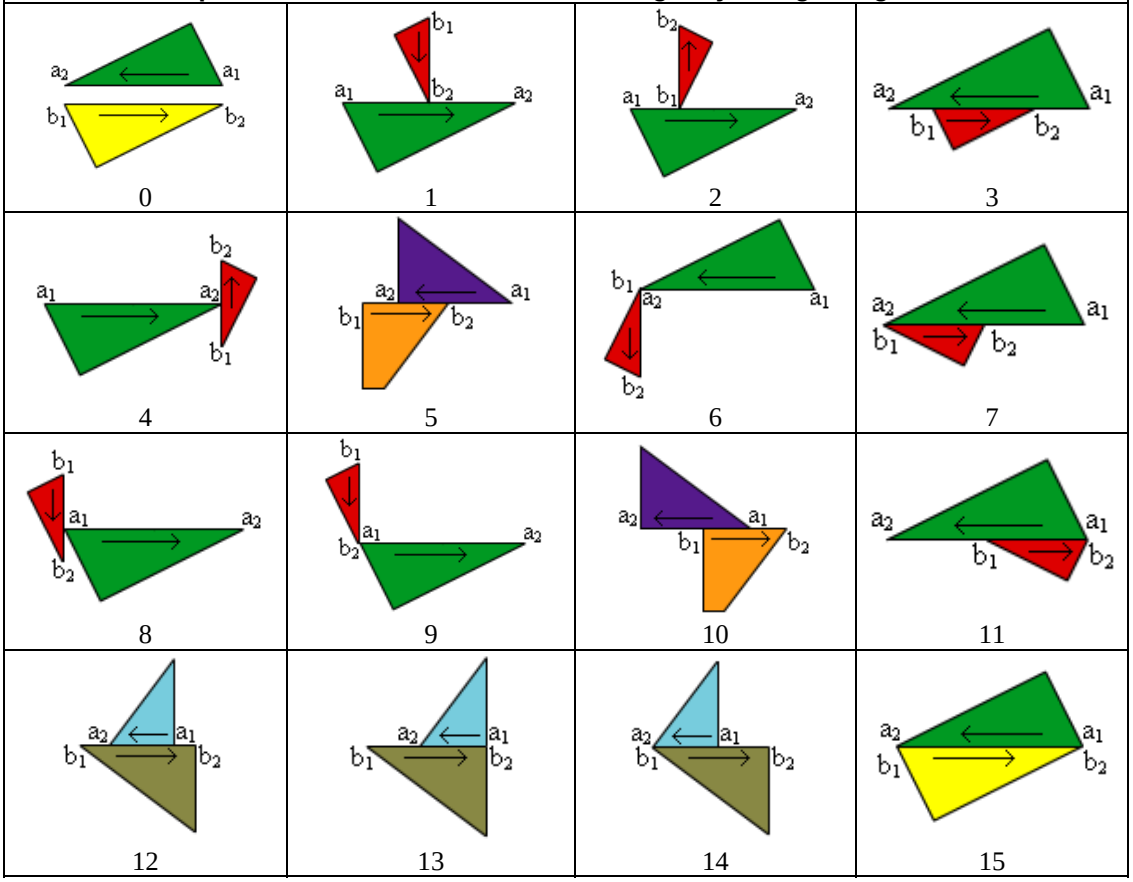


Table 1: Possible representations of straight segment relations, adapted of Scarlatos [1999, p. 4].

Representation of possible relations considering only circumference arcs

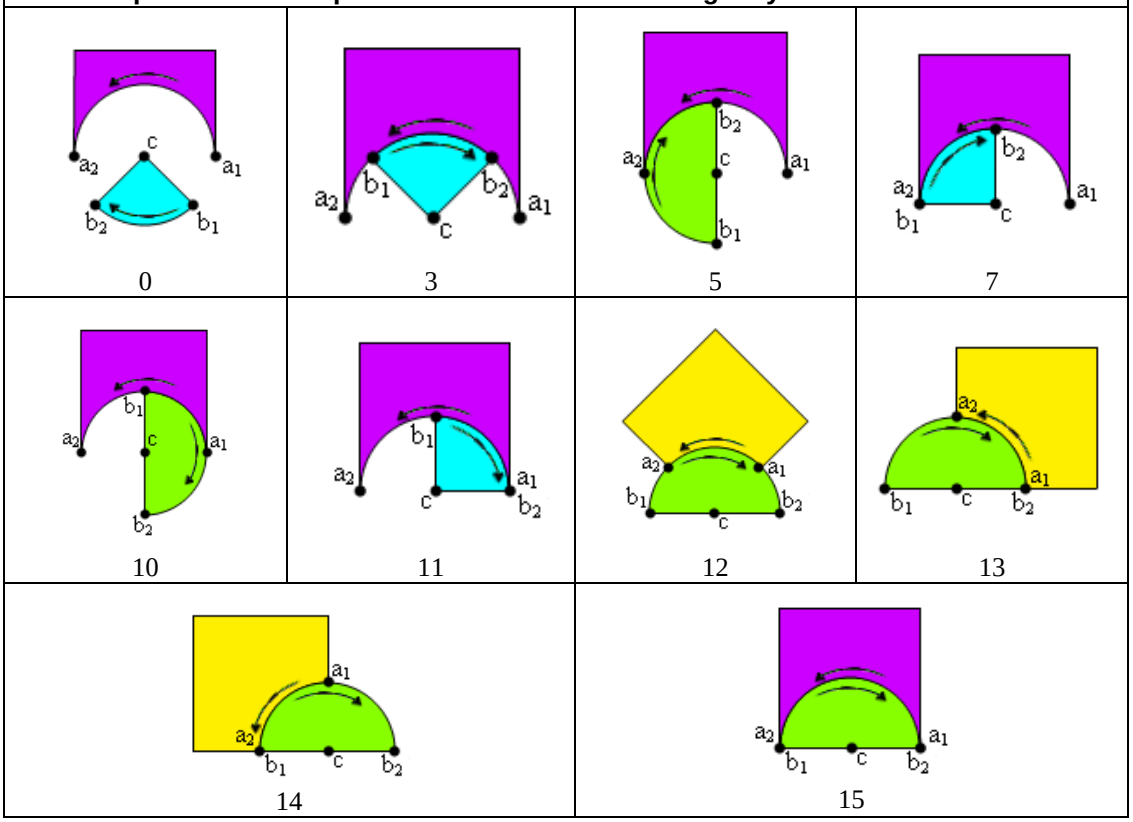


Table 2: Possible representations of circumference arc relations.

A Novel Multithreaded Rendering System based on a Deferred Approach

Jorge Alejandro Lorenzon
Universidad Austral, Argentina
jorgelorenzon@gmail.com

Esteban Walter Gonzalez Clua
Media Lab – UFF, Brazil
esteban@ic.uff.br

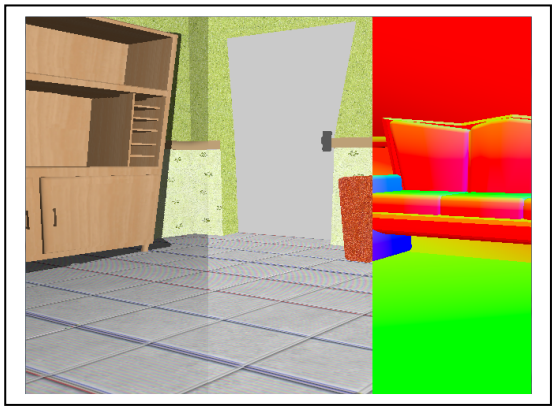


Figure 1: Mix of the final illuminated picture, the diffuse color buffer and the normal buffer

Abstract

This paper presents the architecture of a rendering system designed for multithreaded rendering. The implementation of the architecture following a deferred rendering approach shows gains of 65% on a dual core machine.

Keywords: multithreaded rendering, deferred rendering, DirectX 11, command buffer, thread pool

1. Introduction

Game engines and 3D software are constantly changing as the underlying hardware and low level APIs evolve. The main driving force of change is the pursuit of greater performance for 3D software, which means, pushing more polygons with more realistic models of illumination and shading techniques to the screen. The problem then becomes how to design 3D software in order to use the hardware to its maximum potential.

Central processing unit (CPU) manufactures are evolving the hardware to multi-core solutions. Currently dual core CPUs have become the common denominator while quad cores, like the recently released Intel Core i7 Extreme Edition processor, with the capability of running 8 processing threads, are slowly filling the high end market. In order for software to use all the capabilities and potential of the hardware it is now imperative that it divides its execution tasks among the different cores.

Therefore, the architecture of newer game engines must include fine-grained multithreaded algorithms and systems. Fortunately for some systems like physics and AI this can be done. However, when it comes to rendering there is one big issue: All draw and state calls must go to the graphics processing unit (GPU) in a serialized manner[1]. This limits game engines as only one thread can actually execute draw calls to the graphics card. Adding to the problem, draw calls and state management of the graphics pipeline are expensive for the CPU as there is a considerable overhead created by the API and driver. For this reason, most games and 3D applications are CPU bound and rely on batching 3D models to feed the GPU.

Microsoft, aware of this problem, is pushing forward a new multithreaded graphics API for the PC, Direct3D11. Direct3D11 was designed to remove the current restriction of single threaded rendering. It allows this by:

- Providing the ability to record command buffers in different threads
- Free threaded creation of resources and states

Command buffers are basically lists of functions to execute. There are two types of command buffers:

[1] This holds true for multi-GPU solutions, such as those that use SLI. The actual speedup with these setups (using alternate frame rendering) is accomplished by having the graphics API buffer commands for multiple frames so that each GPU can work on one. For this to happen effectively the application must not be limited by the CPU.

those that are part of an API and those that are not. The common benefit that both provide is that by deferring the communication with the GPU to a later stage, they allow the application to simulate multiple devices and divide its rendering work across multiple threads Natively supported command buffers can provide an additional performance benefit: Usually part of the graphic APIs' functions have a part that needs to be executed in the CPU like the validation of the data passed to them. So if the command buffers are designed to only hold commands ready to be executed by the GPU, the CPU load of the graphic APIs' functions can be processed at the time of their building, thus benefiting from the use of all the CPU cores. The execution of these buffers is then easier on the CPU leading to increased performance in CPU bound applications.

This paper proposes a new architecture of a multithreaded rendering system and shows the performance gains in different scenarios with an implementation based on DirectX11.

2. Related Work

Games traditionally use a game loop that execute update and render logic serially. The first approach in game engines to increase performance in multi-core hardware was to execute natural independent systems in parallel. The problem with this approach is that very few systems are independent from each other. For example: a particle system is independent from the AI system, however, the AI system is not independent from the physics engine as it needs to have the latest state of the world objects to compute the behavior of AI driven entities. The rendering and sound system need to have all the final data for the frame to present to the user so they depend on all of the systems. Thus, just multithreading independent systems is not an adequate enough solution for current hardware.

An engine has to be designed from the ground up with multiprocessing in mind to fully utilize multiprocessor hardware. There are two classic ways to approach this task: multiprocessor pipelining and parallel processing [Akenine-Moller et al. 2008]. Multiprocessor pipelining consists in dividing the execution in different stages so that each processor works on a different pipeline stage. For example: if the pipeline is divided into stages APP, CULL, and DRAW. For a given frame N, Core 0 would work on APP on frame N + 2, Core 1 would work on CULL on frame N + 1 and Core 2 would work on frame N. This architecture increases the throughput with the negative effect of increased latency. Parallel processing, on the other hand, consists in dividing up the work into small independent packages. This approach provides a theoretical linear speedup but requires for the algorithm to be naturally parallel.

Multithreaded engines have adopted different combinations of the techniques of multiprocessor pipelining and parallel processing. One approach has been to let each system of an engine run in a thread of its own [Gabb and Lake 2005]. In this solution systems use the latest available data for them, many times, like in multiprocessor pipelining, the data has been processed in a previous frame by a different system. The data independent systems benefit from the parallel processing speedup. Data sharing between systems is the biggest challenge in this type of architecture. Usage of synch primitives around shared data can be very expensive, so a buffering scheme is usually used to make it possible to for a system to write to a buffer while another system reads from a previously written buffer, thus avoiding heavy use of synch primitives. However, there are two problems with buffering schemes; firstly they utilize more memory, a scarce resource in some platforms. Secondly, copying of memory between buffers[2] might be expensive in terms of processing time [Lewis 2007]. In conclusion, this type of architecture is adequate while the number of systems is greater or equal to the number of cores. However, to scale further the game engine systems need to be designed to be internally multithreaded.

The system that this paper focuses on is the graphics system. The internal multithreaded architecture relies on the use of command buffers. The next paragraph gives an overview of the current support of command buffers under different platforms.

Microsoft's XBOX 360 DirectX and DirectX 11 support native command buffers [Lee 2008]. For other platforms, non-native command buffers can be used. The development team of Gamebryo has created an open source command buffer recording library for older versions of DirectX [Scheib 2008]. Their implementation currently only supports DirectX 9 but they are currently working on the implementation for DirectX 10. The multithreaded architecture of the rendering system is discussed in section 3, but before jumping to that section, it is important to read about the pipeline that will follow the rendering system.

A deferred rendering pipeline is used in the graphics system treated in this paper. This type of pipeline was chosen as many modern games, like S.T.A.L.K.E.R [Pharr 2005] and Tabula Rasa [Nguyen 2007] among others, have adopted this technique. Its main benefit is the decoupling of the geometry stage from the light stage [Deering et al. 1988]. This decoupling allows rendering to have a linear N + L complexity instead of the greater N * L complexity of the forward rendering approach, where N is the total number of objects and L is the total number of lights.

[2] Memory copying is necessary so that a system that reads, processes, and writes to buffer 1 is able to utilize the data written in buffer 1 in the next frame when it will need to do the same process using buffer 2.

Older games did not use this technique, as it requires the support of multiple render targets, extra video memory, and a higher memory bandwidth. However, with current hardware these requirements are no longer prohibitive.

3. Multithreaded Rendering System

Games usually have a 25% to 40% of the frame time used by the D3D runtime and driver [Davies 2008]. The overhead would not be such a problem if the engine could work on something else while the scene is being rendered. However, because the graphics system needs for the shared data to remain unmodified while it is doing its work the other engine systems become stalled. So if the graphics system is single threaded, the application wastes much of the CPU power. Therefore, the system architecture discussed in this paper is designed to utilize all of the CPU cores to create command buffers to give back to the other systems the ownership of the shared data as soon as possible. Once the buffers are created the "update" systems can run again while only one graphics thread remains submitting the command buffers to GPU. Figure 2 illustrates the flow described.

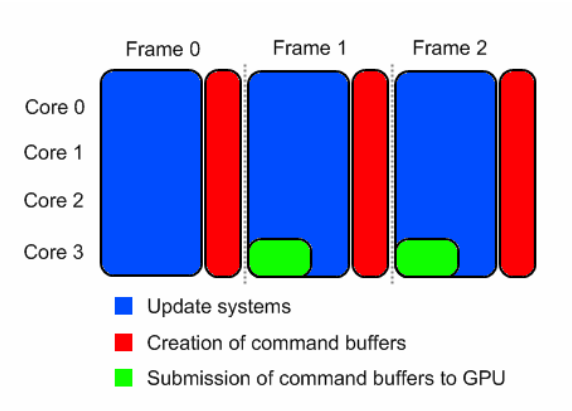


Figure 2: Top level application flow

The next sub section explains the approach and design used to build such a rendering system.

3.1 API abstraction layer

The first step in the design of the system is to abstract it from the API that will be used. This allows the software products that will later be constructed on top of the engine to be able to target more platforms.

Engines commonly grouped the creation of state and resources objects with the draw and state calls under a single rendering interface. The first design decision was to divide the responsibilities in two different interfaces: Device and Context. Device is in charge of the creation of resources and state objects while Context is responsible for the actual rendering. The Device is expected to be only instantiated once, while from Context many instances may be created; one for each thread that will submit rendering work. Having a mapping of one to one between contexts and threads is a necessary limitation to avoid the performance penalty that would appear from the need of synch primitives to maintain a rendering consistency. The Device, on the other hand, may be called from any thread. The best alternative, though, is to have an independent thread that manages the creation of resources. This could allow an application to go through a continuous world without needing to stop with loading screens.

The second step in creating the low level abstraction was to abstract resource and state objects. An abstract data type was declared for each of these. Adapter classes were created to extend from these abstract data types to make the adaptation necessary to communicate with the various graphic APIs. Figure 3 shows the UML class diagram for the depth stencil state.

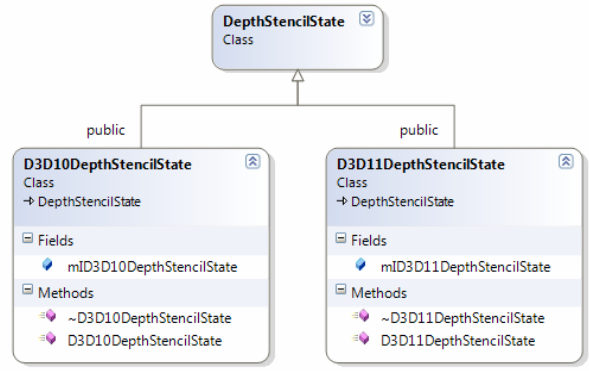


Figure 3: Depth stencil state abstraction

3.2 Rendering in different contexts

The instances of Context are the "renderers". As such, they are the ones that receive the messages to draw or change states. Having a number of them allows the application to submit geometry in parallel.

However, not all of them are equal, since there is **one** context with a special privilege: The Immediate Context. This is the only one that can effectively communicate with the GPU. The other contexts called Deferred Contexts submit state and draw calls to command buffers. Each of the deferred contexts then contains a command buffer ready to be executed by either another deferred context or by the immediate context. Therefore, the creation of command buffers becomes multithreaded while the submission of them is single threaded. The distinction of contexts is made because the nature of the CPU-GPU hardware allows only one CPU thread to send content to the GPU. The benefit in speed comes reducing the time that other systems remain stalled and from using idle CPU cores to help in the task of converting the state/draw calls from the API into a list of low level calls ready to be executed by the GPU. The immediate context can then execute the command buffers at a faster rate than the equivalent content via direct calls.

3.3 Graphics manager and render processes

In the previous section the low level abstraction layer was presented. Its designed was influenced greatly by the Direct3D11 API. In this section the next layer of the architecture will be presented. This layer will be responsible for the load balancing of rendering work.

Context and thread creation is not a lightweight task and so creating them to render every frame is not an option if we want to keep a high performance. The knowledge of how many contexts to create is part of the application design. Therefore, when initializing this layer that information will need to be communicated to it so it can allocate the resources needed at startup.

The Graphics Manager, represented in figure 4, is the central class of this layer. It is responsible for initializing a pool of threads and subsequently feeding them with the work that comes from the application. For each of the threads created a context is instanced and assigned to it. This ownership extends throughout the thread's life. Changing the ownership of the context is not possible because different threads may never make calls to the same context.

These threads remain asleep as long as no work is assigned to them. This prevents the Graphics Manager from consuming CPU cycles when the application is not rendering.

By making the number of possible worker threads variable the application developer has the freedom to choose as many threads as cores are available or any other number that the developer feels that it will provide a better performance.

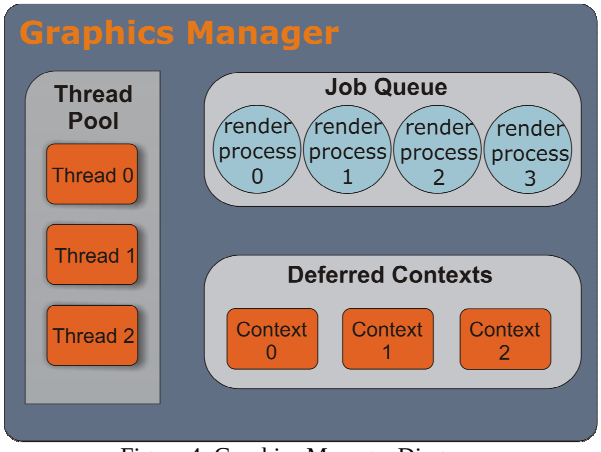


Figure 4: Graphics Manager Diagram

Because the developer should be limited as little as possible by the engine, the type of work that can be submitted to the pool can be as fine grained as executing a single draw call or as coarse as rendering a shadow map. The only limitation is that all work tasks need to be of the type Render Process.

The Render Process abstract class is the second most important class of this layer. Its importance is due to the ability that this class provides to application specific rendering work to be executed by the Graphics Manager through polymorphism.

The applications can define their rendering pipelines by creating and implementing subclasses of Render Process. The Graphics Manager commences the execution of the rendering work by calling the virtual method `Execute()` of Render Process from one of the pool threads.

The Render Process and the derived classes receive the context in which they can submit work as a parameter of `Execute()`. This frees the Graphics Manager from needing to keep a record of which context was used for each render process and allows it to do an optimal load balancing.

4. Multithreaded Deferred rendering

In the previous section we have discussed the architecture of the rendering system. In this section we will discuss how a deferred rendering pipeline fits on top and benefits from the multithreading rendering. A pipeline of this nature includes: a geometry buffer, lighting, transparency and post processing effects stages [Policarpo and Fonseca 2005]. The latter two were not implemented for the testing of the architecture. The following sub sections describe what the stages do and how they were encapsulated to be multithreaded rendered.

4.1 Geometry buffer

The geometry buffer (G-Buffer) creation is the first step in a deferred rendering pipeline. The purpose of the creation of the G-Buffer is to store the information necessary for the shading of each pixel during the lighting stage [Policarpo and Fonseca 2005]. The values that get stored depend on what illumination model the application will use. For the purpose of testing the performance of the architecture by keeping the application CPU bound a simple G-Buffer was used. The values stored are depths, normals and diffuse color.

The flow of data to the G-Buffer for every pixel is what consumes a lot of bandwidth. This is why before starting to create the G-Buffer itself it is better to have a rendering pass that just calculates the vertices' positions in order to set the Z-buffer. This pass, however, was not implemented for the test setup.

The steps to create the G-Buffer were all encapsulated in a class called Geometry Buffer Creator, which, extends from Render Process. The steps that it goes through are:

1. `Set viewport`
2. `Set render targets`

3. Set depth stencil state (read and write depth enabled)
4. Set rasterizer state (cull back, solid fill)
5. Clear the depth texture
6. Render each of the objects
7. Finish the command buffer

4.2 Lighting

Next is the lighting stage, where the application sends to the pipeline the lights that affect the scene. The effect that each light has on the pixels is calculated with the stored information in the geometry buffer and added to the final frame buffer.

4.2.1 Non-shadow casting lights

There are different alternatives to render non-shadow casting lights. The most efficient is to use geometry to represent lights [Calver 2003]. With this approach a spotlight is represented as a cone, a point light as a sphere and a directional light as a screen aligned quad. The benefit of using geometry is that the Z-Buffer rejects pixels more effectively than using scissors rectangles.

For the pixels that are not rejected by the scissor test or Z-Buffer a pixel shader that calculates how the light is influencing it is executed.

The process of rendering the non-shadow casting lights is isolated in another render process called Non Shadow Casting Lighting. The steps that this process goes through are:

1. Set viewport
2. Set final buffer as the render target
3. Set depth stencil state (depth test enabled, write disabled)
4. Set rasterizer state (fill solid)
5. Set the G buffer as shader resources
6. Render each light
7. Finish the command buffer

4.2.2 Shadow casting lights

The difference between non-shadow and shadow casting lights is that the latter have to calculate the obstruction of light due to the scene's geometry. One effective way to calculate this obstruction is through a technique called shadow mapping [Akenine-Moller et al. 2008]. Rendering the scene from the light's point of view while having writes and reads on the depth buffer enabled creates a shadow map. The shadow map contains depth information where the light is obstructed by geometry.

The shadowing lights' pixel shader, before shading, checks if the pixel is affected by the light or if it is in shadow.

The test application uses one shadowing light. It is a hemispherical light and the shadow map algorithm used was parabolic mapping. The process to create the shadow map and light the pixels was encapsulated in the Shadow Renderer class. The steps that it goes through are:

1. Set viewport
2. Set depth buffer
3. Set depth stencil state (read and write depth enabled)
4. Set rasterizer state (cull back, solid fill)
5. Clear depth buffer
6. Render all objects that casts shadows from the point of view of the light
7. Set the shadow map (depth buffer recently set) as a shader resource
8. Render light
9. Finish command buffer

5. Discussion

When designing the graphics system it was assumed that the application would give to it the total ownership of the CPU and data. With this in mind it, the architecture was built to fulfill two major objectives: stall the other application systems the less time possible, and provide transparent scalability throughout different platforms, current and future ones.

To make the system flexible enough it was designed to follow a producer-consumer model. The work products (encapsulated in render processes) are produced by the application and are consumed by the available threads. The means of distribution of the render processes is the Graphics Manager's responsibility. This decision was made to encapsulate the necessary platform dependent code that creates and manages threads. Clean encapsulation of platform dependent code makes not only porting, but also optimizations for different platforms easier.

It is important to note that if the number of cores increases beyond the number of stages that the rendering pipeline has, a further splitting will be needed. The split can occur at the application data level. For example, the Geometry Buffer Creator can be instanced twice so that each instance works on a subset of the application data. The split would be best done by object's material, this way the shader swapping in the GPU is kept to a minimum. Another possible split is to calculate the shadow maps for different shadow casting lights in parallel.

With a further increase in cores it will become harder to divide rendering work in an efficient way, as there are a limited number of materials used by objects or shadow casting lights at a given frame. So an option for the job-based architecture could be to start handling some real time raytracing for some effects like reflections and refractions.

6. Results

The tests were made on a Core 2 Duo E7200 CPU with an ATI Radeon HD4750 GPU. Microsoft's DirectX March 2009 SDK was used. Note that the DirectX11 version in this SDK is a tech preview. The hardware and drivers used are DirectX 10 level. DirectX 11 level GPUs are not yet available in the market.

The test application did not do any update to the objects in the scene so that the frame time was completely used by the rendering system. Even though, the objects did not move or update they were treated as dynamic. The application sent to the rendering system the objects that compose the scene shown in Figure 5.

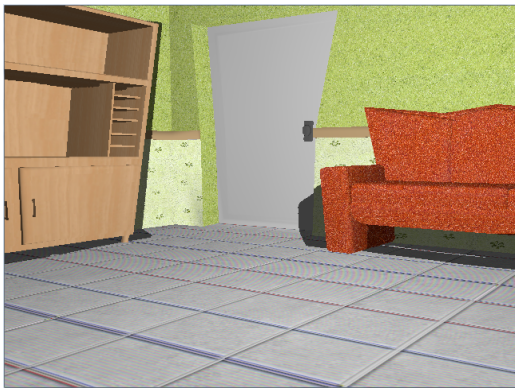


Figure 5: Test scene

Two scenarios were created to compare the system to the traditional ones. The first scenario consisted of using a single thread while the second one consisted in using the multithreaded solution.

In the first scenario the render processes that represent the deferred shading pipeline were executed in a sequential way in the main rendering thread with straight communication to the immediate context. This way is how traditionally games submit rendering work to the GPU.

In the second scenario, the Graphics Manager was initialized to work with 3 threads. At the start of each frame the render processes were queued in the Graphics Manager, which in this particular case used a Win32 thread pool as part of its implementation. When all of them were finished building their respective command buffers the main thread would submit these to the GPU through the immediate context.

The experiment was repeated with five variations. These variations were related to scene complexity. By scene complexity we mean the number of objects drawn. The tests were done with: 7, 16, 106, 1006 and 2006 objects. It is worth mentioning that the final image of the scene did not vary in the different tests as the added objects had the same position and mesh that the original ones. This way the Z-rejection hardware of the GPU would cull the objects before they reached the more processing intensive pixel shader stage. Doing this allowed our application to be CPU bound in the tests 1006 and 2006 objects.

The following chart (figure 6) shows the frames per second obtained by the rendering engine with the use of the multithreaded graphics manager versus the common single threaded solution. The single threaded results are shown by the green bar titled ST. The multithreaded ones are represented by the red bar titled MT.

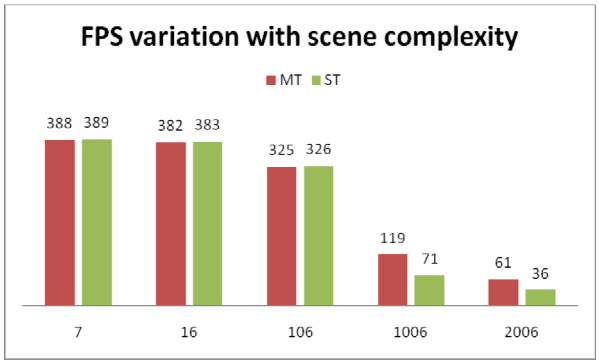


Figure 6: FPS variation with scene complexity.

The first 3 scenarios with 7, 16 and 106 objects show that the multithreaded design does not provide an improvement over the common single threaded one. The multithreaded solution was slightly worse than the traditional one. This is because the multithreaded solution does add a little overhead. Profiling showed that the CPU only used 15% of its capacity. The bottleneck in these tests, therefore, was created by the inability of the GPU to render the polygons faster.

The CPU starts to work harder when the number of objects increases, as it has to issue more draw calls, which take up CPU time. With 1006 objects, the CPU work created by the draw calls becomes sufficiently heavy to shift the bottleneck from the GPU to the CPU. Profiling of the single threaded scenario showed that one the cores was working 4 times more than the other, which was only running other program's processes in the back. The frames per second were in this scenario were 71 in average.

When switching to the multithreaded solution, the profiler showed a more even load among the cores and the frames per second rose around 65% to 119. The explanation to this is that, because the command buffers used were native to the graphics API, the load that each API call adds to the CPU was now being distributed along two cores.

With 2006 objects the ratio of frames per second between the two models only rose 2%. This evidenced that the two cores had hit their limit.

7. Conclusion

In a low scene complexity scenario the benefits of distributing the load of graphic API calls among

multiple cores is very low compared to the added overhead of running a more complex multithreaded system. Also, without other systems running there is no stall caused to them by the single threaded graphics system. So in this scenario the multithreaded solution has a clear disadvantage. With the test results showing a very slight decrease in performance for the multithreaded system. It is promising that with other systems running the application will have a better performance using a multithreaded graphics system than single threaded one.

In the high scene complexity scenario the results show the multithreaded design as clear winner with a 65% increase in speed. This lead would certainly increase with in an application that utilized other systems. The speedup shown by the results is significant considering that the API is still immature and that hardware and drivers were not the optimal for the multithreaded solution.

The speedup of the proposed graphics system will never be linear as there is still a part of the process that is single threaded, however, it is clear by analyzing the results that it is faster and more scalable than traditional ones.

In conclusion the multithreaded rendering solution based on a deferred rendering approach provides a promising solution for applications that need high performance and quality graphics.

Acknowledgements

The authors would like to thank André Luiz Brandão for providing scene geometry to test the architecture.

References

DEERING MICHAEL, STEPHANIE WINNER, BIC SCHEDIWY, CHRIS DUFFY, NEIL HUNT. "The triangle processor and normal vector shader: a VLSI system for high performance graphics". *ACM SIGGRAPH Computer Graphics* (ACM Press) **22** (4): 21–30. 1988.

PHARR MATT, FERNANDO RANDIMA. *GPU Gems 2*. Addison-Wesley Professional. 2005.

NGUYEN HUBERT. *GPU Gems 3*. Addison-Wesley Professional. 2007.

SCHEIB VINCENT. *Practical Parallel rendering with DirectX 9 and 10*. GameFest 2008. [online]. Available from: http://www.emergent.net/Global/Downloads/GameFest2008-ParallelRendering.pdf [Accessed July 23, 2009].

POLICARPO FABIO, FONSECA FRANCISCO. *Deferred Shading Tutorial*. SBGAMES 2005. [online]. http://www710.univ-lyon1.fr/~jciehl/Public/educ/GAMA/2007/Deferred_Shading_Tutorial_SBGAMES2005.pdf [Accessed July 23, 2009]

AKENINE-MOLLER TOMAS, HAINES ERIC, HOFFMAN NATY. *Real-Time Rendering*. Third edition. AK Peters. 2008.

HEIRICH ALAN, BAVOIL LOUIS. Deferred Pixel Shading on the PLAYSTATION 3. [online] http://research.scea.com/ps3_deferred_shading.pdf [Accessed July 23, 2009]

BRABEC STEFAN, ANNEN THOMAS, SEIDEL HANS-PETER. *Shadow Mapping for Hemispherical and Omnidirectional Light Sources*. [online] http://www.mpi-inf.mpg.de/~brabec/doc/brabec_cgi02.pdf [Accessed July 23, 2009]

GABB HENRY, LAKE ADAM. *Threading 3D Game Engine Basics*. November 17, 2005. [online] http://www.gamasutra.com/features/20051117/gabb_01.shtml [Accessed 3 September 2009]

ANDREWS JEFF. *Designing the Framework of a Parallel Game Engine*. February 25, 2009. [online] http://software.intel.com/en-us/articles/designing-the-framework-of-a-parallel-game-engine/ [Accessed 3 September 2009]

LEE MATT. *Multi-Threaded Rendering for Games*. GameFest 2008. [online] http://www.microsoft.com/downloads/details.aspx?FamilyID=DA8816C2-CFBE-4208-8DD8-9DEEA0C2E2B5&displaylang=en [Accessed 3 September 2009]

LEWIS IAN. *Multicore Programming Two Years Later*. GameFest 2007. [online] http://www.microsoft.com/downloads/details.aspx?FamilyID=DA8816C2-CFBE-4208-8DD8-9DEEA0C2E2B5&displaylang=en [Accessed 3 September 2009]

DAVIES LEIGH. *Optimizing DirectX on Multi-core architectures*. Game Developers Conference 2008. [online] http://software.intel.com/en-us/videos/optimizing-directx-on-multi-core-architecture-part-1/ [Accessed 4 September 2009]

CALVER DEAN. *Photo-realistic Deferred Lighting*. July 31, 2003. [online] http://www.beyond3d.com/content/articles/19/1 [Accessed 5 September 2009]

A Serious Game for Exploring and Training in Participatory Management of National Parks for Biodiversity Conservation: Design and Experience

Eurico Vasconcelos[1] Gustavo Melo[2] Jean-Pierre Briot[3] Vinícius Sebba Patto[3]
Alessandro Sordoni[3] Marta Irving[2] Isabelle Alvarez[3] Carlos Lucena[1]

[1] Pontifícia Universidade Católica (PUC-Rio), Computer Science Department, RJ, Brazil
[2] Universidade Federal do Rio de Janeiro (UFRJ), EICOS Program, RJ, Brazil
[3] Universitè Pierre et Marie Curie, Laboratoire d'Informatique de Paris 6 (LIP6), CNRS, France

Abstract

In this paper, we discuss the experience in the design, use and evaluation of a serious game about participatory management of national parks for biodiversity conservation and social inclusion. Our objective is to help various stakeholders (e.g., environmentalist NGOs, communities, tourism operators, public agencies, and so on) to collectively understand conflict dynamics for natural resources management and to exercise negotiation management strategies for protected areas, one of the key issues linked to biodiversity conservation in national parks. Our serious game prototype combines, techniques such as: distributed role-playing games, support for negotiation between players, and insertion of various types of artificial agents (decision making agents, virtual players, assistant agents). After a general introduction to the project, we will present project's current prototype architecture and results from game sessions, as well as some prospects for the future, namely: the design of assistant artificial agents and of virtual players and the integration of a viability-based simulation engine.

Keywords: Serious games, Simulation, Participatory management of Parks

1. Introduction

Serious Games [Michael and Chen 2006] are getting increased attention as a novel and effective approach for training and exploring possibilities, in context but without high costs or risks. Indeed, games are a good substitute for direct experience from real world or real infrastructures because they can generate learning experiences in a relatively fast and safe manner [Warmerdan et al. 2006].

In this paper, we will discuss our experience in the design of a serious game about participatory management of national parks for biodiversity conservation and social inclusion. Its main objective is to serve as an epistemic/educational tool. In this game, humans play some role and discuss, negotiate and take decisions about a common domain, in our case environment management decisions. More precisely, the idea is to help park managers, stakeholders and all researchers involved in park management, to explore and train about conflict identification, negotiation and decision strategies for management of parks, with the various perspectives involved, such as: biodiversity conservation, social inclusion and sustained development. This research project, named SimParc (which stands in French for "*Simulation Participative de Parcs*"), was started in 2006 in order to investigate the use of advanced computer techniques and methodologies (such as serious games) for participatory management of protected areas, more specifically national parks. The current SimParc serious game prototype is based on a role-playing game and computer techniques such as: distributed role-playing interfaces, negotiation support and artificial decision makers. Although intended, primary, to serve as epistemic tool, SimParc also has as ingredients the playfulness and the challenge of a game, presenting funny and interactive maps, negotiations and decision making in different and challenging phases.

2. The SimParc Project

2.1 Project Motivation

A significant challenge involved in biodiversity management is the management of protected areas (e.g., national parks), which usually undergo various pressures on resources, use and access, which results in many conflicts. This makes the issue of conflict resolution a key issue for the participatory management of protected areas [Irving 2006]. Methodologies intending to facilitate this process are being addressed via bottom-up approaches that emphasize the role of local actors. Examples of social actors involved in these conflicts are: park managers, local communities at the border area, tourism operators, public agencies and NGOs. Examples of inherent conflicts connected with biodiversity protection in the area are: irregular occupation, inadequate tourism exploration, water pollution, environmental degradation and illegal use of natural resources.

Our SimParc project focuses on participatory parks management. It is based on the observation of several case studies in Brazil. However, we chose not to reproduce exactly a real case, in order to leave the door open for broader game possibilities. Our project aim is to help various stakeholders at collectively understand conflicts and negotiate strategies for handling them.

2.2 Related Work

Our initial inspiration was the ComMod approach about participatory methods to support negotiation and decision-making for participatory management of renewable resources [Barreteau 2003]. They pioneer method, called MAS/RPG, consists in coupling multi-agent simulation (MAS) of the environment resources and role-playing games (RPG) by the stakeholders [Barreteau 2003]. The RPG acts like a "social laboratory", because players of the game can try many possibilities, without real consequences.

Recent works proposed further integration of role-playing into simulation, and the insertion of artificial agents, as players or as assistants. Participatory simulation and its standard bearer, the Simulación framework [Guyot and Honiden 2006], focused on a distributed support for role-playing and negotiation between human players. All interactions are recorded for further analysis (thus opening the way to automated acquisition of behavioral models) and assistant agents are provided to assist and suggest strategies to the players. The Games and Multi-Agent-based Simulation (GMABS) methodology focused on the integration of the game cycle with the simulation cycle [Adamatti et al. 2007]. It also innovated in the possible replacement of human players by artificial players. One of our objectives is to try to combine their respective merits and to further explore possibilities of computer support.

3. The SimParc Role-Playing Game

3.1 Game Objectives

Current SimParc game has an epistemic objective: to help each participant discover and understand in a playful way the various factors, conflicts and the importance of dialogue for a more effective management of parks. Note that this game is not (or at least not yet) aimed at decision support (i.e., we do not expect the resulting decisions to be directly applied to a specific park).

The game is based on a negotiation process that takes place within the park council. This council, of a consultative nature, includes representatives of various stakeholders (e.g., community, tourism operator, environmentalist, nongovernmental association, water public agency). The actual game focuses on a discussion within the council about the "zoning" of the park, i.e. the decision about a desired level of conservation (and therefore, use) for every sub-area (also named "landscape unit") of the park. We consider nine pre-defined potential levels (that we will consider as types) of conservation/use, from more restricted to more flexible use of natural resources, as defined by the (Brazilian) federal bureau of environment management. Examples are: Intangible, the most conservative use, Primitive and Recuperation.

The game considers a certain number of players' roles, each one representing a certain stakeholder. Depending on its profile and the elements of concerns in each of the landscape units (e.g., tourism spot, people, endangered species…), each player, as in a RPG has to embody the designed/selected role with its respective postures and objectives. To facilitate the incorporation of the role by the player, SimParc offers a set of personas to represent him/her during the game (Figure 1). Based on the role, the player will try to influence the decision about the type of conservation for each landscape unit. It is clear that conflicts of interest will quickly emerge, leading to various strategies of negotiation (e.g., coalition formation, trading mutual support for respective objectives, etc).

Figure 1: Some examples of personas offered in SimParc.

A special role in the game is the park manager. He is a participant of the game, but as an arbiter and decision maker, and not as a direct player during negotiation and interaction phase. He observes the negotiation taking place between players and takes the final decision about the types of conservation for each landscape unit. His decision is based on the legal framework, on the negotiation process between the players, and on his personal profile (e.g., more conservationist or more open to social concerns) [Irving 2006]. He may also have to explain his decision, closing the game cycle. The park manager may be played by a human or by an artificial agent (see Section 6).

3.2 Game Cycle

The game is structured along six steps, as illustrated in Figure 2. At the beginning (step 1), each participant is associated to a role (randomly selected, selected by the administrator or by the participant him/herself.). Then,

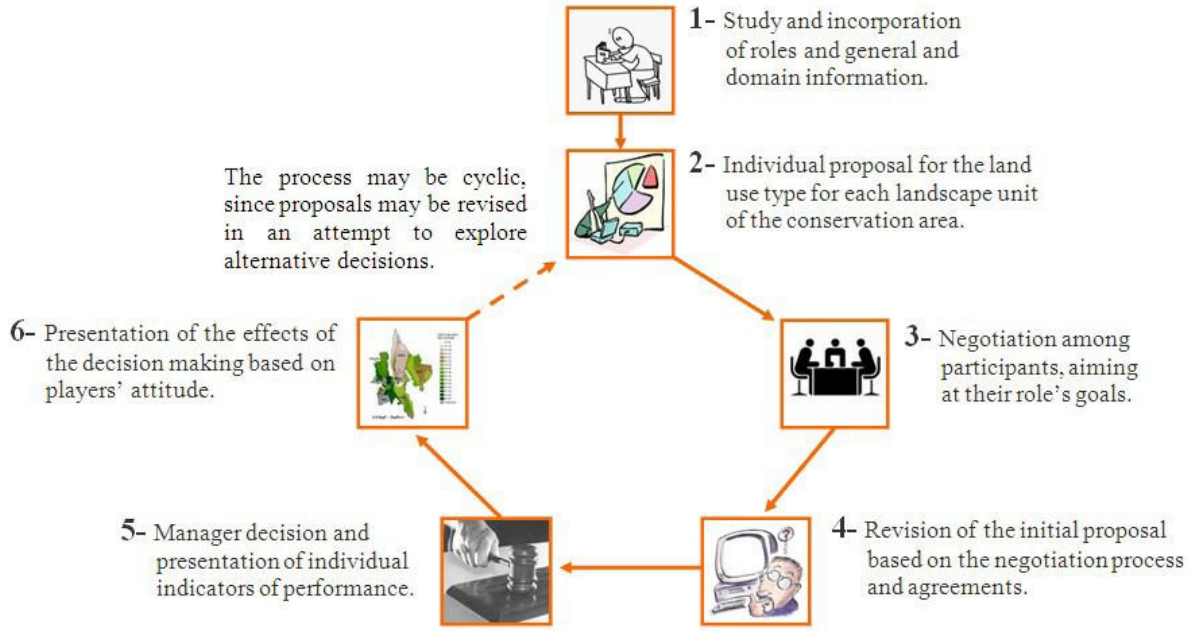


Figure 2: The six steps of the SimParc game.

an initial scenario is presented to each player, including the setting of the landscape units, the possible types of use and the general objective associated to his role. This information is present on a map of a fictitious park that should be "traveled" by the players, by clicking in the elements presented in the map. Then (step 2), each player decides a first proposal of types of use for each landscape unit, based on his/her understanding of the objective of his/her role and on the initial setting. Once all players have done so, each player's proposal is made public.

In step 3, players start to interact and to negotiate on their proposals. This step is, in our opinion, the most important one, where players collectively build their knowledge by means of an argumentation process. In step 4, they revise their proposals and commit themselves to a final proposal for each landscape unit. In step 5, the park manager makes the final decision, considering the negotiation process, the final proposals and also his personal profile (e.g., more conservationist or more sensitive to social issues). Each player can then consult various indicators of his/her performance (e.g., closeness to his initial objective, degree of consensus, etc.). He can also ask for an explanation about the park manager decision rationales.

The last step (step 6) "closes" the epistemic cycle by considering the possible effects of the decision. In the current game, the players provide a simple feedback on the decision by indicating their level of acceptance of the decision.[1]

A new negotiation cycle may then start, thus creating a kind of learning cycle [Kolb 1984]. The main objectives are indeed for participants: to understand the various factors and perspectives involved and how they are interrelated; to negotiate; to try to reach a group consensus; and to understand cause-effect relations based on the decisions.

4. The SimParc Game Support Architecture

Our current prototype benefited from our previous experiences (game sessions and a first prototype) and has been based on a detailed design process. Based on the system requirements, we adopted Web-based technologies (more precisely J2EE and JSF) that support the distributed and interactive character of the game as well as an easy deployment.

Figure 3 shows the general architecture and communication structure of SimParc prototype version 2. In this second prototype, distributed users (the players and the park manager) interact with the system mediated internally by communication broker agents (CBA). The function of a CBA is to abstract the fact that each role may be played by a human or by an artificial agent. For each human player, there is also an assistant agent offering assistance during the game session.

During the negotiation phase, players (human or artificial) negotiate among themselves to try to reach an agreement about the type of use for each landscape unit (sub-area) of the park. We include below two screen dumps to provide a quick idea about current interface support and their look and feel. The interface for negotiation is shown at Figure 4. It includes advanced support for negotiation (rhetorical markers and dialogue filtering/structuring mechanisms, see details in [Vasconcelos et al. 2009]), access to different

[1] A future plan is to introduce some evaluation of the quality of the decision. See Section 7.3.

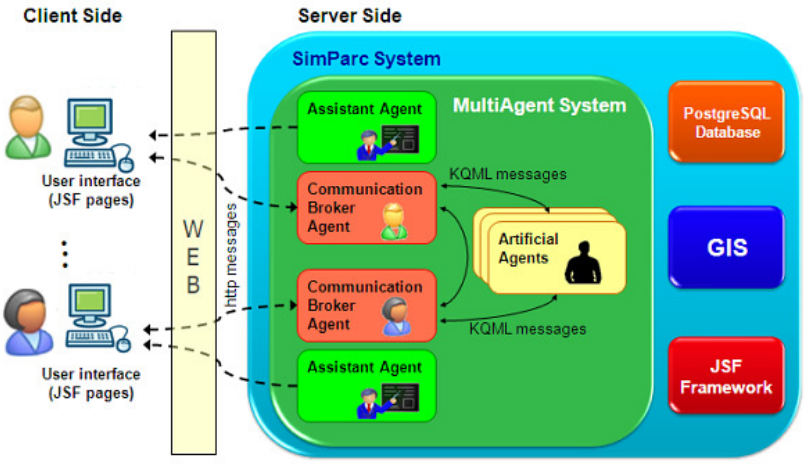


Figure 3: SimParc version 2 general architecture.

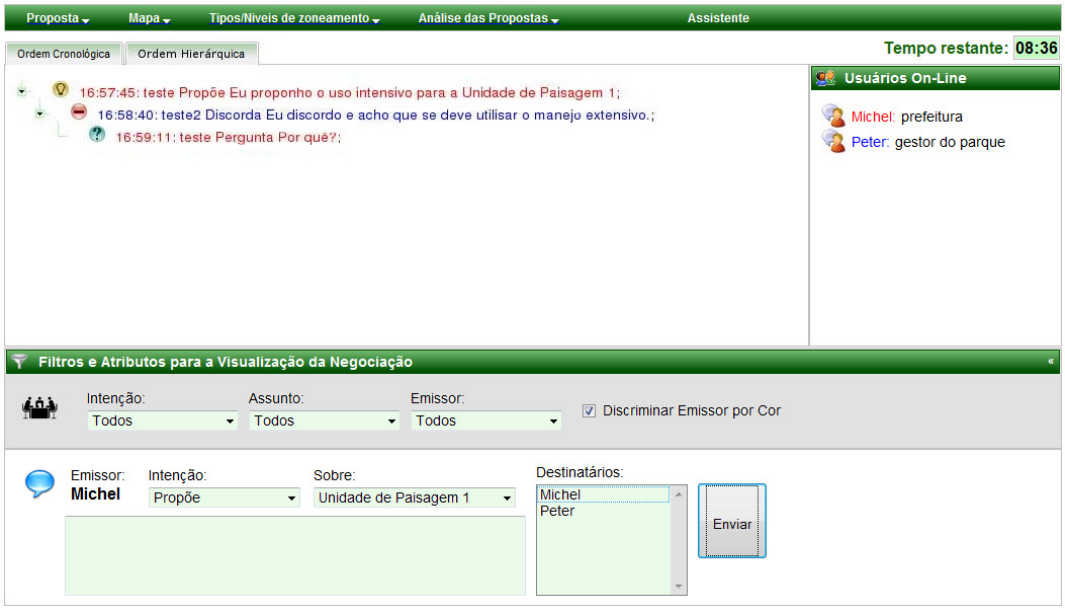


Figure 4: Current prototype's negotiation graphical user interface.

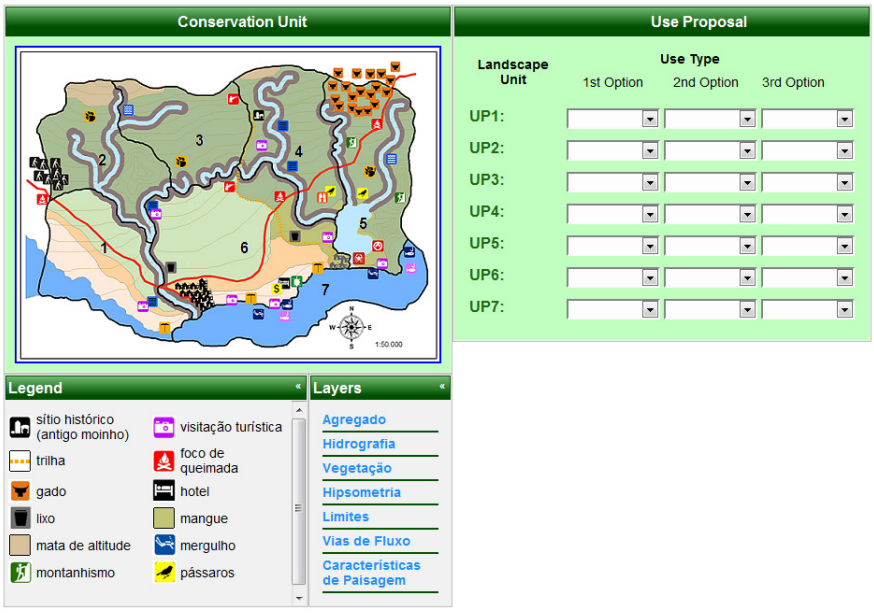


Figure 5: Current prototype's decision graphical user interface.

kinds of information about other players, land, law and the help of the personal assistant. The negotiation and its interface are detailed in [Vasconcelos et al. 2009]. The interface for players' decision about the types of use is shown at Figure 5. In the interface, the players can analyze the area based in its different layers (e.g. land, hydrography, vegetation…).

A Geographical Information System (GIS) offers to users different layers of information (such as flora, fauna and land characteristics) about the park geographical area. All the information exchanged during negotiation phase, namely users' logs, game configurations, game results and general management information are recorded and read from a PostgreSql database. Report functionality also allows the game manager to get structured information about each game session.

5. Preliminary Evaluation

The current computer prototype has been tested through two game sessions by domain expert players (including a professional park manager) in January 2009 (see Figure 6). The 9 roles of the game and the park manager were played by humans. Among these 10 human players, 8 were experts in park management (researchers and professionals, one being a professional park manager in Brazil). The two remaining players were not knowledgeable in park management. One was experienced in games (serious games and video games) and the other one a complete beginner in all aspects.

Figure 6: SimParc current prototype game session (2009).

Two aspects of the game were positively evaluated by the participants of the game session: the structure, (script, rules and set tasks) and content (scenes, conflicts, environmental management). Through successful integration of structure and content, SimParc was evaluated as a game that reached the goal of creating, in fact, a "virtual arena" of management. Although the game does not constitute a tool for decision-making directly applicable to real parks, but only a support for epistemic and pedagogical goals, it was highlighted as a positive aspect the proximity of SimParc as a virtual scenario with the reality of park management, making it more attractive for those people working directly in real parks.

In the analysis of the test, two key aspects to the improvement of the game play were highlighted: information and interaction. About information, were considered the conditions for access to content, form, data quality, the quantity of information available, among others. Regarding interaction, were mainly considered the resources and tools available to help players negotiate.

The information is certainly the key to support the SimParc game. Mainly because the game is structured through the use of different terminologies, that in the background are the basis for negotiation between the players. The large volume, complexity of information and conflicts illustrated, that require an understanding by each player, were one of the main problems identified by participants of the test.

In conceptual terms, the biggest difficulty encountered by participants was the understanding of all the different types of zoning (9 types: Intangible, Primitive…). Therefore, it was highlighted the importance of improving access to information for each player, especially those that explain the different types of zoning. The proposal is to make the players consult with more comfort and efficiency information about the game. For example, it was considered essential that the proposed zoning of each other possible player could always be viewed with the changes visible in "real time" in order to stimulate diversity of strategies for one player, since, that way, each player will be able to see how others players are defending their interests.

The interaction between the players is also a key element in improving SimParc. Considering that the game requires a continuous and dynamic interaction between players, it has been highlighted the importance of the use of flexible systems with additional features such as hyperlinks to send messages directly to a player and use of tools that allow the creation of parallel trading rooms. According to participants, this may facilitate and enhance the negotiation process, an important process in the game.

Aiming to investigate whether SimParc is approaching its epistemic and pedagogical goals, participants were asked about what would be the main goal of the game. The responses were related to the following themes: management practice involving negotiation between different social actors, understanding and experience of different roles that facilitate the learning of the practice of dialogue and negotiation, illustrating the dynamics of conflict, learning environmental expertise and park management, and dissemination of the importance of environmental preservation. In the interpretation of the

players about if the game had reached such objectives, the players felt that yes, the game was a great exercise for negotiation, with active interaction and interest of players, further encouraged by the possible exchange of roles.

Participants also reported that the main knowledge gained after the game was related to the territorial zoning process of parks, mainly for the players who did not have advanced knowledge on the subject of environmental management. Even those players, who work directly in environmental management, or research related to the subject, explained that they acquired more knowledge about the specific characteristics of each type of zoning commonly used in parks. It was also mentioned that the game could be considered as an exercise on process and techniques of negotiation, although the game does not suggest any technique to the players.

Another point was mentioned in relation to recognition of the diversity of interests in the management of a park. Even though most players knew many of the conflicts illustrated by the game (political, environmental degradation, misuse, etc.), they mentioned that it was possible to improve their analysis based on different roles and groups of social actors that the game presented. Besides the importance of conducting further tests, it was considered an important aspect of the game, the fact that the game is hosted using the Internet, instead of the requirement to install a program on computers, which means greater mobility for applications and larger dissemination of this game.

6. Park Management Artificial Agent

As explained in Section 3.1, the park manager acts as an arbitrator in the game, making a final decision for types of conservation for each landscape unit and explains its decision to all players. He may be played by a human or by an artificial agent. The game manager decides when creating and configuring a new game session about the park manager, see Figure 7.

The artificial agent's architecture is structured in two phases. The first decision step concerns agent's individual decision-making process: the agent deliberates about the types of conservation for each landscape unit. Broadly speaking, the park manager agent builds its preference preorder over allowed levels of conservation. An argumentation-based framework (see, e.g. [Dung 1995]) has been implemented to support the decision making. The key idea is to use the argumentation system to select the desires the agent is going to pursue: natural park stakes and dynamics are considered in order to define objectives for which to aim. Hence, decision-making process applies to actions, i.e. levels of conservation, which best satisfy selected objectives. The second step consists in taking account of players' preferences, with the possibility to adjust the profile of the park managers, from autocratic to democratic, and therefore the influence of players' votes. (See details of the complete architecture in [Briot et al. 2009]).

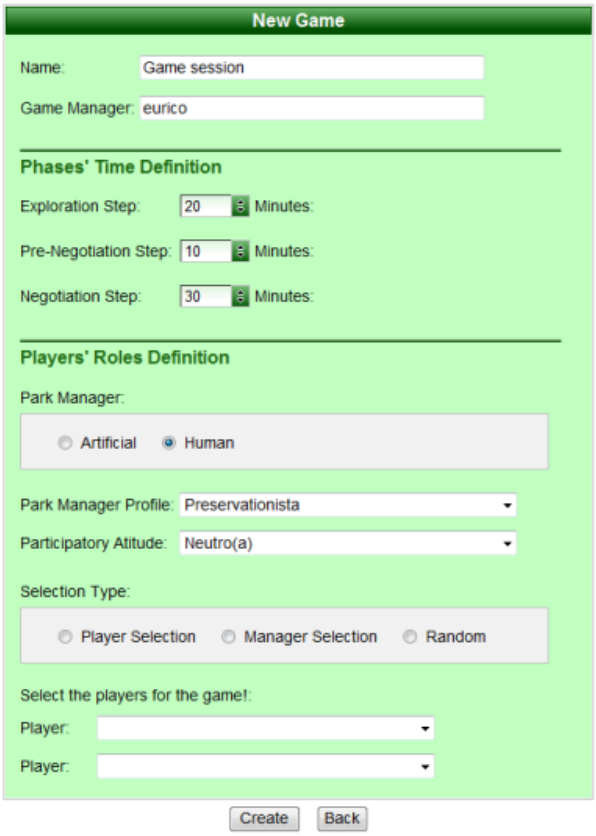


Figure 7: New game configuration interface.

Further details about architecture formal background and implementation are reported in [Briot et al. 2009]. The architecture has been implemented and tested offline and its outputs (decision and arguments) have been validated by our project domain experts. Next step is to organize a new series of game sessions, with an online test of the artificial park manager architecture. Some possible future work is also to use the traces of arguments produced for the decision as a basis for the explanation of the decision to players.

7. Ongoing Work and Future Prospects

We are currently planning on inserting other types of artificial agents into the prototype.

7.1 Artificial Players

Artificial players represent an ongoing research based on previous experience on virtual players in a computer-supported role-playing game, JogoMan-ViP [Adamatti et al. 2007]. The idea is to possibly replace some of the human players by artificial agents. The two main motivations are: (1) the possible absence of

sufficient number of human players for a game session and (2) the need for testing in a systematic way specific configurations of players' profiles. The artificial players will be developed along artificial park manager existing architecture (see previous section), with the addition of negotiation and interaction modules. We plan to use the argumentation capabilities to generate and control the negotiation process. In a next stage, we plan to use automated analysis of recorded traces of interaction between human players in order to infer models of artificial players. In some previous work [Guyot and Honiden 2006], genetic programming had been used as a technique to infer interaction models, but we also plan to explore alternative induction and machine learning techniques, e.g., inductive logic programming.

7.2 Assistant Agents

The assistant agents are being designed to assist players through the game. It is important to emphasize that the user has total control over his assistant, enabling or disabling it at anytime. The basic initial function of these agents is to present and explain each step of the game. During the negotiation step, assistant agents may also propose to participants some helpful information, in order to improve their analysis concerning the negotiation. For instance, they may provide each player with an ordered list of the players taking into account criteria such as the compatibility or incompatibility of proposals of other players with the proposals of the assisted player. Since we decided to favor a bottom-up approach, we decided to avoid intrusive assistant agents through the game. We believe that intrusive assistant agents could interfere in the players' cognitive processes. That is why our assistant agents cannot suggest players a decision. A first implementation has already been completed and we will soon start to test it through small game sessions.

7.3 Expert Agents

Last, we are starting to work on expert agents which will provide the human players (including the park manager if played by a human) with some technical evaluation of the quality and viability of a given park management decision (e.g., considering the survival of an endangered species). Therefore, we plan to identify cases of usage conflicts (e.g., between tourism and conservation of an endemic species) and model the dynamics of the system (in an individual-based/multi-agent model or/and in an aggregated model). We would then like to explore the use of viability theory [Aubin 1992] to evaluate possible effects of the decision. These technical evaluations would be encapsulated into expert agents, technical assistants for the players. Another considered type of expert agent will be based on decision theory analysis, for instance to evaluate the dominance relations or equity properties among players votes.

8. Conclusions

In this paper, we have presented the SimParc project, a serious game aimed at participatory management of protected areas. We have also summarized the architecture of an artificial decision maker park manager. The first game sessions conducted with domain experts have been successful. It is important to emphasize that the game SimParc was developed based on the recovery of initiatives for the construction of methodologies which help to consolidate democratic spaces of decision in cases of protection of nature. In this sense, the game intends to be a tool capable of contributing to the dialogue on consolidation of commitments to conservation, particularly management of national parks and other protected areas. Although this is an innovative proposal, with wide application in the present context, the experience has shown that quick and simple solutions to modeling the complexity of this process can become a great risk of loss of meaning of the game. Considering that the game could be played too by real managers, it is important to reflect how far the game, that is fun and educational, should be closer to reality and what are the necessary representations/abstractions to achieve the required goals. For example, how the process of negotiating social pacts and democratic management of protected areas can be promoted without losing the focus on respect to real problems and operational by the tax legislation and guidelines for management? Similarly, how to balance technical and scientific expertise in the social participation in the management of nature? Although more evaluation is needed, we believe the initial game session tests are encouraging for the future and we are welcoming any feedback and input from similar or related projects.

Acknowledgements

The authors would like to thank to all the members of SimParc team and all the participants of game sessions for their help.

This research has been initially funded by the ARCUS Program (French Ministry of Foreign Affairs, Région Ile-de-France and Brazil) and is currently being funded in Brazil by the MCT/CNPq/CT-INFO Grandes Desafios Program (Project No 550865/2007-1) and in France by the Ingénierie Ecologique Program of CNRS & Cemagref (Project Viabilité SimParc). Some additional individual support is provided by French Ministry of Research (France), AlBan (Europe), CAPES and CNPq (Brazil) PhD fellowship programs.

References

ADAMATTI, D., SICHMAN, J., and COELHO, H. 2007. Virtual players: From manual to semi-autonomous RPG. In Barros, F., Frydman, C., Giambiasi, N., and Ziegler, B., editors, International Modeling and Simulation Multiconference (IMSM'07), Buenos Aires, Argentina,

The Society for Modeling Simulation International (SCS), February, p. 159-164.

AUBIN, J.-P. 1992. Viability theory. Modern Birkhäuser Classics.

BARRETEAU, O. 2003. The joint use of role-playing games and models regarding negotiation processes: characterization of associations. Journal of Artificial Societies and Social Simulation, Vol. 6, No 2.

BRIOT, JP., SORDONI, A., VASCONCELOS, E., MELO, GUSTAVO AND IRVING, M., AND ALVAREZ, I. 2009. Design of a decision maker agent for a distributed role playing game – Experience of the SimParc project. In Dignum, F., Bradshaw, J., Silverman, B., and van Doesburg, W., editors, Proceedings of the AAMAS Workshop on Agents for Games and Simulations (AGS'09), Budapest, Hungary, AAMAS, May, p. 16-30.

DUNG, P. M. 1995. On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games. Artificial Intelligence, 77(2):321–357.

GUYOT, P. and HONIDEN, S. 2006. Agent-based participatory simulations: Merging multiagent systems and role-playing games. Journal of Artificial Societies and Social Simulation, 9(4).

IRVING, M. (Org.). 2006. Áreas Protegidas e Inclusão Social: Construindo Novos Significados. Rio de Janeiro: Aquarius.

KOLB, D.A. 1984. Experimental learning: Experience as the source of learning and development. Prentice Hall.

MICHAEL, D. and CHEN, S. 2006. Serious Games – Games that Educate, Train and Inform. Thomson Course Technology.

VASCONCELOS, E., BRIOT JP., IRVING, M., BARBOSA, S., FURTADO, V. 2009. A user interface to support dialogue and negotiation in participatory simulations. In Nuno David and Jaime Simão Sichman, eds, Multi-Agent-Based Simulation IX - International Workshop, MABS 2008, Estoril, Portugal, May 12-13, 2008, Revised and Invited Papers, LNAI, No 5269, Springer-Verlag, p. 127-140.

WARMERDAM J., KNEPFLÉ M., BIDARRA R., BEKEBREDE G. AND MAYER I. 2006. Sim-Port: a multiplayer management game framework. In Proceedings of the 9th International Conference on Computer Games (CGAMES'06), Dublin, Ireland.

An Integrated Development Model for Character-based games

Marconi E. Madruga Filho, Allan J. S. Souza, Patrícia C. A. R. Tedesco, Danielle R. D. Silva, Geber L. Ramalho

Universidade Federal de Pernambuco, Centro de Informática, Brazil

Abstract

Character-based games are strongly dependent on the quality of their Non Player Characters' behavior. Developing games of this nature usually requires a high investment of time and money on Artificial Intelligence techniques, in order to provide better credibility to the characters. For the independent game developers' community, affording this kind of extra complexity may be highly expensive. Therefore, this article proposes a way of developing character-based games while maintaining high quality and low-cost. This is done by using a modular Artificial Intelligence technique called Rule-Based Systems, integrated with a game development support tool, XNA. Since Java is better provided with Rule-based System tools, the IKVM application is used to allow this integration. All mechanisms and tools used are free. A comparison of free Rule-based Systems is presented, as well as an application of the proposed model on a real game project.

Keywords: virtual characters, artificial intelligence, character-based games, game development process

Authors' contact:
{memf,ajss,pcart,drds,glr}@cin.ufpe.br

1. Introduction

The development cost of a commercial game has been steadily increasing. More and more, predictions come up stating that developing for the next-generation consoles will cost more than double the current-generation value [Morris 2009]. Furthermore, the demand for more realistic games also increases. This realism is not only about good-looking scenarios and game environments, but also about game characters' behavior and intelligence. This is the case of character-based games, like the The Sims [Maxis 2000] series or even Role-playing games, which are games that rely mainly on their characters' illusion of life shown to the player. This search for behavioral realism normally entails a demand for a more sophisticated game AI to define the characters. Furthermore, the AI sophistication creates new challenges for the game development, as well as a higher game development cost.

For independent game developers, this extra cost may cause the development of good character-based games to be too expensive. These developers always look for ways to produce games while executing the hard task to maintain the balance between high quality and low monetary and time cost. One way of doing so is to use less expensive AI techniques that still have great power to simulate characters' behavior, as is the case of Rule-based Systems (RBS). In this case, RBS need to have good performance, documentation and tool support, and easy integration, in order to give a better support to character-based game development.

Therefore, this article shows a comparative study of various available RBS tools, based on the requirements related to character-based game development. By using the results of this study, the article proposes a process that supports the modeling of characters with rich behavior. This is done by using free platforms and tools. This approach is also base for a character-based training game, VTEAM [VTEAM 2009], whose goal is to improve the players' capacity of managing human resources. Then, the characters are modeled richly using true personalities which are commonly found on working teams, in order to create conflicts and situations to be surpassed by the player.

This paper is organized as follows. Section 2 illustrates the use of Artificial Intelligence in character-based games, especially Rule-based systems. Section 3 discusses choosing an adequate RBS and game development technology. The character-based game development model is proposed on section 4. The study on applying it on a real case is presented on Section 5 and section 6 shows conclusions about the research and possible future works.

2. Character-based games and Independent Development

Character-based games are the type of game that rely strongly on the behavior of its non-player characters (NPCs). These are games like Black & White [Lionhead 2001], by Lionhead Studios, where the player interacts with an autonomous character which may behave differently, depending on the player's actions. The illusion of life and intelligence passed by the game's autonomous characters captivates the player and makes the game more interesting. However, to achieve this effect it is necessary to reach a certain level of realism on the character's behavior. Thus, character-based games require a strong focus on their character's modeling and on thinking about how these

characters are going to make the player feel like dealing with a living being.

This credibility is shown when the character's behavior is adaptable. Some attempts have been done to develop this kind of behavior in games. In the beginning this was done using simple AI techniques such as Finite-state Machines (FSM) [Millington 2006], as done in classic games such as Pacman [Namco 1980]. However, the amount of work required to model character behavior using FSM depends on the size of the problem. As the demand for believability increased, modeling behavior using FSMs became more complicated.

Therefore came up the need to incorporate more sophisticated AI techniques such as Bayesian Networks, machine learning, fuzzy logic, rule-based systems and others. Among the available approaches, rule-based systems are, probably, the most used [Bourg and Seemann, 2004]. One of their main advantages is that they model the way how people usually think, since the rules are written in declarative language. For this reason, they are easy to work with, as well as flexible and powerful to solve problems of various natures. Rule-based systems have been used in many games [Cavazza 2000], such as Civilization: Call to Power [Activision 1999] and Rainbow Six [Red Storm 1998].

On the other hand, investing in RBS or any other artificial intelligence technique might add an extra cost to game development. This goes against the demand from independent developers' community for low-cost solutions and mechanisms to develop character-based games. For this reason, even when choosing RBS for their technical advantages, it is necessary to consider how much one has to spend in order to include this approach in one's project.

2.1 Rule-based Inference Engines

Rule-based systems (RBS), also known as Production Systems, are already a very popular AI technique, due to being easy to implement and powerful. The control of the RBS rules is made by an inference engine, the heart of the system. The rules are, in general, condition-action pairs; the inference engine checks the rule conditions and, if they are true, the rule is fired, which means the rule actions are executed. One can opt to implement an inference engine from scratch to use in one's game, but there is also already a great variety of implementations available, such as Drools [JBoss, 2009] and Microsoft's BizTalk [Microsoft 2009], even though they are not directed specifically to game development. When choosing an inference engine, the developer must keep in mind the character's demand for realism and believability.

Based on our previous development experience, on character-based games, we have elicited some requirements for choosing RBS in Character-Based games. They are:

- **Embeddability**: The language in which the rules are written must be easy to comprehend. It may be close to natural language or to a standard programming language. This rule language should be implemented integrating concepts of object orientation and production rules, a reusable and easily integrated approach known as Embedded Object Oriented Production Systems (EOOPS) [Pachet 1995]. On EOOPS, due to their integrated behavior, objects from the programming language might be used on the rule conditions and actions, preserving and dealing with the aspects of object-oriented language (encapsulation, inheritance, and so on). For instance, object's fields can be checked on the rule conditions and object's functions can be called on the rule actions. This facilitates rule writing and reuse.
- **Performance:** Since the goal is to develop games, performance seems to be RBS's main issue. It is not useful for a game to have great character behavior realism if it will sacrifice the game's performance. The RETE algorithm is the pattern matching method most commonly used and has already been shown to be efficient [Forgy 1982].
- **Development support:** Even when there is good performance, readability and integration, a fine support for developers is essential. This includes precise documentation, additional tools, development support features such as rule debugging. In addition, integrated environments and support tools are also helpful and should be taken into account. Besides that, it is preferable to choose active RBS projects, for they have more intense updating, bug correction and community support.
- **License cost:** A free engine is vital to avoid adding additional costs associated with the rule engine. This is fundamental when development is independent or academic.

	Platform	Embeddability	Performance	Docs & Tools	License
Drools.NET	.NET	High	+(RETEOO)	None	Open-source
NxBRE	.NET	Poor	Unknown	Medium	Open-source
Simple Rule Engine	.NET	Medium	Unknown	Poor	
Drools Expert	Java	High	+(RETEOO)	High	Open-source
JEOPS	Java	High	+(RETE)	Medium	Open-source
Jess	Java	Poor	+(enhanced-RETE)	High	Academic use
Soar	Independent	Poor	+(RETE)	High	Open-source

Table 1: Comparison between free rule-based systems

Besides the above criteria, the choice of the inference engine is also associated with the development platform on which the game will be developed. The choices of an adequate RBS as well as of the development platform are discussed on the next section.

3. Developing a Character-based game

Character-based game development clearly stands for a hybrid of game and intelligent system development. Hence, requirements come from both sides. From games, comes the need to use a proper game development platform, with support to audio and input control, as well as other game development features. Besides, from intelligent systems, comes the need for choosing an appropriate AI technique to implement, on this case, Rule-based systems. The selected RBS and game development platform must be decided, in order to develop an idea of development process for this kind of games.

3.1 Choosing an RBS

There are many RBS available. A great number of them, though, are proprietary, like Microsoft's Business Rule Framework [Microsoft 2009], ILOG Rules for .NET [ILOG 2009], and Jinni [BinNet Corporation, 2005] and do not conform to the requirements of independent or academic development. Among the free rule-based inference engines available, seven were selected and they are described in the list below. These RBS were evaluated using the requirements listed on section 2.1; the result is detailed below and compiled in the Table 1.

- **Drools.NET (.NET):** Drools.NET [Drools.NET 2006] is an open-source .NET implementation of a java inference engine named Drools. It is out-of-date, compared to its java companion, has poor unfinished documentation and does not have tool support. Its strong point is the embeddability, since it is an EOOPS and hence has high integration with code. Its rule language is a clean union of C# and predicate logic. Drools aims to have good performance by implementing the RETE algorithm. It has not been updated since 2007.
- **NXBRE (.NET):** NXBRE [NxBRE 2009] is an inactive rule-based inference engine for .NET. Unlike Drools.NET it is well documented and its rule language is XML-based, but it has no tool support either. Objects referenced by the rules must be written in XML, which means the developer cannot make a direct integration between the objects used inside the code and the objects used by the rules. Hence, NXBRE is not an EOOPS, an essential feature when regarding embeddability. It is open-source too, but its matching algorithm is not known.
- **Simple Rule Engine (.NET**): Simple Rule Engine (SRE) [Sierra digital solutions 2005] is another open-source rule-based inference engine written in .NET. It is, like NXBRE, based on xml, but its objects may come from C# code, making it easier to embed. This project claims to be better in performance than NxBRE. SRE is another case of nearly abandoned project with extremely poor documentation and tool support.
- **Drools Expert (Java):** Drools Expert [JBoss 2009] is an object-oriented rule engine which is also open-source. It uses the RETE-OO algorithm - an optimization of Forgy's original algorithm using Object Oriented Programming concepts, intending to have high performance. It has the option of writing rules in XML as well as in the "native" rule language DRL (Drools Resource Language). It also allows the user to create DSLs (Domain Specific Languages) and is fully integrable with Java code. Drools has great development support, with rule debugging and rule authoring tools and an extensive documentation. It is possibly the most complete option available.
- **JEOPS (Java):** JEOPS [Figueira and Ramalho 2000] is an embedded object production system, whose rule language syntax is Java-based. Unlike the other rule engines, in JEOPS, the rule base file is compiled to a Java file to be used in the application. It is also open source and well

documented, but it has no tool support. It implements the classic version of the RETE algorithm.

- **Jess (Java):** Jess [Friedman-Hill 2008] uses an enhanced version of the RETE algorithm to process rules, claiming to have better performance than most engines. It also manipulates and reasons about Java objects, which demonstrates high embeddability. The rules can be expressed in XML-like or Lisp-like languages, both designed for the Jess engine. It has extensive documentation and some additional tools such as IDE integration. Although it is not free, Jess was included in this evaluation since it is available at no cost for academic use.
- **Soar:** Soar [Laird et al. 1987] Soar is a cognitive architecture for applications with intelligent behavior, in general. It has tool support and is extensively documented. Besides, it is platform independent, working as service that can be accessed by any system. However, this broadness makes it necessary to develop a protocol of communication between Soar and the application, which raises integration work significantly.

Regarding documentation and tools, Drools Expert and Jess stand out, while the .NET counterpart of Drools stays in the last position. Most of the verified engines implement the RETE algorithm or enhanced versions of it, making performance comparisons harder. In addition, Drools and JEOPS are the engines that can be easily classified as EOOPS. Although Soar has tool support and documentation, it has some integration issues. Thus, it is not hard to notice that Java inference engines are the ones that better fit the elicited requirements for developing a character-based game with low cost. This is natural, given the existence of a great variety of open-source consolidated inference engines in Java [Stamper 2008]. However, to choose an inference engine it is still necessary to define which game development technology will be used.

3.2 The game development platform

There are many game development platforms available today, but none of them is specifically directed to character-based game development. Hence it is necessary to choose, from the general purpose development options, one that is free, easy to use and integratable with the inference engine. Among the various options, Microsoft's XNA has attracted attention of the independent game developers' community, even though it is not a game development industry standard. This is due to XNA's good learning curve (coming from C#.NET's ease of use), game development support as well as to it being free. In addition to this, the XNA creators club [XNA 2009], the official community of XNA game developers, has over 300 published games and 100,000 users.

XNA is a set of managed code development libraries that intends to make it possible for game developers to be more productive when creating games for Windows and the Xbox 360. It encapsulates low-level technological components involved in coding a game, allowing developers to focus on the content and gaming experience. The XNA Framework is part of XNA Game Studio toolset, which works with Visual C# Express and Directx SDK, all free tools provided by Microsoft [MSDN 2009].

The best-fitting inference engines, though, are not written in .NET. But using the .NET inference engines, which are less adequate, in order to match XNA may compromise the main characteristic of character-based games, which is the credibility of the NPCs' behavior. Soon comes up the need to integrate XNA's pro-game technology and the simulation capabilities of the Java inference engines.

4. The proposed model

When designing a development model proposal that matches character-based games, it is necessary, as pointed out previously, to integrate Java inference engines and XNA game development framework. The latter is responsible for supporting game development and the former for assuring the quality of the character-based game development main requirement: character behavior credibility. In order to achieve this integration, IKVM.NET [Frijters 2008] toolset was used. It can compile Java packages (JARs) to .NET libraries and also allows .NET code to access the Java API, promoting the communication between both sides.

In the proposed approach, hereafter referred to as Cross-platform Reasoning Support Model, the compiler is used to convert the Java inference engine and its related classes (packed in a JAR file) to a .NET library (as described in Figure 1), which can be accessed in the .NET environment. Thus, the code written in C# over the XNA platform can communicate directly with the inference engine. Hence, the developer can benefit from the strong points of both sides, being able to fulfill the requirements of character-based games.

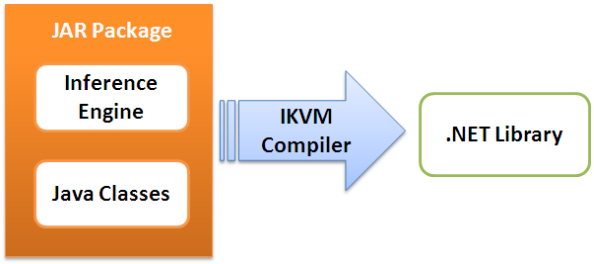


Figure 1: Compiling a Java Engine to .NET

This approach has weak spots, though. It may sacrifice one of the criteria used to evaluate de quality of inference engines: the tool support. This occurs

because the Java engine-related tools only work in the Java environment. By compiling the engine to .NET, all game components stay in .NET, becoming unable to access the Java toolset. And during the development phase it is extremely important to have features such as rule debugging, rule logging, authoring tools and so on.

In order to bypass this problem and be able to use all the engine advantages, a specific model directed to the development phase of the game is also proposed.

4.1 Cross-Platform Solution

For the purpose of using the tools provided by the Java engines to help the development, it is necessary to run parallelly the Java and .NET environments. Therefore, specifically during the game development stage, the already mentioned approach is not appropriate, since it focuses on fully developing in the .NET environment.

The solution to avoid this is a new approach that keeps the .NET and Java components apart and implements communication between them using the Java Remote Method Invocation (RMI) [Sun 2006]. It is a way of communication that makes it possible for two objects in different virtual machines to contact each other, locally or remotely. By doing so, through the use of RMI service provided by IKVM, the components in .NET are able to communicate with the Java engine, and the two environments can work simultaneously and individually. This can be done simply by using a remote engine interface in .NET that will work as a communication protocol. The process is described in Figure 2.

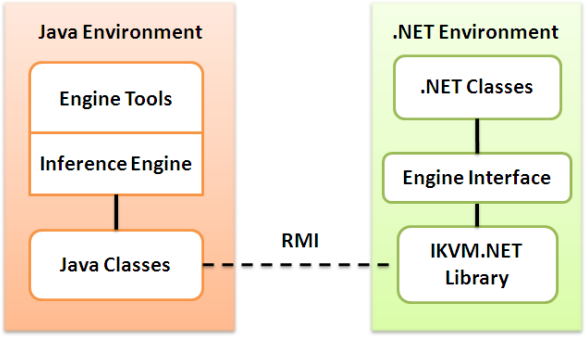


Figure 2: Communication between Java and .NET environments

By using this process, it is possible, for instance, to debug rules, and thus raise the chance of bug resolution during the development phase of the game. However, since communication using RMI may have a negative impact on the game performance, it is recommended to use this model only during the development phase. At the release phase, compiling the engine to .NET maintains the necessary functionality and attenuates the effects of adding the inference engine to the game.

Next section presents Virtual Team, a game prototype where the proposed approach was successfully applied.

5. VTEAM

Figure 3: Main game screen of VTEAM

Virtual Team [VTEAM 2009] (Figure 3), a.k.a. VTEAM, is a serious game prototype created to assist software project managers' training. During the game, the player incorporates a project manager whose goal is to finish a project while trying to deal with personal issues of the development team. The player has to assign tasks, help the team when needed, promote outstanding members, and arrange meetings, among other actions, in order to finish the project successfully.

It focuses on improving the manager's ability to deal with problems related to Human Resources Management. VTEAM's goal is to provide for the manager being trained a greater experience of organizational, methodological, cultural and personal processes that are generally characteristic to software development teams. This mechanism offers to both trainer and trainee a richer experimentation scenario when compared to traditional methods and simulators used to teach those concepts.

VTEAM is also a character-based game, since the interaction between player and virtual characters is its main feature. In this game, the characters are represented by Synthetic Actors - intelligent agents with special features, such as emotions, personality and beliefs [Silva 2009]. As in any character-based game, the NPCs (team members) need to have behavior credibility, especially when this will have great impact on the player learning curve. Hence the need for a strong focus on the character's AI.

RBS were chosen to model the characters' behavior, mainly due to the ease of representing character knowledge using rules. Moreover, RBS usually do not require the developers to be experts on the subject, because declarative language is easy to understand. For the purpose of guiding the game development, the approach proposed in this game was applied.

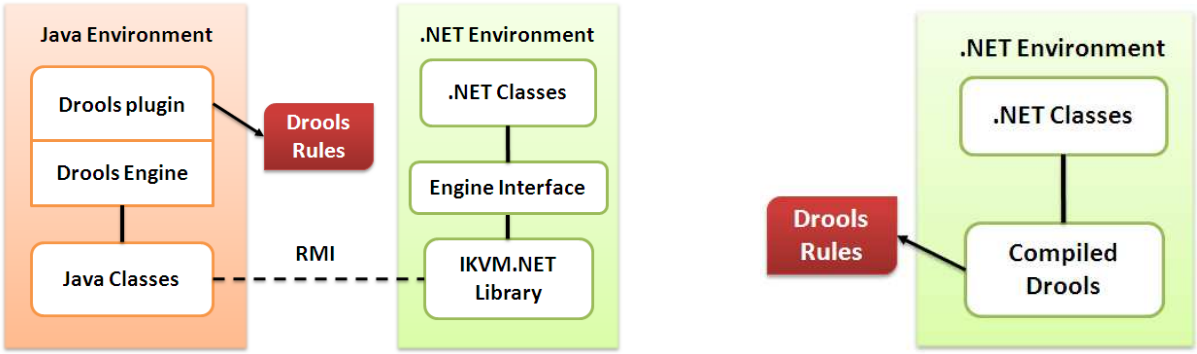


Figure 4: VTEAM's communication with Drools on development phase (left) and release phase (right)

5.1 Cross-platform Reasoning Support Model on VTEAM

In order to apply the proposed approach to VTEAM's prototype, it was necessary to look for an adequate inference engine. At the first version of the game, developed in C++, the Soar engine was used to model the character's behavior. Soar works like a black box, completely separated from the game code, and in order to use it properly components of communication were implemented. Besides that, the Soar rule language is not very readable, since it uses its own concepts regarding operators and automatic state generation.

Thus, on this version of the project, a different engine was chosen, Drools Expert. Drools rule language is a hybrid between Java and declarative programming, making it easy for developers to understand it. Besides that, Drools can be used via an integrated eclipse environment, which adds important features such as on-line working memory viewing and rule debugging - the latter is important, since VTEAM deals with a large number of variables and has an intense need for game balance. Figure 4 illustrates the implementation of the proposed approach on VTEAM.

VTEAM's prototype was developed on the .NET framework using the Visual C# Express IDE, due to the game development support provided by XNA. In order to integrate XNA technology and the Drools engine, the proposed process was used. During the development phase, Drools is executed from eclipse, while the game code is in Visual C# Express. For deploying, Drools is compiled into a .DLL, used by the C# code. So far, it is possible to say that the application of this article's proposal to VTEAM was successful.

One of the main concerns when working with an RBS is its impact on game performance. In order to investigate this issue, the prototype FPS rate was measured using the proposed approach. There were no changes in FPS (Frames per second) rate when using the release phase model. And, as expected, the use of RMI caused the FPS to fall during the development phase. But to bypass this problem, the game and the inference engine interface were implemented in two different threads, running in parallel.

6. Conclusion

As the demand for high quality and innovative games increases, so does the game development cost. Due to this state of affairs, independent game developers look for low-cost ways to produce high quality games. In the case of character-based games, this quality is deeply associated with the characters realism. Not only visual details, but also, and more importantly, believable behavior transmitted by the characters, provided by Artificial Intelligence. Thus, it is necessary to find ways of including this extra complexity avoiding a negative impact on financial cost.

The proposed approach allows the creation of a character-based game, based on a powerful AI mechanism: Rule-based systems. This is done using only free tools in a cross-platform way, bypassing the need for additional cost on development. However, this approach does have some drawbacks. In order to implement a cross-platform approach, it is necessary for the development team to know both languages involved, a demand that might be a problem. In this research's case, Java and C# are very similar languages [MSDN 2009], which attenuates this problem

As future work, it is important to test the study with a complete game and a large amount of rules implemented, so that the real impact on performance might be verified. Besides, it is also important to run tests with various magnitudes of games, which have different requirements on AI. Character-based games that call for interaction between virtual characters, like The Sims, for instance, might have different needs compared to those where the character only interacts with the player and the scenario, like Black & White and Nintendogs [Nintendo 2005].

This article study might also be used on developing games that have a rule module, but not necessarily associated to intelligent agents, such as The Distributor Game [van Houten et al 2005], which uses rule-based

scenarios. Besides, the model that used RMI, might also allow that one Java inference engine works as a server for many XNA game clients. These approaches have yet to be tested, though.

Acknowledgements

First of all the authors would like to thank the VTeam development team and project coordinators for all the effort and support. Also thanks to the project partners and sponsors: FINEP, Jynx playware, Valença & Associados, Qualiti, CNPq and CIn-UFPE.

References

MORRIS, C., 2009. *The Next Generation of Gaming Consoles* [online] CNBC. Available from: http://www.cnbc.com/id/31331241 [Accessed in 24 July 2009]

MAXIS, 2000. *The Sims.* [computer game]. Redwood City, USA: Electronic Arts.

LIONHEAD, 2001. *Black & White*. [computer game]. Redwood City, USA: Electronic Arts.

NAMCO, 1980. *Pac-man*. [computer game]. Tokyo, Japan: Namco.

MILLINGTON, I. 2006. Artificial intelligence for games. San Francisco: Morgan Kaufmann.

BOURG, D. M. AND SEEMANN G., 2004. AI for game Developers. Sebastopol, USA: O'Reilly.

ACTIVISION, 1999. *Civilization: Call to Power* [computer game] Santa Monica, USA: Activision.

RED STORM ENTERTAINMENT, 1998. *Tom Clancy's Rainbow Six* [computer game] Morrisville, USA: Red Storm Entertainment.

CAVAZZA, M., 2000. AI in computer games: Survey and perspective. *Virtual Reality* 5 (4), 223-235.

PACHET, F., 1995. On the embeddability of production rules in object-oriented languages. *Journal of Object-Oriented Programming*, 8 (4), 19-24.

FORGY, C. L., 1982. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19 (1), 17-37.

MICROSOFT, 2009. *Microsoft Biztalk Server 2009: Business Rule Framework* [online]. Available from: http://www.microsoft.com/biztalk/en/us/business-rule-framework.aspx/ [Accessed in 24 July 2009].

ILOG, 2009. *ILOG rules for .NET* [online]. Available from: http://www.ilog.com/products/rulesnet/ [Accessed in 24 July 2009].

BINNET CORPORATION, 2005. *Jinni: Java INference Engine and Networked Interactor* [online]. Available from: http://www.binnetcorp.com/Jinni/ [Accessed in 24 July 2009].

DROOLS.NET, 2006. *Drools.NET home* [online]. Available from: http://droolsdotnet.codehaus.org/ [Accessed in 24 July 2009].

NXBRE, 2009. *NxBRE home* [online]. Available from: http://nxbre.org/ [Accessed in 24 July 2009].

SIERRA DIGITAL SOLUTIONS, 2005. *Simple Rule Engine* [online]. Available from: http://sourceforge.net/projects/sdsre/ [Accessed in 24 July 2009].

JBOSS, 2009**.** *Drools: Business Logic Integration Platform* [online]. Available from: http://www.jboss.org/drools [Accessed in 24 July 2009].

FIGUEIRA FILHO, C. AND RAMALHO, G., 2000. JEOPS - The Java Embedded Object Production System. In: Monard M. and Sichman J., eds. *Advances in Artificial Intelligence*, 1952. London: Springer-Verlag, 52-61.

FRIEDMAN-HILL, E., 2008. *Jess, the Rule Engine for the JavaTM Platform* [online]. Available from: http://www.jessrules.com/ [Accessed in 24 July 2009].

CARNIEL, M., 2006. *JRuleEngine – OpenSource Java Engine* [online]. Available from: http://jruleengine.sourceforge.net/ [Accessed in 24 July 2009].

STAMPER, J., 29 July 2008. *The 10 Best Open Source Rules Engines. Jason Stamper's Blog* [online]. Available from: http://www.businessreviewonline.com/os/archives/2008/07/10_best_open_so.html [Accessed in 24 July 2009].

XNA, 2009. *XNA Creators Club* [online]. Available from: http://creators.xna.com/en-US/ [Accessed 23 June 2009].

MSDN, 2009. *XNA Developers Center* [online]. Available from: http://msdn.microsoft.com/en-us/xna/default.aspx [Accessed 23 June 2009].

SUN, 2006. *Java Remote Method Invocation* [online]. Available from: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp. [Accessed 24 July 2009]

VTEAM, 2009. *Projeto VTEAM* [online]. Available from: http://vteam.cin.ufpe.br. [Accessed 24 July 2009].

SILVA, D. R., 2009 *Atores Sintéticos em Jogos Sérios: Uma abordagem baseada em Psicologia Organizacional*. PhD Thesis. Universidade Federal de Pernambuco.

MSDN, 2009. *The C# Programming Language for Java Developers* [online]. Available from: http://msdn.microsoft.com/en-us/library/ms228602.aspx [Accessed 24 July 2009]

NINTENDO, 2005. *Nintendogs*. [computer game]. Kyoto, Japan: Nintendo.

FRIJTERS, J.., 2008. *IKVM.NET* [online]. Available from: http://www.ikvm.net/ [Accessed 24 July 2009]

LAIRD, J., E., NEWELL, A. AND ROSENBLOOM, P.S., 1987. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1), 1-64.

VAN HOUTEN, S.P.A., VERBRAECK, A., BOYSON, S., AND CORSI, T., 2005. Training for Today's Supply Chains: An Introduction to the Distributor Game. *In: Proceedings of the 2005 Winter Simulation Conference,* 04-07 December 2005 Orlando. 2338-2345.

An open-source framework for air guitar games

Lucas S. Figueiredo João Marcelo X. N. Teixeira Aline S. Cavalcanti Veronica Teichrieb Judith Kelner

Universidade Federal de Pernambuco
Centro de Informática
Grupo de Pesquisa em Realidade Virtual e Multimídia

Figure 1: *Left: example application being controlled with two yellow gloves. Right: scratch of a game using the platform.*

Abstract

This paper presents an open-source framework for developing guitar-based games using gesture interaction. The goal of this work was to develop a robust platform capable of providing seamless real time interaction, intuitive playability and coherent sound output. Each part of the proposed architecture is detailed and a case study is performed to exemplify its easiness of use. Some tests are also performed in order to validate the proposed platform. The results showed to be successful: all tested subjects could reach the objective of playing a simple song during a small amount of time and the most important, they were satisfied with the experience.

Keywords:: air guitar, music game interaction

Author's Contact:

{lsf, jmxnt, vt, jk}@cin.ufpe.br, aline@gprt.ufpe.br

1 Introduction

Over the past few years, game industry has added peripheral tools to games enhancing gameplay experience. These peripherals have also increased levels of active engagement with games and widened the appeal of games to audiences who may never have played them before. Examples of such tools include the Eye Toy [Entertainment 2007], Dance Mats [Byrne 2004], the Wii motion controller [Nintendo 2006], shown in Figure 2, among others. One of the most successful examples is the evolution of game interaction with the advent of music games.

Figure 2: *Console peripherals, from left to right: PS3 Eye Toy, Dance Mat and Wii basic controllers.*

The concept of music games refers to video games in which the gameplay is meaningful and often almost entirely oriented toward the player's interactions with a musical score or individual songs. In the last decade, they have conquered a significant space on game market, and in consequence, game industry is now focusing its efforts on musical instruments and simplified devices that favor user interaction.

Video game interaction represents a research area in constant evolution. It has started with game-specific, few functional controllers, to more sophisticated ones, including features such as motion sensors and adaptable design formats, as shown in Figure 3. Recently, Microsoft presented Project Natal [Microsoft 2009], a system containing depth-sensors, a camera and a microphone (shown in Figure 4), which claims to be capable of replacing conventional controllers. Such technology would allow a person to use his/her own arms and legs to directly interact with the game.

Figure 3: *Controller evolution example, from the Atari controller (left) to the sophisticated Wii Wheel and Wii Zapper ones (right).*

Figure 4: *Project Natal sensors.*

This "controller-less" approach may be applied to games similar to Guitar Hero or Rock Band. The idea of playing an "air guitar" refers to imitate rock guitar gestures along music without the actual physical instrument. It is more showmanship than musical performance and does not require real musical skills. Instead of just acting along with the imaginary guitar, the user could control an actually playable "virtual" instrument, capable of emitting sounds according to his/her movements.

Concerning the project of an "Air Guitar game", it is possible to emphasize some requirements in order to obtain a robust and usable application:

- Interaction has to occur in real time. Since the user is playing music, the time constraint is very strict and implies that the delays on visual and audio responses are minimum. This may be obtained by using simple image processing techniques, capable of executing on most computer configurations.

- User actions may mimic the act of playing a real guitar. There is no meaning to create an "air guitar" application if users must press buttons instead of holding an instrument and actually playing it.
- Realistic sound response is required. The sound output may reflect what should happen in case the user was playing a real guitar. Simplifications can be used, in order to make the interaction easier, but the sensation of immersion may only be reached by correct action-reaction correspondence.

In this context, an open-source framework for developing guitar-based games using gesture interaction is proposed. The goal is to develop a robust platform capable of satisfying the previously mentioned requirements that can be easily extended and usable in order to allow integration in interactive applications such as music games. The main contribution is the framework itself, since no similar open-source tool was found in the literature. Making available such tool should drive the gaming community to develop more and better music games using the "controller-less" approach.

This paper is organized as follows. The next section presents some related work regarding music games, specifically about guitar-based applications. Section 3 describes the proposed framework architecture, the main technologies and the techniques used. Section 4 shows as case study an example application and the results obtained performing user tests. Finally, some final remarks are presented in Section 5 and future works are pointed in order to improve the proposed framework.

2 Related Work

Many works share their piece of contribution to the air guitar framework's idea. Among them, music games play their role as the main motivator of this work. Due to the increased popularity of this type of games in the last few years, it has been conquering space in the current game market [Wixon 2007]. Adagio [Games 2007], GuitarFreaks [Konami 1998], FreQuency [Systems 2001] and DJ Hero [Europe 2009] are some games that can be cited, varying from simple flash-based web implementations to more sophisticated ones made for the latest console generations.

The games cited before served as a starting point for the evolution that was going to happen. Frets on Fire (an open-source music game for PC) [Voodoo 2006] suggested a different use for the keyboard, in which the player should hold it using both hands, simulating an electric guitar. Games such as Guitar Hero [RedOctane 2005] and Rock Band [MTV Games 2007] improved user interaction by adopting simplified musical instruments for game input. Consequently, they remained for a long time on the top of sold games, and still are a fever among players scattered all over the world.

Starting with plastic guitars, the use of specific input devices simulating musical instruments for game control has been popularized. Nowadays, it is possible to form an entire band by the use of drums, microphones and guitars (see Figure 5), all played by different users. As a way of improving or evolving the current user interaction available on games, especially on music games, a new type of interaction was necessary. The concept of the proposed framework was influenced by a great number of input devices and user interaction methods, such as Dance Mats, Wiimotes and the recently announced Project Natal (presented at the E3 conference in June 2009 and not yet available). From those three aforementioned, Project Natal is the one with closest relation to the proposed work, since it also deals with the recognition of player's movements without using wires or buttons for interaction. Another example of body motion capture is found in [Camurri et al. 1999]. They proposed a system that recognizes some user intentions into a dancing performance and uses this information as input for some tasks, i.e. sound generation.

The Virtual Air Guitar [Mäki-Patola et al. 2005], [Karjalainen 2006] project is the related work that mostly approximates to this work. In fact, it implements basically the same features that are available on the proposed framework, and also uses the same image processing methods. However, since The Virtual Air Guitar project

Figure 5: *Input devices for music games. Electric guitars joysticks made based on a Gibson SG on the left, and on the right a simplified drums joystick.*

happens to be a proprietary solution, there is no code neither an executable demo application available to the community. The same occurs with the Virtual Slide Guitar project [Pakarinen et al. 2008], a similar work that differs basically by the capture method, using infra-red sensors. There is no downloadable source code or a demo application of it. So, some of the major contributions of this framework is the fact that it is open-source, makes use of a simple but robust color tracking algorithm (which favors application performance), interprets movements like a virtual guitar being played, and provides developers with a hands-on demo application, that can be modified and used as a start point for more complex simulations.

Besides applications similar to guitar games, the air guitar [Turoque 2006] practice was analyzed as well. It is important to notice that Air Guitar is a consolidated activity, even with championships disputed all over the world (the oldest one is in its 14th edition). The performance of Air Guitar players was observed and evaluated regarding the movements that could be incorporated in the framework. Some of them, such as fingered-ones, are not detected using the color tracking method proposed in this paper; in spite of that, the framework interaction is based on their hand movements. The idea of moving a hand to play the strings and the other one to choose the chords to be played has been focused on this research. Details regarding framework's implementation, example applications and tests performed are detailed further.

3 The Framework

The proposed framework consists of a tool capable of enabling the player to use his/her hands as an "input device" for games and other entertainment applications. More specifically, it focuses on transforming the "air guitar" performance in a way of interaction. The idea is to provide the game developer with the basis for constructing the interaction with the "imaginary guitar", in a way that it is possible to adjust the framework's use according to the needs of the project being developed.

The complete framework is modularized in order to be easily used, altered and updated. The overall architecture is presented in Figure 6. It represents a pipeline of a generic application containing all five major steps involved in the air guitar processing: input image acquisition, image processing, guitar simulation, sound generation and rendering. It is important to notice that the framework makes use of other libraries that provide support for each processing phase. DSVideoLib [Pintaric 2005] was used for capturing webcam's image, FMOD Ex [Technologies 2005] was responsible for playing the output sounds and all visual results were rendered with OpenGL [Graphics 1992]. The entire code is written using the C++ language. Each pipeline step is better described in the next subsections.

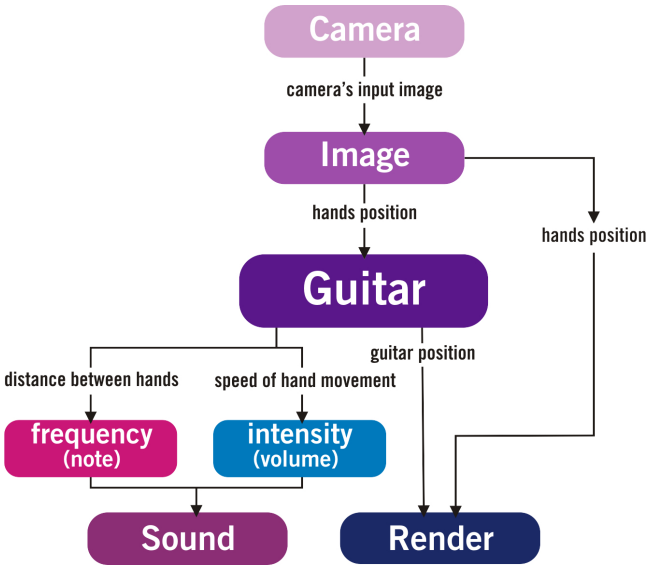


Figure 6: *Framework pipeline architecture.*

3.1 Camera

Tracking the player's hands can be achieved by using many different techniques. Haptic gloves, Wiimotes or model-based hand tracking [Nintendo 2006] are well-known alternatives for performing such task. The proposed framework adopts a color tracking approach as it proved to be the best cost-benefit method to acquire knowledge about the position of user's hands in real time. This type of tracking requires, besides the PC itself, a common webcam and a pair of gloves to be worn by the user, with a color that contrasts well with the surrounding environment.

This method presents many advantages, listed as follows. The first one is the cost of required equipment, which is lower when compared to the price of haptic gloves or even Wiimotes. The processing load demanded by the algorithm is low, which makes possible its real time execution. It is also robust to occlusions, rapid motions and blur, being capable of recovering the tracking immediately after such events.

The Camera module comprises all implementation regarding webcam configuration and image capture. It has functions that provide access to different webcam information, such as the acquisition of data from the last frame (for further rendering). This module communicates with the webcam attached to the computer using the DSVideoLib library.

3.2 Image

Given the current frame, obtained by accessing Camera functions, the color tracking is then initialized. The role of the Image module is to find the two groups of pixels that represent both player's hands (assuming they are already wearing the gloves). This information is used on the estimation of hands' screen position. The processing phase responsible for this task searches for all pixels inside the current frame that are within a pre-defined range of color values and then organizes the encountered groups as described next.

The search starts at the top left pixel and iterates over all subsequent ones. The first step attempts to create color groups based on the searched color along all image. In order to recognize if a pixel remains within the specified color range, the relationship between its RGB components is analyzed. For example, in case the application is looking for yellow gloves, the search takes into account that both Red and Green components must present similar values and that the Blue component value has to be considerably lower than the other ones. This approach works well since the relationship between RGB components is robust to most illumination changes.

A labeling algorithm [Suzuki et al. 2003] is a procedure for assigning a unique label to each object (a group of connected components) in an image. This type of algorithm is used for any subsequent analysis procedure and for distinguishing and referencing labeled objects. Labeling is an indispensable part of nearly all applications in pattern recognition and computer vision. In this work, a specific labeling algorithm was implemented in order to locate the regions corresponding to the selected glove colors. It is described as follows.

Initially, an image map is created in order to store the pixels already visited, and the ones marked with a valid label. For each pixel, a verification is performed to check if its color matched the range of the one searched. In case there is no match, the pixel is marked as visited in the image map and the search continues on the next adjacent pixel. If there is a match, then a pixel group object is created and the search starts using the current pixel as origin. The search considers a 4-connectivity neighborhood scheme (vertical and horizontal - up, down, left and right directions) and takes as radius size **n** a customizable value (the default value is 2). This value guarantees that even if a pixel is separated from the origin pixel by a small distance, according to the search radius, it still can be included in the pixel group.

The labeling algorithm developed adopts a greedy approach, where each successfully neighbor found (a pixel that is within the specified color range) is added to the initial pixel group and a new search is initiated using this pixel as start point. It is important to highlight that the search for pixel groups always marks the visited pixels in the image map and such pixels are not considered on further searches. Since small image resolutions are used (320 x 240 pixels), this guarantees an acceptable processing time for real time applications. When a pixel group search ends (there are no more successful neighbors found), the algorithm continues to iterate across the image, ignoring already marked pixels and creating new pixel groups, until the entire image is visited.

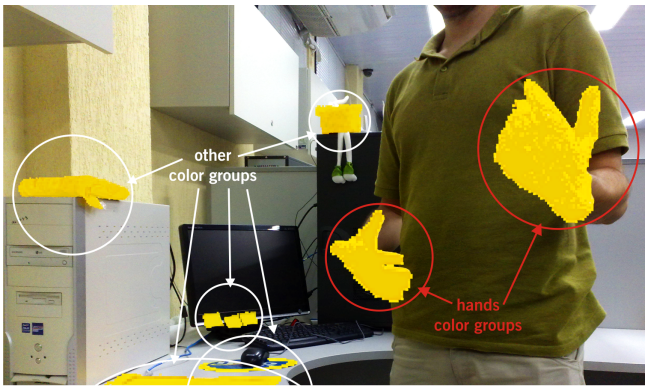


Figure 7: *Color groups composed by yellow sets of pixels. The greater groups almost ever represent the player hands.*

In sequence, after all pixel groups are already constructed, the two with the majority of pixels are selected. In case the size of these two groups is lower than a pre-defined threshold, the processing stops and the current frame is considered as tracking failure. If this is not the case, these two greater groups will represent the hands of the player as shown in Figure 7. This is a guaranteed way of finding the player hands, considering that the gloves' color contrasts clearly with the background environment.

In case there are other elements that match the color requisite, they must be represented by smaller pixel groups, since the hands of the player are located near the camera and in consequence they occupy a larger area on the screen. For simplification reasons, only the center of each hand is considered on further processing. The center of a color group (also known as geometric center, or just centroid) is obtained by summing all pixel coordinates (horizontal and vertical) related to that pixel group and then dividing the result by the amount of pixels of the group.

Once both centers are calculated, the one located on the leftmost part of the screen refers to the left hand and the other the right one. Considering that when playing the guitar a person rarely inverts the

side of his/her hands, this approach presents a good result. Besides this functionality, the Image module provides developers with functions to access the encountered color groups in a way that it is possible for him/her to understand and alter some framework behaviors.

3.3 Guitar

Considering the hand points were obtained, the virtual guitar simulation is initialized. The Guitar module task is to detect player's intentions when performing movements similar to the act of playing a real guitar. The idea is to automatically interpret his/her movements and give as return both visual and audio cues. The guitar model available from the framework corresponds to a simplified version of common electric guitars, since it has less frets and a single string. This simplification is fundamental in order to make possible the player interaction. Playing with more strings would make the interaction impracticable, since the hand movements should be much more precise (as happens with a real guitar). The number of frets is customizable, but it is highly recommendable to use only a few (from 2 to 10 frets) instead of the number corresponding exactly to a real guitar (about 22/23 in most cases), due to the same problem described earlier (precision of interaction).

The guitar simulation process works as described next. The result of this process is the correct placement of a virtual guitar based on the position of the hands. Assuming that the left hand is positioned over guitar's arm (it is possible to change this configuration), the following steps are executed in sequence:

1. The two points in red shown in Figure 8 are used as reference points, found based on the hands locations. According to them, it is possible to make an approximate estimative of guitar's arm and body positions. Both arm and body possess motion speeds, which are defined before the application is running. Such speeds are used to make the guitar follow the user hands, as if the player were holding it. Since the left hand movements follow the guitar axis direction and the right hand movements are perpendicular to this same axis, different motion speeds are applied to the reference points. Because of its perpendicular amplitude of movement, the right hand presents a higher motion speed than the left one.

2. Whenever the current frame represents the first successful frame tracked (i.e. all previous frames were considered tracking failures or this was the first frame captured from the webcam), the reference points will correspond exactly to the hands points. If this is not the case, both guitar's arm and body will follow respectively left and right hand points, as shown in Figure 8.

3. The guitar movement is described by the following algorithm. The position of reference points is updated based on a fixed percentage of the difference vector between reference and hand point. Such percentage corresponds to the motion speed of guitar's arm or body. If the distance found is lower than a customizable jitter threshold, then the reference point keeps its position. This approach guarantees a soft displacement for the guitar's arm and body points, which favors visualization and interaction experience.

4. The player interaction is defined as follows. Whenever he/she crosses the "string line" (the same as the guitar axis), the framework captures the intention of playing. The fact that the guitar's body moves slower than the arm helps the user preventing non-intentional sounds. On the other hand, the speed of guitar's arm does not need to be small.

5. The framework makes use of other few mechanisms to guarantee that players's interaction with the air guitar was correctly captured. The jitter threshold is one of them. Because of it, the string line stops following the player's hands whenever they are too close to the line. This way, the string line will not cross the right hand point by its own. Another important policy is that only high intensity movements represent the intention of playing. These are measured by the distance between last and current right hand points. Finally, a single cross direction is taken into consideration, down or up. In case both were used, the sound would eventually be played even if it was not user's intention.

6. Simultaneously to the capture of player's intention, two values are stored for further use by the Sound module. Such values correspond to the captured intensity and distance between right and left hands at the moment the player crossed the string line.

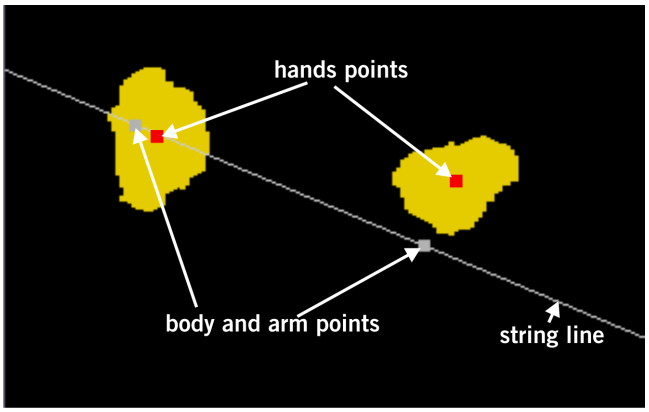


Figure 8: *Guitar points (arm and body) following the hands points. Note that the arm point is much closer to its corresponding string point than the body point. This occurs due to the motion speed of each one.*

3.4 Sound

After gathering both distance and intensity values, as described earlier, the sound processing starts. The module must be adequately configured before running in order to work properly. At first, the number of frets needs to be specified. Based on this information, the virtual guitar arm is divided into different frets that play distinguishable sounds. The closer the fret is to the guitar's body, the higher is the frequency of the corresponding sound.

Every sound generated by the virtual guitar is related to a previously chosen base sample, which is recommended to be a played single note stored in any format supported by FMOD Ex (.wav, .mp3 among others). Such base sample will be played whenever the user plays the guitar, what means that the intensity entry value has surpassed the intensity threshold (pre-defined value of the Sound module). The base sound will be played in a way that its frequency will be modified according to the fret pointed by the user's left hand. In other words, the Sound module takes into consideration the information of the loaded standard frequency audio sample (sound file) and increases its frequency whenever necessary. The first fret of the guitar (the farthest to the guitar's body) will correspond to the base sound, while the next ones will increment the frequency played as described next.

The Sound module uses the twelve-tone equal tempered scale as base to perform frequency increments. It was chosen because of its segmentation method, which is widely used in current music panorama. Almost all fret-divided instruments use it, and it represents the division that most approximates the just intonation (a segmentation which presents a not equal division but has perfect consonant intervals) without adding a lot of additional notes. The difference between the tempered scale and the reference case (the just intonation) is not higher than 1%, as shown in Table 1. The just intonation could also be used, but at the same time it presents perfect intervals, in some combinations it does not perform well. Others equal tempered scales could also be used, since the number of divisions per octave (currently twelve) can be easily altered in real time (e.g., some Indian musicians use 31 divisions).

This way, the initial frequency of the base sample is altered by multiplying its value by $2^{1/12}$ (approximately 1.0595) as many times as the number indicated by the current fret. The base sound is considered the matrix for all other sounds, even in case it is not a well

Table 1: *Relation between the just intonation and the twelve-tone equal tempered scale. Notice that the difference between the correspondent increments is never larger than 1%.*

Increments	Just Intonation	Equal Temp.	Difference
0	$1/1 = 1.000$	$2^{0/12} = 1.000$	0.0%
1	$16/15 = 1.067$	$2^{1/12} = 1.059$	0.7%
2	$9/8 = 1.125$	$2^{2/12} = 1.122$	0.2%
3	$6/5 = 1.200$	$2^{3/12} = 1.189$	0.9%
4	$5/4 = 1.250$	$2^{4/12} = 1.260$	0.8%
5	$4/3 = 1.333$	$2^{5/12} = 1.335$	0.1%
6	$7/5 = 1.400$	$2^{6/12} = 1.414$	1.0%
7	$3/2 = 1.500$	$2^{7/12} = 1.498$	0.1%
8	$8/5 = 1.600$	$2^{8/12} = 1.587$	0.8%
9	$5/3 = 1.667$	$2^{9/12} = 1.682$	0.9%
10	$9/5 = 1.800$	$2^{10/12} = 1.782$	1.0%
11	$15/8 = 1.875$	$2^{11/12} = 1.888$	0.7%
12	$2/1 = 2.000$	$2^{12/12} = 2.000$	0.0%

known note, only preserving the relative relationship between them; i.e., the absolute tuning considering A4 as 440Hz is not taken into account, since the objective is to guarantee that different frets perform coherent sounds, differing from each other.

Some improvements can be adopted in order to make the interaction more attractive. To avoid undesired notes, it is possible to exclude some sounds, thus limiting the playable tone notes set to a few ones. Any of the twelve sounds and their corresponding doubled frequencies (octaves) can be activated/deactivated at any time. Consequently, in case the player wants to play a relative pentatonic scale with the base sound as matrix, he/she just needs to turn off the undesired notes as shown in Figure 9. The interface from the demo application developed for testing the framework shows at the top all enabled notes, from 1 to 12. In this case, for enabling/disabling any of them, the user should press any of the F_n keyboard keys, with n varying from 1 to 12. This feature also makes it possible to restrict the notes to a set of sounds necessary to play a specific song, as detailed later on the Case Study section. When a note is deactivated, the fret that would play its sound now plays the next activated one. Then all frets remain valid; what changes is the notes that are played.

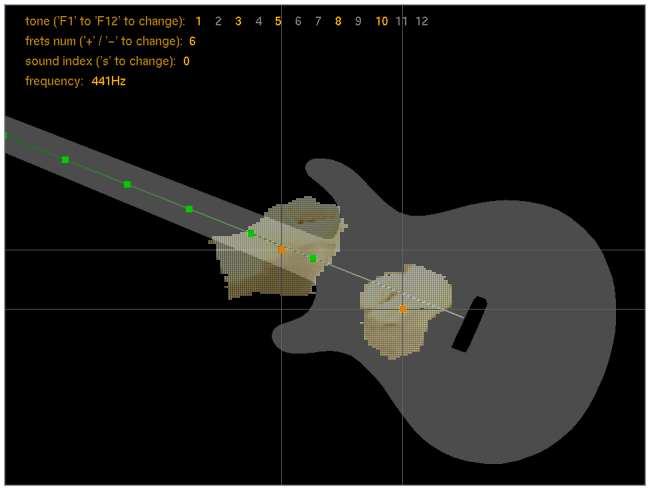


Figure 9: *Demo application interface. At the top of the image it is possible to see that the tone is limited to a pentatonic scale using the base sound as the first note. The current number of frets is six and can be set at any time during the playing. In the guitar, the frets division is represented by the green squares. Finally, the sound index represents the current sound archive that is being used as base sound and can also be dynamically altered.*

Another implemented feature corresponds to the volume of the played sound according to the intensity of the input. The faster the right hand movement is, the higher will be the sound played. The envelope of the played sound (which determines the volume of the sound during a specific time) can also be altered by providing an increased sustain effect to the played note. A fret-based slide was also implemented, which makes the sound to change even in case the player does not move the right hand, by only sliding the left one after a played note, in a similar way to the slide technique on real guitars. Finally, more sound channels can be added in order to create more complex sound combinations, thus enhancing user's experience by simulating chords, for example. The Sound module, together with FMOD Ex, provides functions capable of managing all these parameters at any time, on the fly.

3.5 Render

This module is responsible for returning visual information to the player, helping him/her to achieve the application objectives. Although it is entirely possible to play without visual cues, interaction becomes more interesting with the viewing of the virtual guitar (and its frets divisions) according to the gloves' positions. Besides that, it can show other information such as text indications for activated/deactivated notes, current played frequency, among others, as shown in Figure 9.

The Render module also provides developers with some debugging functions, like rendering guitar points (both arm and body). This is useful for the application's test phase, since debugging is one of developers' most time consuming activities. This module uses OpenGL (and GLUT [Graphics 1996]) to perform such functions. If necessary, the render engine can be changed by constructing another render module (using OGRE3D [Team 2005], for example), similar to the one provided by the framework, without modifying the other framework modules.

4 Case Study

In this section will be described the case study, it is a simple example application using the framework. The user interface was a similar version of the interface shown in Figure 9. The framework evaluation process is described next.

4.1 Evaluation Methodology

In this work, the proposed framework was evaluated using a demo implementation of a simplified air guitar game. The tests were performed on an AMD Athlon X2 4800+ CPU, 1 GB of RAM equipped with an NVidia 8800 GTX GPU. The time necessary in order to correctly play a pre-defined sequence of notes was recorded. To assure the results were consistent, it was tried to expand the tested subjects to the highest number possible.

A total of 23 people were subject to the tests performed, varying on their age (from 17 to 47 years old) and gaming experience. After getting used to the demo application (playing freely during some minutes), they were asked to play a version of the introduction of the Smoke on the Water song (by Deep Purple), which is mainly composed by 4 different notes, as shown in Figure 10.

The virtual guitar was previously configured to allow this specific sequence to be performed. An electric guitar A3 played note with some distortion was used as base sound, being the first note of the introduction (differently from the B2 based power chord of the original version of the song). Only four different sounds were enabled, being the base sound (sound number 1) and the sounds 4, 6 and 7. The number of frets were set to four, each one representing one of the enabled notes, by this way, facilitating the players task.

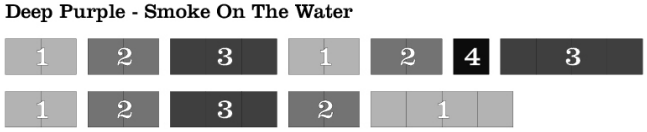


Figure 10: *Initial sequence from the "Smoke on the Water" song.*

The test environment was organized as follows. An A4Tech webcam was placed at 1.30 meters above ground and about 1.5 meters in front of the user. Due to the camera characteristics, this placement showed to be a good choice for comfortably capturing the user's hands, allowing a good amplitude of movements. The demo visualization was projected on the wall, about 2.5 meters away from the spot where the subjects should stand. This should improve user's view and his/her immersion in the demo.

Preparation: the subjects put on the yellow rubber gloves and took place standing in front of the webcam. After that, they were told how to correctly position body and hands in order to enable the viewing of the entire virtual guitar, as well as their hands, on screen, as shown in Figure 1.

Familiarity with the application: before performing the test itself, the subjects had up to 3 minutes to understand the interface and check their performance by playing random notes.

Test: after understanding how the virtual guitar works, the subjects were presented to the sequence of notes that should be played (shown in Figure 10). They were told about the correspondence between the numbers written on the paper and the notes on the guitar, which varies according to the left hand's position. The total amount of time considered was the one spent performing correctly the entire sequence, from the moment the subject played the first note until the last correct one. In case of making some mistakes while playing the notes, the subject was told to restart the sequence from the beginning, without resetting or stopping the time count.

After performing the test, the subjects were oriented to answer a small free-form questionnaire about their experience with the application. The subjects were asked about difficulty in hands positioning, the application's visual and audio feedback and their impressions about the experience.

4.2 Results

The tests with the virtual guitar framework demo showed that the application is functional, intuitive and interesting. All 23 subjects submitted to the test were capable of performing the musical sequence correctly, without any problems. The mean execution time was 33.6 seconds, varying from 11 to 80 seconds.

Figure 11 shows the age of subjects and the time spent doing the test, indicating more density between the 20-30 year-old range, which is a potential target for a product using the framework technology. Most of the subjects in that age range spent between 11 and 41 seconds. This time was considered very satisfactory, since these people had never interacted with the application before.

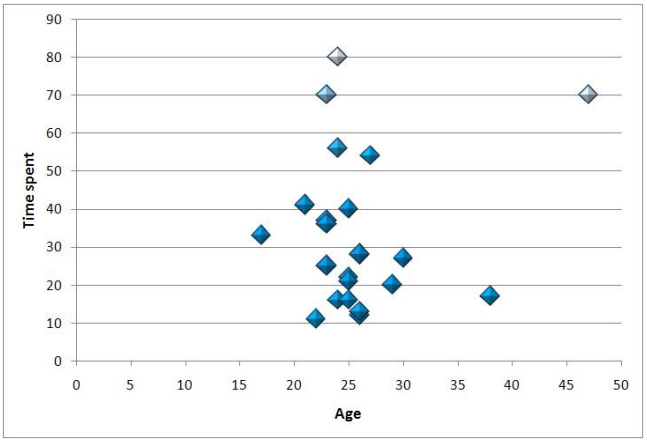


Figure 11: *Time spent × Age results.*

Asked about the guitar feedback according to their hand interaction, 100% of the subjects said not to have felt any delay or misinterpretation of their movements. That real time feedback capacity is one of the framework's main attributes, due to the fact that any delay, even the smallest one, would compromise the usability of the virtual guitar, making hard for the user to move his/her hands spontaneously and intuitively.

Some of the subjects found difficulty while positioning their hands, in part due to the complete unknowing of the interface, in part due to the position of the webcam, which was fixed at the same height in all tests. Variations in height and scale of the subjects were not considered, and as a consequence some of them had to position their hands too high, too low, too open or too closed. However, most people managed to overcome this initial problem by finding an adequate way to position their hands in order to play the music sequence.

All tested subjects provided positive results regarding their experience. They seemed excited about the possibility of applying the framework capabilities into a real scenario.

Many of the answers suggested developing a game based on the demo application. Besides that, they pointed some improvements and adaptations, such as: adding more strings to the virtual guitar, in order to get a more realistically simulation; applying the technology to other instruments, like bass and drums; possibility of changing the guitar settings using the gloves themselves instead of the computer keyboard; implementation of other types of user interaction, like slide (by moving the left hand continuously along the frets, the sound would change without moving the right hand).

4.3 Lessons Learned

The tests described before were performed using a simplified set of the framework features. However, it proved to be possible to play entire songs, using all the frets and notes possibilities. The framework has several use possibilities, such as game development (as shown on the right of Figure 1), or an educational artifact, which could teach musical concepts, like notes, tones and rhythm to the users.

Due to its features, it was possible to notice that the framework is capable of providing both developers and users with all three requirements previously listed: real time interaction, actions mimicking a real guitar control and a realistic sound response. The satisfaction of tested subjects showed that performing a known type of interaction increases the application's playability. It is possible to have a simple but functional application only by using the framework, while complex functionalities could be added taking as base the demonstrated example application.

Taking a game as an application example, some features could be added to make it more interesting, such as: player's possibility to choose between several guitar models to play (as shown in Figure 12); monetary rewards system, in exchange for the correct execution of the songs; possibility to play new songs, progressively unlocked during the game; navigation through the game menus using only the hands wearing gloves, instead of a mouse or keyboard, using the same color-tracking technology applied in the guitar framework; support to guitar effects such as vibrato and slide (as shown in Figure 13 and Figure 14); more attractive design to the game interface, showing information of score and performance of the player (besides the sequence to be played, of course).

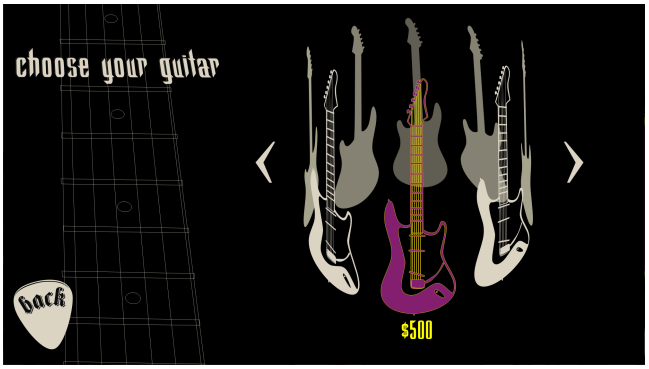


Figure 12: *"Choose your guitar" screen.*

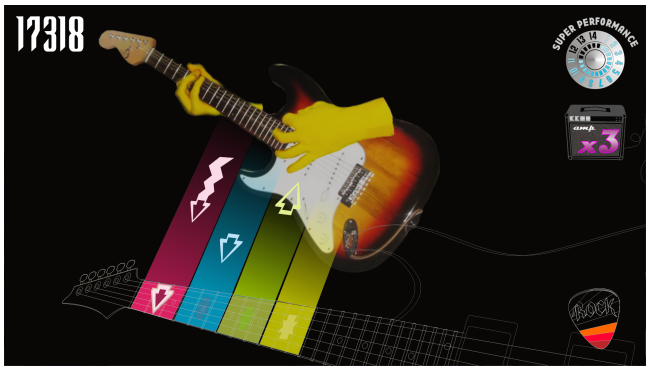


Figure 13: *Gameplay screen showing a "vibrato" indication.*

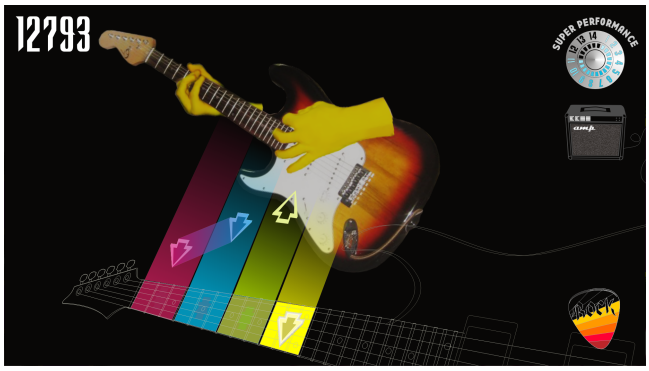


Figure 14: *Gameplay screen showing a "slide" indication.*

5 Conclusion

In this work, an open-source framework for air guitar based applications has been detailed, showing that with a pair of colored gloves and a webcam it is possible to recognize user's intention of playing a virtual guitar. It was also described a case study, where an example application was used to validate the framework. This application was tested and the results showed to be successful: all tested subjects could reach the objective of playing a simple song during a small amount of time and the most important, they were satisfied with the experience.

As future work, a number of new features can be added to the framework. Some effects such as guitar bend and vibrato simulation can be implemented to make the player interaction more interesting. A multi-player mode should also be possibly present in the next versions, with the only restriction that the players must use gloves with different colors.

The air guitar framework is available for download at http://sourceforge.net/projects/airguitarframew.

References

BYRNE, M., 2004. Console players win on their points. http://www.fyrne.com/Article_pages/Console_players_win.html.

CAMURRI, A., RICCHETTI, M., AND TROCCA, R. 1999. Eyesweb - toward gesture and affect recognition in dance/music interactive systems. In *ICMCS '99: Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, IEEE Computer Society, Washington, DC, USA, 9643.

ENTERTAINMENT, S. C., 2007. Ps3 eyetoy. http://en.wikipedia.org/wiki/PlayStation_Eye, October.

EUROPE, X. H. I., 2009. Dj hero coming this year. http://xbox.hdtvinfo.eu/component/content/article/21-dj-hero-coming-this-year.html, March.

GAMES, A., 2007. Adagio. http://www.newgrounds.com/portal/view/388085.

GRAPHICS, S., 1992. Open graphics library (opengl). http://www.opengl.org/.

GRAPHICS, S., 1996. The opengl utility toolkit (glut). http://www.opengl.org/resources/libraries/glut/.

KARJALAINEN, MATTI; MKI-PATOLA, T. K. A. H. A. 2006. Experiments with virtual reality instruments. In *JAES Volume 54*, Audio Engineering Society, San Francisco, USA, 964–980.

KONAMI, K. D. E., 1998. Guitarfreaks. http://en.wikipedia.org/wiki/GuitarFreaks.

MÄKI-PATOLA, T., LAITINEN, J., KANERVA, A., AND TAKALA, T. 2005. Experiments with virtual reality instruments. In *NIME '05: Proceedings of the 2005 conference on New interfaces for musical expression*, National University of Singapore, Singapore, Singapore, 11–16.

MICROSOFT, 2009. Xbox360 project natal. http://www.xbox.com/en-US/live/projectnatal/.

MTV GAMES, E. A., 2007. Rock band. http://www.rockband.com/, November.

NINTENDO, 2006. Wii remote controller. http://en.wikipedia.org/wiki/Wii_Remote, November.

PAKARINEN, J., PUPUTTI, T., AND VÄLIMÄKI, V. 2008. Virtual slide guitar. *Comput. Music J. 32*, 3, 42–54.

PINTARIC, T., 2005. Dsvideolib. http://www.ims.tuwien.ac.at/ thomas/dsvideolib.php, October.

REDOCTANE, A., 2005. Guitar hero. http://hub.guitarhero.com/index_uk.html, November.

SUZUKI, K., HORIBA, I., AND SUGIE, N. 2003. Linear-time connected-component labeling based on sequential local operations. *Comput. Vis. Image Underst. 89*, 1, 1–23.

SYSTEMS, H. M., 2001. Frequency. http://www.harmonixmusic.com/#games, November.

TEAM, T. O., 2005. Object-oriented graphics rendering engine (ogre3d). http://www.ogre3d.org/, March.

TECHNOLOGIES, F., 2005. Fmod. http://www.fmod.org/, December.

TUROQUE, B. 2006. *To Air is Human: One Man's Quest to Become the World's Greatest Air Guitarist*. Riverhead Trade.

VOODOO, U., 2006. Frets on fire. http://fretsonfire.sourceforge.net/, August.

WIXON, D. 2007. Guitar hero: the inspirational story of an "overnight" success. *interactions 14*, 3, 16–17.

Automatic Sprite Shading

Djalma Bandeira and Marcelo Walter
Centro de Informática - UFPE, Brazil
{dbs, marcelow}@cin.ufpe.br

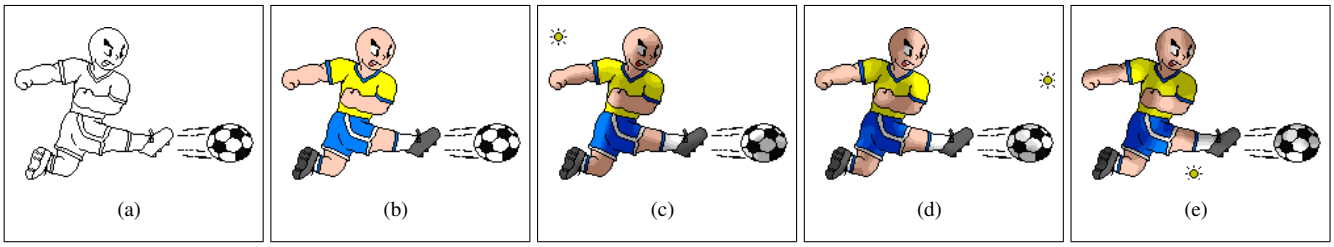


Figure 1: *From the outlines (a) we consider the basic colors (b) that are taken as input to generate different shading distributions (c, d, e) controlled by a local light source.*

Abstract

Sprites have been present since the first arcade games and console generations. Despite the advances in computer graphics and the whole representation of virtual worlds on 3D environments, 2D-based games still have their market, specially on portable consoles and mobile devices. The visual quality of today's games have increased as a result of hardware improvements in processing power, memory, and a richer color gamut. However, most of the editing task is still manual, using graphics editing tools. We present a method to automatically generate shading distribution on sprites, one important step during the editing process that is time demanding for most manual works on art design today. Our method allows to control the position of a local light source and to generate an approximation of the shading distribution effect. Although our solution is not physically accurate, according to the shape of the represented object, it can produce well usable results for 2D game systems in many practical cases, when compared with handmade sprite shading.

Keywords:: Real-time processing, art design, programming techniques, computer animation

1 Introduction

Two-dimensional games have a large impact on the entertainment industry today. Even with the evolution of 3D graphics, these games still assume an important role on the market. This is motivated for several reasons, from availability and low cost of mobile devices, to a requirement of simple but "catchy" graphics on games for kids. Besides, the development of native 2D games is generally less complicated than 3D games, specially on art related stages. The production is also widely explored by beginner game developers, for learning and application of basic 2D fundamentals.

Inside the game development process there is the art design stage, which is generally time consuming. For many projects, a team of game artists share the task of designing the visual identity for a game. In this stage, there are steps of graphics production and editing. This involves, among other things, art and effects on raster graphics, texture manipulation and, as the focus of our work, sprite editing.

The process of sprite generation and editing in game development is still a manual artistic composition for most cases. In fact, most of the literature describes the process as a time demanding procedure. It is generally presented as tutorials on websites and pixel art communities [Yu 2005; Tsugumo 2001; Sedgeman et al. 2004], although some basic concepts can be found on references such as [Feldman 2001]. Here we address the step of sprite editing related to the manipulation of shading effects, used to increase details on 2D images and make them look more like a 3D object, lit by a local light source.

Most of the solutions used nowadays rely on manual graphics editing on the image considering one or more fixed light sources. Alternatively, some solutions build a 3D coarse representation of the shape to better estimate and generate shading details, and then synthesize and express this information as a 2D image. Both methods have drawbacks when a new shading configuration is desired, requiring more image editing or another retrieval of shading information for the 3D representation in order to achieve the expected results.

We propose here an automatic solution to estimate and generate shading distributions on sprites which allows to control the position of a light source and experiment the results in real-time. Our approach, although straightforward, generates a visually consistent approximation of shading effects considering the light source. The results presented show good visual quality when applied on basic 2D sprites, which by nature do not require a highly precise shading distribution. Figure 1 shows the use of our method to automatically synthesize shading distributions controlled by the position of a local light source.

2 Related Work

To the best of our knowledge, there are no references involving shading manipulation directly related to sprites. However, more general approaches were presented, particularly focused on animations. One relevant work is [Johnston 2002] that presents a method to approximate and control the lighting distribution for 2D cel animation to aid the composition of cartoon live-action scenes. He uses a multi-channel image information to correct and obtain what would be a good approximation of normals, generating very convincing results for the final composed shading. This inspired the later work of Bezerra and colleagues [Bezerra et al. 2005], where they present a pipeline for 2D animation that also uses estimated normals from an outlined image to approximate and generate shading details, in addition with some simplifications from the previous work. According to the authors, the technique they proposed was the only method genuinely image-based so far and is easily applied to 2D cel animation.

Another work that deserves attention is [Anjyo et al. 2006]. They proposed a method to manipulate stylized light and shading animations in real-time, allowing the control of properties such as scaling, rotation, translation and splitting on shaded areas. Although flexible, their method still needs additional capabilities on tasks like making shaded areas as stylized highlights more editable. The work in [Todo et al. 2007] follows the same concept, presenting a set of algorithms to manipulate local shading distribution for a better consistence and enhancement on the final results. Although it is possible to obtain very good shading approximations with these methods, both approaches work based on 3D domain applications.

These related works present techniques on shading construction relying on 3D information or normal reconstruction to work, and while some may be suitable to be applied on sprites as well, they

generally demand several adjustments and/or layer constraints and manipulation for a good result. Our technique, on the other hand, provides a visual consistent shading approximation, exchanging the not so required physical accuracy on sprite images for a more flexible and fast method aiming both artists and beginners users. A good argument for our method is that what mainly differs a sprite shading from conventional shading effects on animations, is the fact that the former case generally considers an arbitrary light source not related to the external environment.

3 Method

The method we propose is divided into the following four stages, explained below:

1. Image Segmentation
2. Highlight Spots Approximation
3. Shading Distribution
4. Final Composition

3.1 Image Segmentation

We consider initially as input a sprite image containing only basic colors. In this first stage, we segment the image and separate the pixels that will be processed later. We first mark the background pixels with an uniform color as invalid pixels. The outline pixels themselves may or may not be also set as invalid pixels. This particular condition for outlines can alter the result drastically, as we use the key idea that the outline pixels separate distinct regions of the sprite and therefore determine basic shape and sections of the object being represented. This affects directly the shading distribution of separated small regions instead of considering the whole image as a single object without interior outlines. The final shading result will depend on the outlines distribution. On the other hand, ignoring the interior outlines can be useful in cases where a global shading distribution is desired.

The result of the image segmentation can be seen as a mask, where all the other colors are interpreted as valid pixels and represent the same region to be processed. Figure 2 below illustrates how the presence of outlines affects the mask and the final result:

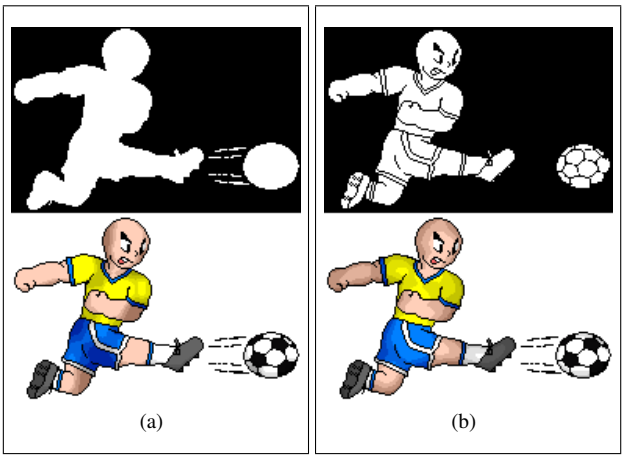


Figure 2: *Example of mask and the respective shading without (a) and with (b) outline constraint.*

3.2 Highlight Spots Approximation

Once the image is properly segmented, we now have to search for possible highlight spots. These spots are the regions of the image with the maximum exposure to the light source. First, we make image sections in one of the two axis directions (x or y) by tracking possible segments following the opposite axis. Considering initially the y-direction sectioning, we scan the entire image for x-segments as continuous lines of valid pixels. The segment ends when an invalid pixel is found or the bound of the sprite image has been reached. This may lead to several segments per line on the image. Since our method allows to control the position of the light source, we define the parameter l_p as the local position of the light inside the range $[0.0, 1.0]$, by taking the $[0.0, 0.0]$ position as the top-left corner of the coordinate system. Suppose that the end of a certain segment was found and p_0 and p_1 are the start and ending pixels positions, we calculate the center pixel p_c using a simple linear interpolation:

$$p_c = p_0 + l_p(p_1 - p_0)$$

Since l_p controls the light position, it consequently adjusts the center pixel on both directions to a position that "follows" the light source. We repeat this procedure for every single line of the sprite image and once all center pixels were computed, we have a sectioning on the y-direction. We restart this process by following now the x-direction looking for y-segments. Figure 3 shows each separated sections and the combination of both.

The next step is related to the shading weight of pixels. Now that we computed the sections, we classify every pixel in three levels of shading weight: weight 2 for those pixels inside the intersection of both X and Y segments, weight 1 for those pixels that belong to only one section and weight 0 for those out of any section. These weights determine the exposition of every pixel from the current light source, considering that those with weight 2 are the center of a highlight spot.

3.3 Shading Distribution

After the shading weight is properly assigned for every pixel, we now calculate the average shading distribution. We do this for all pixels by averaging the weight of every pixel inside the neighborhood of a certain regular window of size S_w. The window size is a parameter that varies according to the size of the sprite and the desired shading diffusion of a pixel among its neighbors. We consider setting the size $S_w = 7$ as a good value for examples around 100 pixels of resolution. Once the shading distribution in the whole image is processed, we blur it to reduce some sharpness that may affect the final result. We do this by averaging them again on a neighborhood window of size S_b that may be different in size from S_w. We set $S_b = 3$ for the case previously discussed, redefining both windows sizes to vary according to the sprite image dimensions. We can adjust both window sizes S_d and S_w independently, to better fit the resolution and desired shading effect. Figure 4 shows how we can improve the computed 3D look of two basic forms by adjusting the window size parameters.

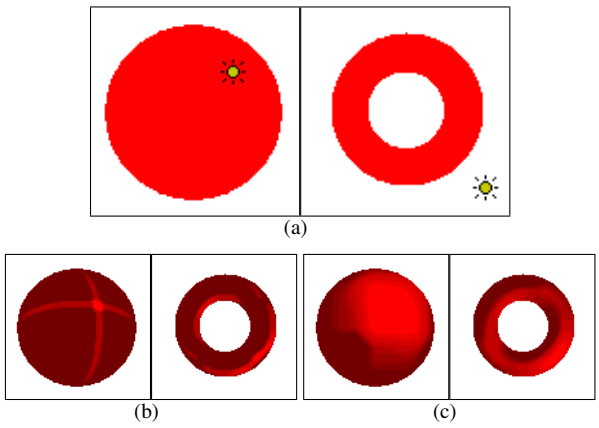


Figure 4: *3D shape approximation on shading of the input (a) can be improved from (b) to (c) by resetting the window values $S_w = 7$ and $S_b = 3$ accordingly. The parameters were reset to $S_w = 49$ and $S_b = 15$ for a sphere shape, and $S_w = 21$ and $S_b = 9$ for a torus shape. The yellow dot in (a) is the light source position.*

Once all average weights are computed, we reset all the values to the range $[0.0, 1.0]$ by considering the minimum and maximum average shading distribution found on the previous step to finally have

(a) (b) (c)

Figure 3: *Sectioning on the y-direction (a), x-direction (b) and the combination of both directions (c) with the highlight centers marked as red pixels. The green pixels are the center pixels for each segment.*

the estimated shading distribution. In Figure 5 we show some shading distributions considering the light source in different positions. Notice how the lighter and darker regions change according to the light position.

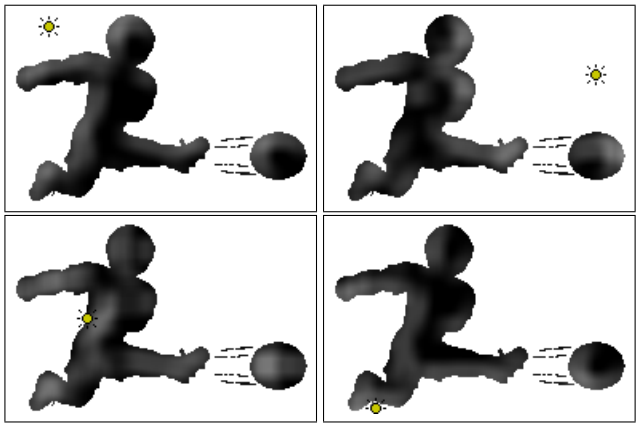


Figure 5: *Variety of shading distributions based on the local light source position (seen as a yellow dot).*

3.4 Final Composition

This last stage consists of combining the shading distribution previously computed to obtain different levels of shaded colors from the basic ones. We call these variations **color shades**, whose number per basic color is defined as the number of shades. To allow variable compositions, the color shades may be controlled by two parameters: **shading offset** and **shading shift**.

3.4.1 Shading Offset

The shading offset controls the distribution of color shades for every basic color inside the range $[0.0, 1.0]$ of the shading distribution. This parameter may be set evenly for every color shade considering the number of shades, or may be adjusted to change the balance distribution among them. Figure 6 illustrates different shading offsets for the basic skin color as well as the result on the entire sprite.

3.4.2 Shading Shift

The second parameter, shading shift (S_s), will control to which direction the variation of colors will assume for the current color shade. Let us consider a simple case of four shades for the previous sprite. The trivial case of $S_s = 0$ will treat the first color shade as the basic color of the sprite image among the possible colors. Lower values of shading distribution with this set leads to darker versions of the basic color. Now, if the shading shift is set to 1, as can be seen on Figure 7, the basic colors will be "shifted" to the right on the palette of possible colors. The same will occur for negative values, shifting to the left of the palette instead.

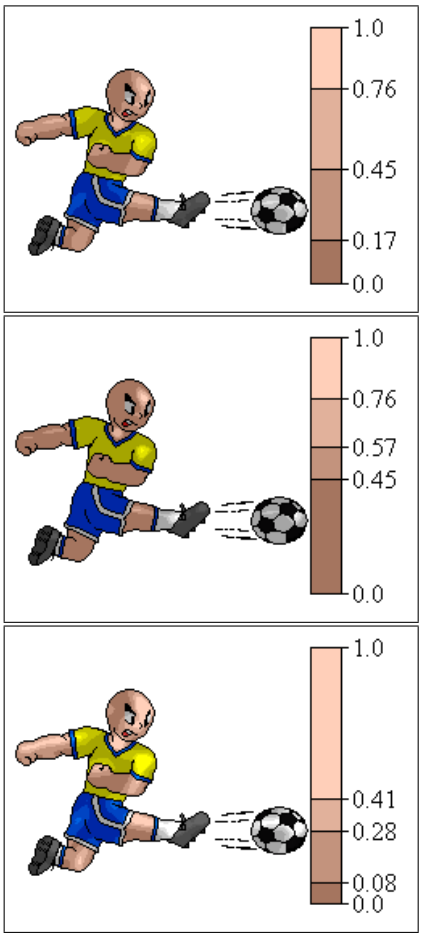


Figure 6: *Different shading distributions adjusted by the offset.*

3.4.3 Shading Composition

The two parameters discussed above will allow a wide range of shading variations. To calculate new colors from the basic color, we consider the shading shift S_s and a delta shading parameter ΔS. The delta shading is the percent of the basic color that separates different color shades. So, for every increase in the color shade, we have the corresponding color as a subtraction of the basic color.

Since we have the shading offset for each color shade, we can find from which shade S every pixel belongs to, by evaluating the value of its shading distribution. We then set a γ parameter as the value of the highest RGB component for the basic color. Now considering c as the color component and the previously discussed parameters, the final color $c\prime$ can be defined as:

$$c\prime = c - (\gamma \Delta S)(S - S_s)$$

Notice that the shading shift S_s adjusts the overall color variation by controlling positive or negative subtraction of the basic color

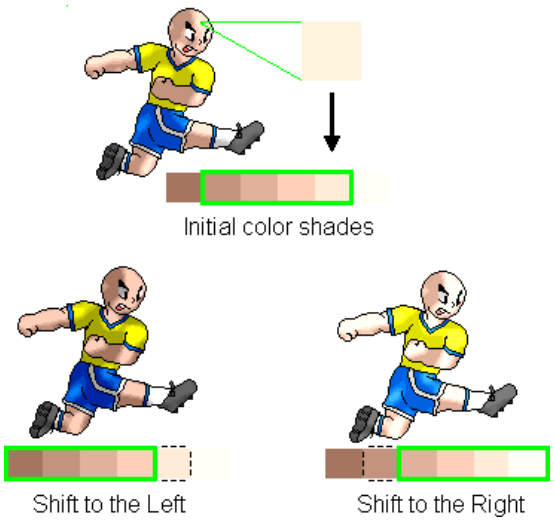


Figure 7: *One of the color palettes (skin color) with selected shades inside the green rectangle. The shift adjusts a new set of color shades on the palette.*

from the difference of the shade S. With the final color computed for every color component, the last operation is to clamp values outside the range $[0, 255]$ of possible colors on the RGB color space.

3.4.4 Extended Applications

After we have computed the shading information, other shading-based effects can be provided. One example is the highlight specular effect. Assuming that the shading distribution s of a certain pixel is higher than a threshold ω, we can obtain the new color c_h inside the specular region from the previous color $c\prime$ and the highlight level $\beta((s-\omega)/(1.0-\omega))$ as:

$$c_h = (1.0 - \beta)c\prime + \beta s_c$$

In this equation, β is the highlight level for the shading considering the distance from the threshold to the maximum value allowed 1.0. Then, we obtain the new color by balancing this highlight level between the previous color shade $c\prime$ and a specular color s_c. This is a simple approach we propose to control the effect with just two parameter tweaks, but other ideas can be elaborated and applied as well. Figure 8 shows and example using this method.

Figure 8: *Our application extended to handle specular highlights. The glass goblet sprite (a) without (b) and with (c) highlights. The parameter values used were $\omega = 0.4$ and $s_c = 0.9$ for all color components.*

Another extension for application using shading information is to generate dithering effects, instead of uniform color shades. We used the Floyd-Steinberg algorithm [Floyd and Steinberg 1977] to generate an alternative shading with less color shades, such as presented in Figure 9. The dithering effect as a shading representation with less color shades can be used for an alternative shading style or a palette simplification solution duo to hardware/software limitations.

Figure 9: *Sprite shading without (a) and with (b) dithering effect.*

4 Results

Most results presented in this paper were computed considering sprite images around 100 pixels of resolution. With minimum adjustments, we can handle specular highlights by considering that pixels whose shading distribution is over a certain threshold will assume a specular color. Figure 8 shows and example of this application on a glass goblet sprite.

Figure 10 shows a comparison between a manual shading effect and the automatic shading from our method. Notice that, even with the differences on shading distribution, our result is still visually compatible with a manual shading work and good enough for practical use. The computed result can later be refined by user manipulation. The manual artistic work may take a few hours whereas our solution is real-time, and besides, allows arbitrary light positions.

One possible drawback of an automatic solution is the shading coherence on sprite animation. Our approach, however, preserves this coherence, allowing batch processing without compromising the final work. Figure 11 shows a few frames of an animation loop after applying the automatic shading from the same light source.

Finally, in Figure 12 we show the shading processing with different parameter sets for shading shift and light source position to simulate alternative local light conditions.

5 Conclusion

We proposed an approach to estimate shading distributions on sprite images, considering adjustable parameters such as position of the light source and style related parameters involving variation of color shades. Although simple and fast, our method can be applied on a wide variety of sprites with a very flexible control of shading details at interactive rates. This ensures a better user experimentation and a faster achievement of desired results that on traditional techniques. The shading information generated allows the implementation of effects that extend beyond those presented here. Like [Bezerra et al. 2005], our method is genuinely 2D and the only one focused on sprite images. Since it does not require any 3D information, it is easy to implement and so highly recommended for beginner game developers to learn and test on their applications. Our technique is more recommended for low resolution sprite images, although its application on large sprites or even entire scenes may generate good shading results depending on the color distribution and/or outline details. Figure 13 shows how the processed shading can be evaluated as a normal distribution computed with a center light source and image gradient calculation from the shading. This is just to establish a comparison between the shading on the formal procedure and the use of recovered normals from the same shading.

We used the Phong illumination model [Phong 1975] for the normals with sphere mapping coordinates.

For future work, we will consider possible adaptations of the method for vector graphics, since games with vector-based images have drawn a lot of attention in the last years. We will also focus on improvements to approximate the shading coherence to the object shape.

Figure 10: *A basic color sprite (a) and the result of a manual shading work (b) and our method (c).*

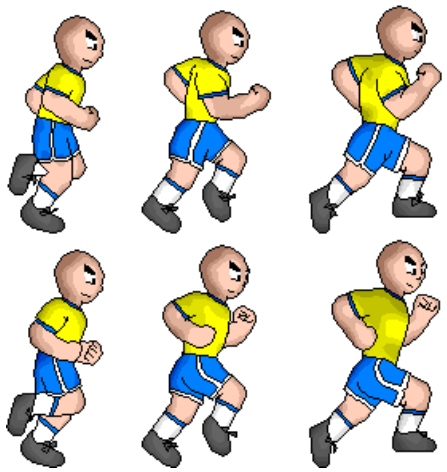


Figure 11: *Frame sequence of an animation loop. The shading coherence on the resulting animation is well preserved.*

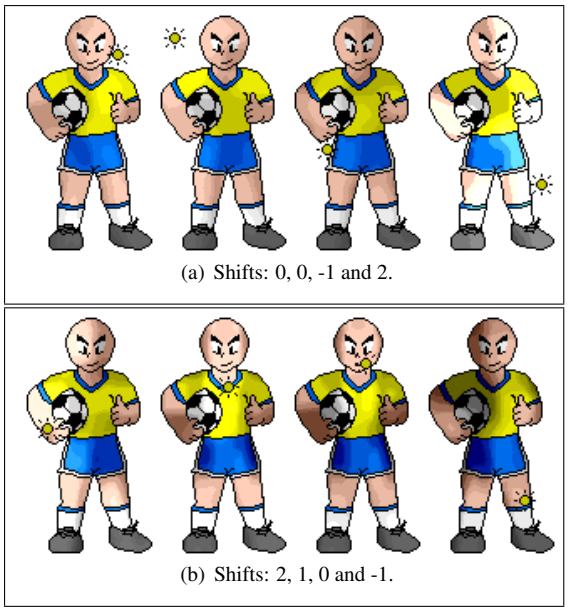


Figure 12: *Applications using 4 (a) and 8 (b) shades with variations on local light source position and shading shift.*

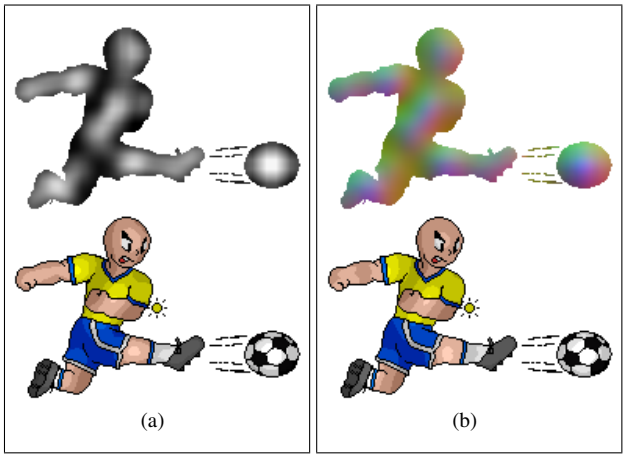


Figure 13: *An evaluation of normal approximation. The shading on a standard procedure (a) and using estimated normals (b) obtained from the same shading.*

Acknowledgements

We would like to thank Meantime Mobile Creations for helping us test the method by sending some state of the art game sprites (Figure 10). Work partially supported by FACEPE through grants IBPG-0216-1.03/08 and APQ-0203-1.03/06 and CNPq through grant 483356/2007.

References

ANJYO, K., WEMLER, S., AND BAXTER, W. 2006. Tweakable light and shade for cartoon animation. In *NPAR '06: Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, 133–139.

BEZERRA, H., FEIJO, B., AND VELHO, L. 2005. An image-based shading pipeline for 2d animation. In *SIBGRAPI '05: Proceedings of the XVIII Brazilian Symposium on Computer Graphics and Image Processing*, IEEE Computer Society, Washington, DC, USA, 307.

FELDMAN, A. 2001. *Designing Arcade Computer Game Graphics*. Wordware Publishing, Inc.

FLOYD, R., AND STEINBERG, L. 1977. An adaptive algorithm for spatial grey scale. In *Proceedings of the Society for Information Display*, 75–77.

JOHNSTON, S. F. 2002. Lumo: illumination for cel animation. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, 45–ff.

PHONG, B. T. 1975. Illumination for computer generated pictures. *Commun. ACM 18*, 6, 311–317.

SEDGEMAN, L., DAVIES, K., DIXON, I., VAN BRUGGEN, B., PLEIZIER, W., AND LEE, O., 2004. Pixel joint. `http://www.pixeljoint.com`.

TODO, H., ANJYO, K.-I., BAXTER, W., AND IGARASHI, T. 2007. Locally controllable stylized shading. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, New York, NY, USA, 17.

TSUGUMO, 2001. So you want to be a pixel artist? `http://www.petesqbsite.com/sections/tutorials/tuts/tsugumo/default.htm`.

YU, D., 2005. Derek Yu's Website. `http://www.derekyu.com`.

Charack: tool for real-time generation of pseudo-infinite virtual worlds for 3D games

Fernando Bevilacqua
Cesar Tadeu Pozzer
Marcos Cordeiro d Ornellas
UFSM, Programa de Pós-graduação em Informática, Brazil

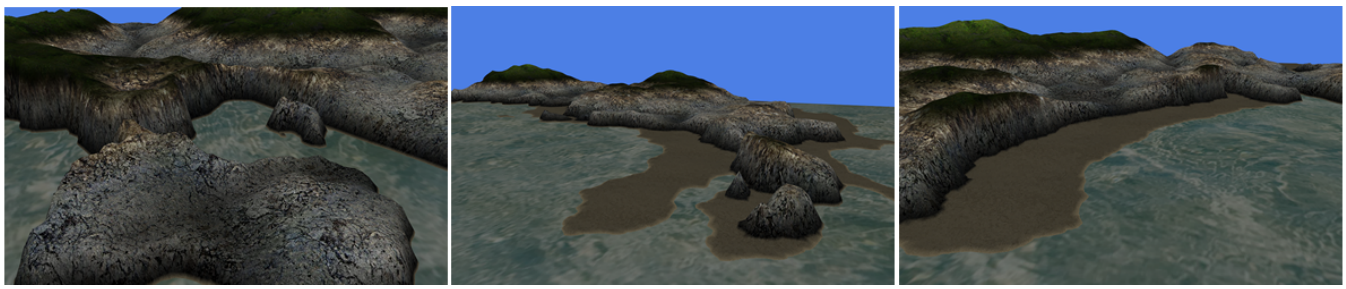


Figure 1: *Coastlines and islands procedurally generated by Charack*

Abstract

In MMO games the player's experience is mainly influenced by the size and details of the virtual world. Technically the bigger the world is, the bigger is the time the player takes to explore all the places. This work presents a tool (named Charack) able to generate pseudo-infinite virtual worlds with different types of terrains. Using a combination of algorithms and content management methods, Charack is able to create beaches, islands, bays and coastlines that imitates real world landscapes. The tool clearly distinguish the generation of each type of content. The contribution of the tool is the ability to generate arbitrarily large pieces of land (or landscape) focusing on detailed coastline generation, by means of using procedural algorithms.

Keywords:: MMO, virtual worlds, terrain generation, 3D games, noise, procedural generation, multifractal

Author's Contact:

{fernando,pozzer,ornellas}@inf.ufsm.br

1 Introduction

The computer games market has been evolving considerably over the years. Since the first console, the hardware performance has increased and new graphic technologies were developed, resulting in a wide range of themes and game styles. In the multiplayer games, players interact with other human beings and also with NPCs, which are represented by virtual characters. That kind of game is popular and the social interaction between players is a matter of research [Griffiths et al. 2003; Ducheneaut et al. 2006]. In the category of multiplayer games there are the massively multiplayer on-line (MMO) ones, which are on-line games featuring large number of players interacting with each other in a huge virtual world.

An MMO can feature millions of players, such as EverQuest [Sony Entertainment 2007] and World of Warcraft [Blizzard Entertainment 2007], the latter with more than 6 million subscribers [Clark 2006]. A persistent virtual world is an important topic to keep the game fun and attractive to the player. The bigger is the world to be explored, technically the bigger is the time the player has to spend in order to explore all the places. As a result of such huge virtual worlds, their creation and subsequent upgrade are a complex task. EverQuest and World of Warcraft present a virtual world with a wide diversity of geographical features such as mountains, valleys, forests, fields, caves, etc, and most of them have specific names and are related to the game story. The manual creation of those virtual worlds requires a team able to design heightmaps, adorn landscapes, ensure usability of the map (avoid unreachable places, for instance), create interesting places for players, etc. To help on that task, the development of a tool able to generate complex virtual worlds is useful to speed up the development of 3D games such as MMOs.

The solution proposed in this work is the development of a tool, called Charack[1], able to generate complex virtual worlds in real-time using noise-based techniques for terrain generation. Charack was designed to allow developers to use its features in order to generate 3D terrains for games, particularly MMOs, with minimal human intervention in the generation process. The content generation is made on demand. As the user moves along the world, the elements inside the user's view are processed and stored into the memory and the ones away from the user's view are removed. Even though the generation of all elements is based on random numbers, if the player visits an specific point A, then walks for miles generating a completely different set of landscapes, and returns to point A, the same previously seen landscape will be shown again. Charack handles separately the content generation of continents, topography and coastlines, so each of those elements can be independently adjusted in order to produce highly customized results.

This paper is organized as follows. Section 2 describes related work concerning the generation of finite or infinite virtual worlds. Section 3 presents the tool structure and the techniques used in the content generation process. Section 5 describes and illustrates the results that Charack produced. Finally section 6 presents a conclusion and ideas for future work.

2 Related works

There are several related works concerning the generation of finite or infinite virtual worlds. One of them creates an infinite city that is presented to the user on demand as it walks on the ground [Greuter et al. 2005]. The world were divided into a grid composed of several squares, called cells. The location of each cell is used with a global seed as an input for a hash function [Wang 2000]. The result of this function is used as a seed for a pseudo random number generator and it defines all the characteristics of the buildings within a cell. As a consequence of that approach the contents of a cell is always the same, no matter if the user moves and that cell is removed from the memory. That work was the ground zero for Charack development, however the original idea was changed in order to make the tool suitable to generate more types of terrains (mountains, plains, continents, etc.), not only streets and buildings. The approach of content generation made on demand was maintained, but the cells organization was removed.

[1] Charack is available at http://code.google.com/p/charack

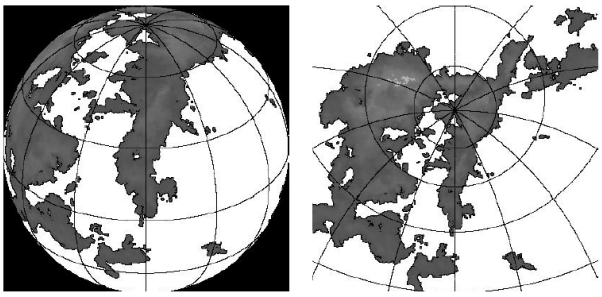


Figure 2: *Results of the planet map creator used by Charack*

The use of a procedurally generated world approach [Linda 2007] is very close to the concept of content generation aimed for Charack. In that work, a spherical planet is created as a result of a recursive division of a geometric shape, then noise functions are applied to the mesh to generate the heightmap. There is no distinction between the content generation approach for continents and the content generation for the terrain within the continents. As a result the continents are created by flooding the heightmap with a water plane, which will produce the coastlines based on the sea level height and the amount of ripples in the topography. The content itself is not generated on demand. Charack was created from an evolution of that idea, but with limitations. The world created by Charack handles differently the content generation of continents, coastlines and the heightmap and it also generates the content on demand, however it does not use a spherical approach.

Another related work was a tool used to build the SkyCastle multiplayer game engine [Häggström 2006; Häggström 2009]. For the heightmap generation, parameterized procedures and fractal based systems are combined in a layered approach: starting with a base map, the application merges a new map with the base one in each iteration. The new maps are pre-calculated and generated using Perlin noise [Perlin 1985]. To texturize the landscape and to adorn it with plants, several techniques are used [Cohen et al. 2003; Przemyslaw and Lindenmayer 1990; Lintermann and Deussen 1998; Weber and Penn 1995]. Charack uses a similar noise approach in order to create the terrain height, however it was not initially designed to generate extra content such as trees and plants.

The generation of a virtual world as a result of recursive subdivisions of a quadtree [Dollins 2002] is very similar to the Charack proposal. In that work, a world with huge proportions is created and its content is generated on demand as the user moves. The heightmap is created in a parameterized and multi-resolution way, so the closer the user is of place, the greater is the amount of detail there. There is also no distinction in the content generation process of continents, coastlines and land. The proposed heightmap generation is used by Charack, however continents and the coastlines generation process are completely different.

Another approach uses fractals affected by erosion for real-time, procedural generation of terrains [Olsen 2004]. For the erosion simulation, thermal [Musgrave et al. 1989] and hydraulic methods are used. Charack has no feature connected with the weather influence, even though it produces some sort of very basic erosion simulation when all sharp edges of the heightmap are removed by a smoothing algorithm.

The planet map creator based on the generation of a spheric world using a recursive subdivision of a tetrahedron is another approach concerning procedural content generation [Mogensen 2009]. All the generated information is part of a complete virtual world featuring highly customizable continents and oceans created as a result of a projection of pixels onto a sphere, a method similar to ray tracing [Whitted 1980]. That planet map creator is used by Charack as a starting point on the continent generation process. Figure 2 illustrates the planet map creator results.

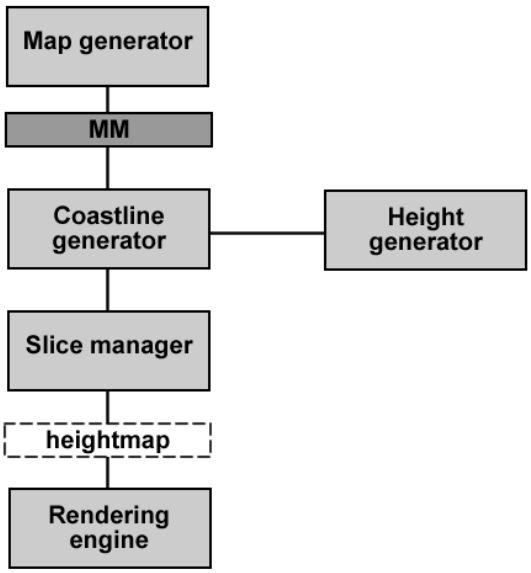


Figure 3: *Charack's basic structure*

3 Tool organization

3.1 Basic structure

Analyzing the related works, virtual worlds are generated through several approaches, but none of them handles differently the content generation for continents, coastlines and topography. Although there are variations in how the heightmap is created, the generation of continents is a result of a water flooding plane. This method allows the developers to focus on the content generation for the land, however it has a simple approach concerning continents and coastlines. The main idea and contribution of Charack is the content generation handled differently for each world element (continent, coastline, etc.), with an aggressive and specific approach for each one. This is a new approach for the content generation process, which is different from the the related works that focuses on content generation as a unified process. The term *pseudo infinite* used in the paper title is necessary due to physical limitations in computers hardware: an unsigned integer, for instance, can store a certain amount of data; if there were no physical limitations, the tool would be able to generate, in fact, an infinite world.

In order to create a virtual world that reaches the presented proposal, a top-down plan is used for the content generation. The Charack data flow begins in a macro view of the world, which are the continents, evolving to a micro view of the planet, which are the content generation for each vertex that will be drawn in the screen. Figure 3 shows Charack basic structure.

3.1.1 Maps generator

At the top of the chain is the map generator, which creates the continents that exist throughout the virtual world. This module is an encapsulation of the solution created by [Mogensen 2009]. When the tool is initiated, it uses a user defined seed to generate all the continents. Once the continents are generated, all the information related to terrain types (land, water and coast) are stored in a matrix called **macro-matrix (MM)**, which is used by all the other algorithms.

3.1.2 Slice manager

Below the MM and the map generator is the slice manager (SM). It extracts a portion of the virtual world (the user's view described as a regular mesh) and provide the rendering engine with information about the heightmap. In order to obtain the required information to create the heightmap, the slice manager uses the coastline generator (CG), which uses the height generator (HG) and the data stored in the MM.

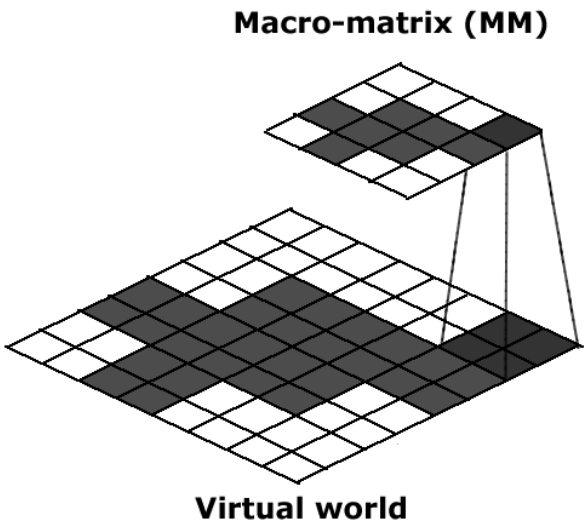


Figure 4: *Mapping the MM to the virtual world: each MM vertex represents several vertices in the virtual world*

In the context of the SM, there is no information about land or water, only a set of vertexes and their height values. Using the position of the user as a guide, the SM slices the virtual world and, for each collected vertex, it queries the CG in order to find out the height value for that vertex.

3.1.3 Height generator

The height generator (HG) defines the height value for each vertex in the virtual world. To ensure that the developers can create a customizable heightmap based on their needs, new functions to generate content can be added to the tool in a simple way.

3.1.4 Coastline generator

The coastline generator (CG) will map each vertex of the slice manager to the MM in order to find out the terrain type of that vertex. If the vertex being analyzed is mapped to a location in the MM that is described as water, then the CG assigns a height value equals to sea level for that vertex and returns it to the SM. If the vertex is mapped to a place described as (simple) land, then the CG will use the information provided by the HG in order to set the height value for that vertex. Finally, if the vertex is mapped to a place described as coast, then the CG uses its own structure (together with the MM) to set the height value for the that vertex.

The resulting virtual world technically has height and width defined by the maximum size of a signed integer, however it is physically impossible to generate a MM with such proportions. Since the MM is smaller than the virtual world, an MM's entry `(i, j)` represents several vertices in the virtual world. Figure 4 illustrates the MM mapping process.

The smaller is the MM size, the more vertices in the virtual world will be represented by the same entry in the MM. If the virtual world had `1000x1000` as its size and the MM had `10x10` as its size, for instance, it means that each entry of the MM represents 100 vertices in the virtual world. If one of these entries is described in the MM as coast, then there is an area of `100x100` vertices in the virtual world that must be a coast. When any of those vertices in that particular area of the virtual world is analyzed by the CG, it will work on it and return it with different values, which will result in a coastline for that area, not an area entirely filled with land or water.

3.1.5 Rendering engine

The rendering engine draws the created heightmap in the screen. The result is rendered as a triangles mesh that is texturized according to the height value of each vertex in the mesh.

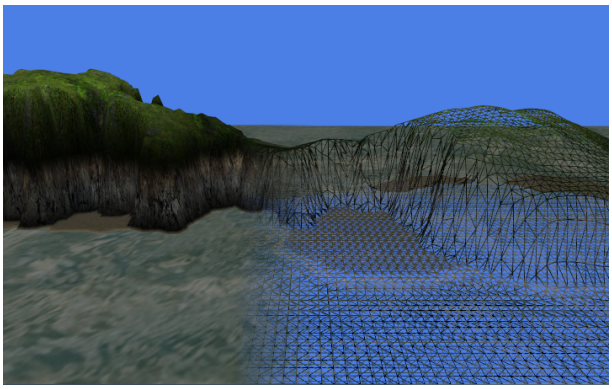


Figure 5: *Regular mesh describing the virtual world. At the right a wireframe rendering*

4 Implementation

The main problem concerning Charack's implementation was the content generation performed on demand. Based on the fact the user can only see what is inside the visible area, all the content generation algorithms need to take into account *only* the information that is available in the user's view. Even though this approach is efficient for resources management (process only the visible elements), it increases the complexity of the content generation algorithms.

The algorithm that generates mountains, for instance, has no way to determine where the mountain ends, because the world outside the user's view technically does not exist yet, it will be generated as the user moves. One approach to solve that problem would be the use of a function that describes the mountain backbone, but this function should not rely on begin/end points, because they could not exist in a certain time. If that function does not need any begin/end points, at least it would have to rely on the position of the user in the virtual world. If the function must be aware of some special points, those points have to be previously processed, which would break the on demand content generation concept.

In addition the algorithms are drastically affected by the fact that the information they receive in a certain time may disappear altogether in the next iteration, since the user can move and change the visible content. Using the example of the mountain generation, a mountain could present an abrupt end, because the points being used for the content generation left the user's view.

To circumvent these problems the content generation was divided into three main stages: infinite terrain, continents and height generation. The on demand content generation affects differently each of these stages and the problems and solutions related to each stage are described in the following sections.

4.1 Infinite terrain

The main idea for the content generation is to allow the user look at new content each time a significant movement is performed. As the user walks, the tool must be able to identify where the observer is located in the world in order to generate the content around that position. To solve this problem, the player is able to look at the screen and see a slice of the virtual world, however it has no explicit divisions such as cells. The slice is described as a regular mesh as figure 5 shows.

In order to texturize the sliced data a set of images is interpolated and managed by the shading language GLSL [OpenGL 2007]. The height value of the vertex defines the interpolation weight of each texture. As a consequence, a sand texture has higher weight for vertices featuring a low height value, for instance. Figure 6 shows all the available textures.

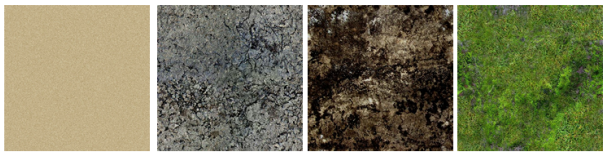


Figure 6: *Set of image used for terrain texturization*

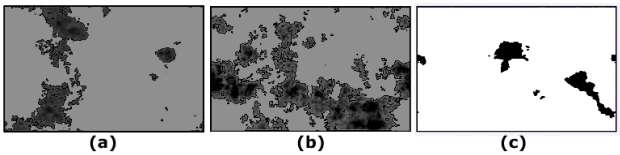


Figure 7: *Random planets generated with different seeds. (a) and (b) maps featuring height value information; (c) map featuring only information about what is water/land*

4.2 Height generation

The main idea for the terrain height generation is the use of a parametric function that informs the height value of each vertex. The function is seeded with the point location in the world. As a consequence, the function is able to describe all the height information in the world with no limitation concerning the world size. The processing time is related to the size of the user's view because the function uses the point information to calculate its height value. The height values are generated with a Perlin noise function.

4.3 Continents

The generation of continents and oceans has been proposed in order to break the monotony of a landscape composed only by land and to increase the similarity of the virtual world with real world landscapes. The solution for the continents generation consists in pre-process the land areas and store that information for later calculations. With that approach, the on demand content generation has been partially broken, since the continents are generated before all the other content, but it ensures a better control over water/land areas.

The continents generation is based on the planet map creator described in section 2. That planet generator was used because it has several parameterization options, such as the possibility to use a seed to manage all the random calculations. Figure 7 *(a)*, *(b)* and *(c)* illustrates the results obtained with the planet generator.

4.3.1 Problems with continent generation

As previously explained in section 3.1, each entry of the MM is mapped to several vertices in the virtual world. A direct consequence of that mapping process is the generation of large areas featuring straight land lines, as illustrated by figure 8. If there were no hardware limitation and if it were possible to generate a MM featuring the exact size of the virtual world, the matrix would contain the necessary resolution for the tool to accurately determine whether a vertex is land or not land, in a ratio of 1:1 (one MM entry is mapped to one world vertex). This approach, however, is not suitable because a matrix with such proportions consumes many resources and processing time. Although the tool allows customization of the MM size, tests showed that a MM featuring `800x800` as its size has enough information to be processed by all the other algorithms.

All images showing Charack results were generated from a world with `3x3` million vertices and a MM with size of `800x800`. It means that each MM entry represents 3750 vertices in the virtual world. Figure 8 illustrates the results obtained by the tool when no algorithm is used to generate extra content to fill the empty spaces in the virtual world. This figure illustrates a place in the virtual world that represents the transition between two different points of the MM (a land point and a water point). To explain what is happening, assume the tool is drawing the world at position `(x, y, z)`, which is the mapping result of an entry `(i, j)` in the MM, which is described as land; as the tool increases the coordinate in order to draw the landscape, each new position is mapped to the MM. If the result of the mapping process of the new coordinate, `(x + 1, y, z))` for instance, is still the entry `(i, j)` in the MM, then the tool will again draw a land vertex on the screen. Assuming that only at point `(x + 10, y, z)` the vertices start being mapped to a different entry in the MM, such as `(i + 1, j)` (assuming it is described as land), then all points before the position `(x + 10, y, z)` are drawn as water and all vertices after that location are drawn as land.

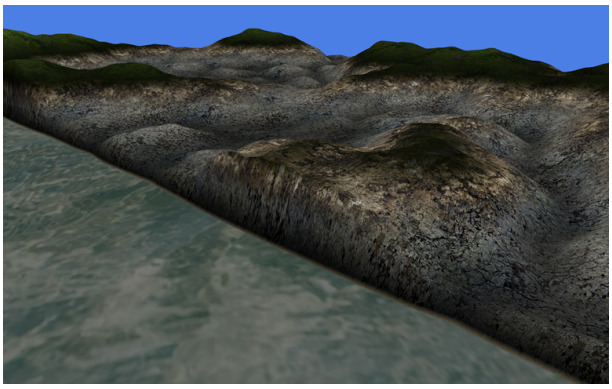


Figure 8: *The result of no algorithm to generate extra content to fill the discrepancies in the MM mapping process*

Figure 8 shows clearly when the world coordinates start being mapped to a different entry in the MM, which is when the tool replaces the land rendering with the water rendering. As a consequence of no algorithm being applied to generate content for that transition area, the user will move along the coastline and will see only straight lines.

4.3.2 Coastline disturbance

The mapping process of the vertices of the virtual world to the MM produces very unrealistic landscapes. A real world beach has a natural curvature and is not likely to have a length of 20 Km in a perfectly straight configuration. Although the objective of this work is not to create photo-realistic landscapes, such unreal beaches are not acceptable. To circumvent this problem, a coastline disturbance is applied to the locations where the mapping process is made between two MM points, one of them described as land and the other one described as water. The algorithm is described below.

The MM has a complete description of what is land and what is water in the virtual world. Each of its entries has a descriptor, which tells the other algorithms what type of terrain one vertex of the virtual world is after it is mapped to the MM. Charack features three types of terrain: water, land (continent) and offshore (land in contact with water). After the continents are pre-processed and stored in the MM, it only contains information about land (continents) and water.

From that moment, the first step of the coastline disturbance algorithm is performed. Using the current MM as its input, the algorithm scan each MM's entries updating the descriptor of all entries that represents coasts. A entry is classified as coast when at least one of its neighbors is water. After the algorithm ends, the MM contains the three types of terrain described before (water, land and offshore). The next step to apply disturbance to the coastline is the content generation based on the descriptor of each MM entry. When the tool is creating content to be drawn on the screen, each vertex being drawn is tested against its descriptor in the MM. If the vertex is mapped to a land entry in the MM, then the function will set a height value for that point. If the vertex is mapped to a water entry in the MM, then the function will set the sea level height to that point. Finally if the vertex is mapped to a offshore entry in the MM, then the function will disturb the land/water information of

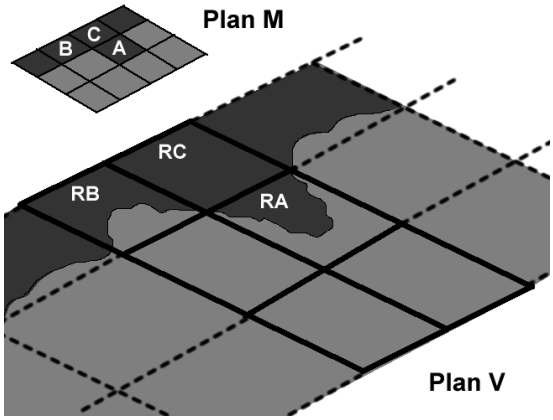


Figure 9: *Coastline disturbance algorithm*

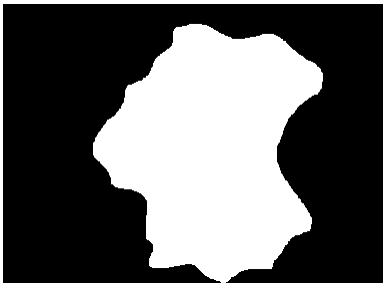


Figure 10: *Small heightmap generated by the coastline disturbance algorithm*

that vertex, which will result on a non-straight coastline. Figure 9 illustrates the algorithm.

The MM entries `A` and `B` have a descriptor indicating that they are classified as a coast. Plan `M` describes the MM and plan `V` describes the result of the mapping process between them. It is not described in the figure, but each block of plan `V` is composed of several vertices, while each block of plan `M` represents only one MM entry. The MM entry `C` is mapped to a massive block of land in the plan `V`, as its descriptor tells the tool that it is a entry described as land. The entry `A` would also be mapped to a massive block of land, but with the intervention of the coastline disturbance algorithm it is mapped to a different configuration. During the content generation for the vertices that are inside the block `RA`, the coastline disturbance algorithm alters the land/water information for each vertex, so that the block will not be composed of land or water vertices only, but a combination of them instead.

The implementation of that process is based on a noise function and random numbers with a parametric function deciding what is land and what is water for all vertices described as a coast in the MM. Using the vertex position in the block `RA`, the function maps that information into a spectrum of values created by a Perlin noise function. What the parametric function does is check if the hash of the vertex being analyzed is inside or outside of the spectrum. The process can be illustrated as a height test of a value against a small heightmap (which is created as a result of the noise spectrum): if the return of the noise function for that vertex is greater than a certain value (which is the granularity of the block being analyzed), then it is classified as land, otherwise it is classified as water. The higher is the granularity of the block, the greater is the amount of land on that location. Figure 10 illustrates the small heightmap generated by the coastline disturbance algorithm when block `RA` is being processed.

4.3.3 Beaches

The coastline disturbance algorithm minimizes the problem of unrealistic coastlines, but the outcome is not quite good enough. When Charack is rendering a slice of the world, for each vertex described as land a height value is set it; the same applies to the vertices that are described as water, but in that case the height value

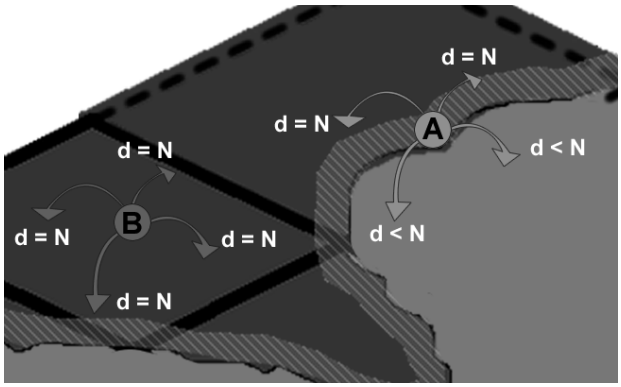


Figure 11: *Beach generation algorithm*

is always the same (the sea level). As a direct result of that approach if the tool is rendering a set of vertices which belongs to a mountain and the next vertices are described as water at the MM, the landscape will features a ”step”. It happens because the mountain backbone was generated very close to the water, which means that its rendering is abruptly interrupted when Charack finds vertices described as water. Although there are cliffs in the real world, they are not present in all coasts. To solve this problem, a special algorithm is applied in order to create beaches in certain locations of the world, which makes the generated landscape looks more realistic.

The beach generation algorithm is performed right before the content is rendered on the screen. After Charack maps the vertices to the MM and after the coastline disturbance algorithm is performed, the result is a heightmap ready to be rendered. The heightmap is treated by the beach creator algorithm before being drawn on the screen, as figure 11. The procedure scans all the vertices in the map and for each one its distance to a near water vertex is checked. The vertices around the target are mapped directly to the MM, so the only information that is used from the heightmap is the vertex location in the world (which is necessary to map it to the MM). The checking process is performed in four directions (right, left, up and down) and it ends when a water vertex is found or when `N` vertices were analyzed. After that, the four distances are added and used to calculate the height of the beach. The possible results are:

- If the vertex has `4N` as its distance (Figure 11, point `B`), it means the tool has iterated through the four possible directions and found no water. In this case, the height value for the vertex remains the same. It happens to all the vertices that are within the continent or on the coast but away from the water: they do not belong to the beach area and their height value is defined by the height generator;

- If the vertex has a value smaller than `4N` as its distance (Figure 11, point `A`), then its height value will be recalculated, because the vertex is located at the beach. The greater is the distance from that vertex to the water vertex, the greater is the height value that will be applied. The height variation is calculated within a range of `[T,B]`, where `T` is the maximum height and `B` is a minimum height value of all vertices in the beach. The result of that approach is a beach featuring higher height values near the continent and lower height values near the water.

Figure 12 shows the results of the beach generator algorithm.

4.3.4 Island generator and beach disturber

The coastline disturbance and the beach generation algorithms make Charack able to generate more realistic landscapes. The final result, however, presents a well defined pattern, which is unusual to happen in the real world, where the lines and landscapes are more likely to follow a random patterns. If the player walks in the virtual world only through the coast, he would see beaches with the same configuration and no islands along the path. To avoid that problem,

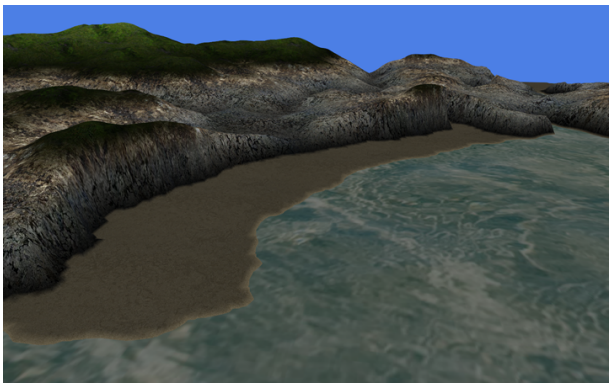


Figure 12: *Results of the beach generator algorithm*

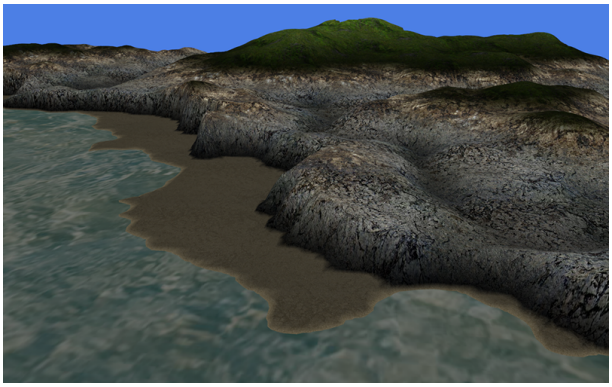


Figure 13: *Results of the beach disturber algorithm*

two new algorithms are applied to the coastline vertices: beach disturber and island generator.

The **beach disturber** disturbs the distance used to calculate the water vertices neighboring a certain vertex. Instead of using `N` as value to calculate the distance from the vertex to the water, the beach disturber uses the vertex position as a seed and generates a new value that will be used as the distance. Using this technique, the beach disturber is able to change the size and shape of the beaches, so that certain regions may have a greater amount of sand than others. Figure 13 shows the beach disturber results.

The **island generator** creates land portions in some MM entries. After the MM is created and all the descriptors are configured, the generator iterates through all entries described as coast and for some of them it sets a flag describing that region as a place that features islands. Each vertex mapped to that special regions of the MM has its position used as a hash that is tested against a spectrum created by a noise function. According to the test result, the vertex is classified as land, so a group of vertices classified as land will produce an island. The noise spectrum used for that are different from that one used in the coastline disturbance algorithm, since the expected outcome are small portions of land (islands). Figure 14 shows the island generator results.

5 Results

This section aims to evaluate each of the techniques used in the content generation process, explaining the obtained results for each approach. It is important to highlight that Charack's purpose is not the generation of real or photo-realistic content, but elements that can be used to create a 3D game scene. A result is classified as graphically acceptable if it can be integrated into a game and not surprise the player in a negative way, such as a pyramidal mountain instead of a smooth mountain.

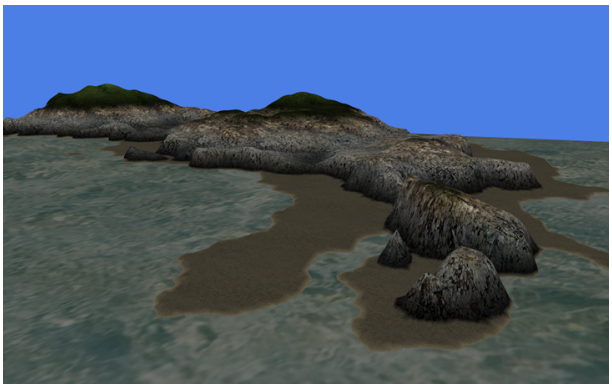


Figure 14: *Island generated by the island generator (beach area has been influenced by the beach disturber)*

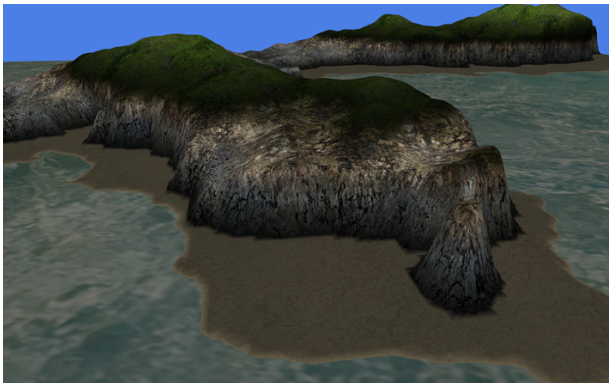


Figure 15: *Continents and oceans generated by Charack*

5.1 Continents evaluation

The time spent for the continent generation is directly proportional to the size of the specified MM. The reduction of the MM size to `800x800` yielded significant performance improvements. As a consequence, the smaller is the MM size, the more linear and square are the coastlines of each continent. To avoid that problem, it is possible to adjust the coastline disturbance algorithm in order to make it produce more aggressive changes in the coastlines. Figure 15 shows the continents and oceans generated by the Charack.

5.2 Terrain height evaluation

The terrain height generated by Charack is fully customizable. The tool has a built-in terrain height generator based on Perlin noise, however it was designed for testing purposes only. The main focus of the present work are the continents and the coastline generation, so any activity related to terrain height generation was very superficial and presents no contribution. Figure 16 shows the terrain height created by the built-in generator.

5.3 Coastline evaluation

The coastline generation is composed of two main elements, a global and a local one. The global one only uses data available in the MM in order to create the coastlines, as described in section 4.3.1. The final result for that approach is a unreal straight coastline. Figure 17 shows two completely straight coastlines which has no content generation algorithm applied to them.

After the coastline disturbance algorithm was introduced, Charack started producing more acceptable landscapes. Figures 18 and 21 show small bays in some places of the coast. It happened because at those locations the coastline disturbance algorithm created pieces of land towards the ocean and at the same time the beach disturber reduced the amount of sand on the newly created land pieces. Charack is also able to create gulfs, which are large bays, but it is not possible to predict the exact location where those bays will hap-

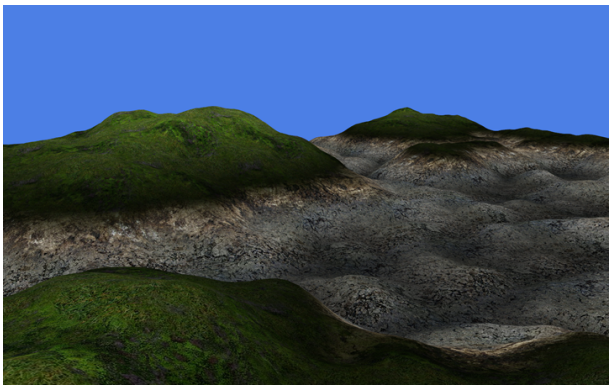


Figure 16: *Terrain heightmap created by Charack's built-in generator*

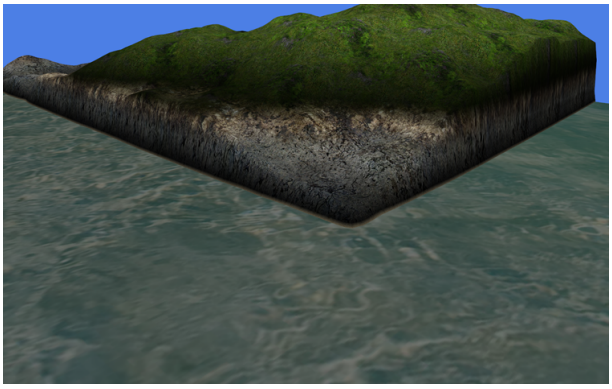


Figure 17: *The interception of two coastlines with no extra content being applied to them*

pen because it depends on a set of specific values (location, beach size, etc).

Figures 19 and 20 show the final result obtained with the combination of all the previously described algorithms: coastline disturbance, beach disturber and island generator.

5.4 Performance evaluation

All tests were performed running Charack on Windows Vista on a Intel(R) Core(TM)2 Duo 1.66Gz, with 2Gb RAM and a graphics card NVidia 8600 GT, using Microsoft Visual C++ 2008 Express Edition to compile the source code. Figure 22 shows the time that Charack takes to process each step on the virtual world generation: height calculation for each vertex, coastline generation, beach generation and the rendering process. The `X` axis shows the world slice size in vertexes, e.g. 200 means a regular mesh of `200x200` vertexes. The `Y` axis shows the time in miliseconds that Charack takes to generate the respective world slice.

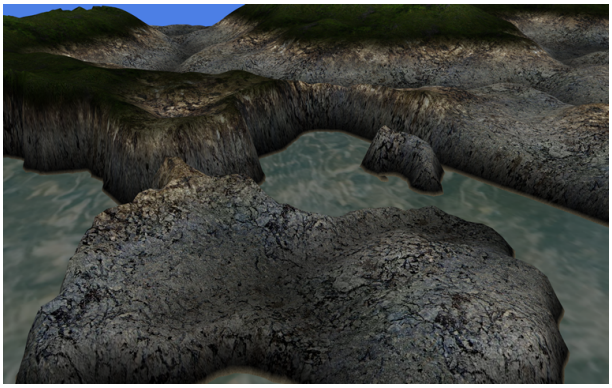


Figure 18: *Small bay featuring rocks*

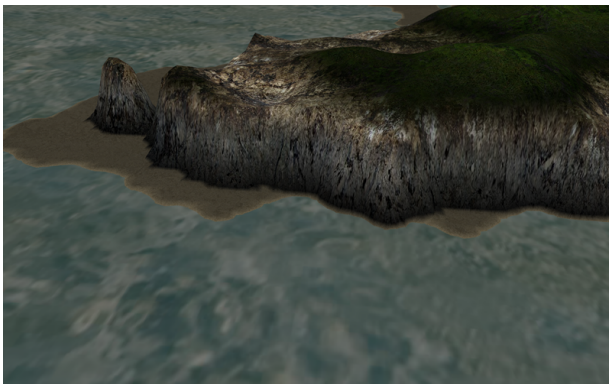


Figure 19: *Coastline featuring almost no beach area*

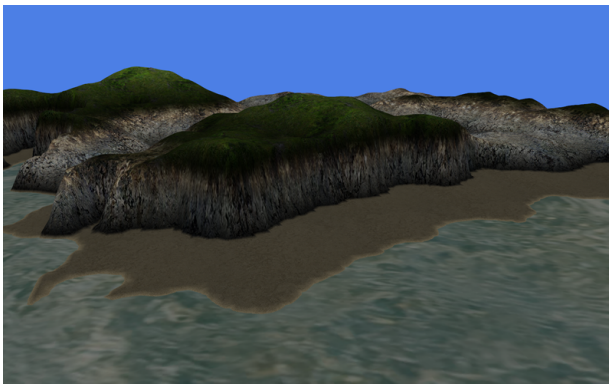


Figure 20: *Coastline featuring beaches with different sizes*

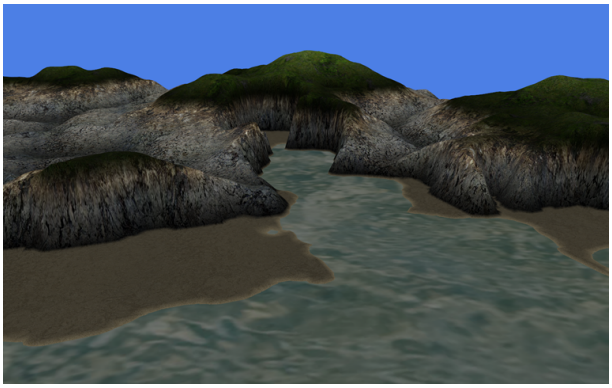


Figure 21: *Result of the coastline disturbance algorithm*

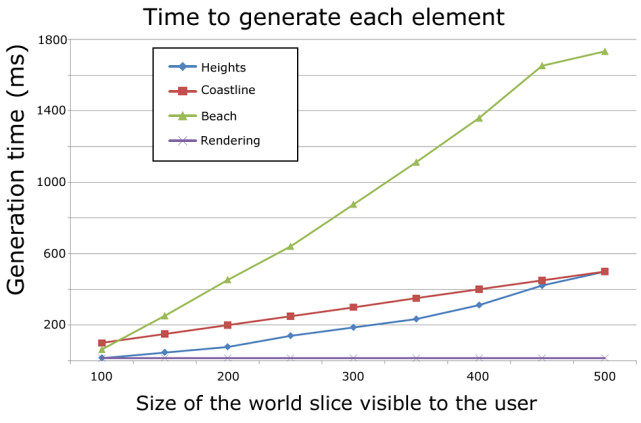


Figure 22: *Time that Charack takes to process each step on the virtual world generation*

If the world slice visible to the user has a size of `100x100` pixels, then Charack takes about `200 ms` to generate a complete landscape (the sum of times for all steps). This process is not executed for each rendered frame, it is only performed when the user moves too far from his initial position, what makes Charack remove the old content from the memory and replace it with new data. If a slice with size `300x300` pixels is used, then Charack takes about `1000 ms` to generate the landscape. Most of the time is spent on the beach creation because Charack has to analyze each vertex on the screen and its four neighbours to decide if it is a beach point or not.

The coastline generation is not significantly affected by the slice size because the algorithm does not analyze an arbitrary amount of vertexes to generate the content. It uses a combination of noise functions and MM meta data in order to generate the required information.

According to the chart the slowest step performed by Charack is the beach generation. The second slowest step is the coastline generation, closely followed by the height generation for each vertex.

6 Conclusion and future work

The automated creation of virtual worlds is one of the available methods that can help developers to create games featuring detailed environments in less time and using fewer resources. Unlike the purely non-automated approaches where a game designer has to design the entirely world, an automated approach is able to generate a complete world with almost no human interference. There are several researches on that subject using different approaches and focusing on a wide range of results.

This paper presented a tool able to generate pseudo-virtual worlds featuring different continents, coastlines and landscapes. Using a combination of algorithms and methods for content management, the tool is able to create beaches, islands, bays and coastlines similar to the ones found in the real world.

One of the Charack's contributions is the ability to generate arbitrarily large pieces of land focusing on coastline generation. The development of the present work aimed to handle separately the content generation for all elements in the world (continents, terrains, etc.). The main point in the work is the coastline generation, not the content inside the continents. The final virtual world can be huge: a player with a 100 vertices per second speed in a virtual world generated with the maximum value allowed by a integer will take about 1 year and 3 months to across the whole world.

One suggestion of future work is the enhancement of the height generator, which currently produces a very simple result. Another suggestion is the addition of new types to the MM's descriptors, such as deserts, forests and cities. All the new content can be created tweaking Charack's content generator algorithm in order to produce variations in the current results, such as lowering the height values of all vertices in an area described as a desert, or increasing them in a volcanic area. Another suggestion is to port all the content generation algorithms to the CUBA platform [nVidia 2009]. The generation of each world vertex can be calculated separately, so the CUBA parallelism capabilities can be fully used. It will drastically improve Charack's performance and it will allow the generation of a bigger world slice to be displayed in the screen. Another suggestion is the addition of rivers, which can be done with a new MM descriptor and some changes in the content generation algorithm.

References

BLIZZARD ENTERTAINMENT, 2007. World of warcraft. Available at: http://www.blizzard.com.

CLARK, N. L. 2006. *Addiction and the Structural Characteristics of Massively Multiplayer Online Games*. Master's thesis, University of Hawai, Hawai.

COHEN, M., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. In *Siggraph 03 Conference proceedings*.

DOLLINS, S. C. 2002. *Modeling for the Plausible Emulation of Large Worlds*. PhD thesis, Brown University, United States of America.

DUCHENEAUT, N., YEE, N., NICKELL, E., AND MOORE, R. J. 2006. Alone together exploring the social dynamics of massively multiplayer online games. In *CHI 2006 Proceedings*.

GREUTER, S., PARKER, J., STEWART, N., AND LEACH, G., 2005. Realtime procedural generation of 'pseudo infinite' cities.

GRIFFITHS, M. D., DAVIES, M. N., AND CHAPPELL, D., 2003. Breaking the stereotype the case of online gaming.

HÄGGSTRÖM, H. 2006. *Real-time generation and rendering of realistic landscapes*. Master's thesis, University of Helsinki, Finlândia.

HÄGGSTRÖM, H., 2009. Skycastle - free multiplayer game engine focusing on player creativity and world simulation. Available at: http://www.skycastle.org/.

LINDA, O. 2007. Generation of planetary models by means of fractal algorithms. Tech. rep., Czech Technical University.

LINTERMANN, B., AND DEUSSEN, O., 1998. A modelling method and user interface for creating plants. Computer Graphics Forum.

MOGENSEN, T. ., 2009. Instant planet generator. Available at: http://www.eldritch.org/erskin/roleplaying/planet.php.

MUSGRAVE, F. K., KOLB, C. E., AND MACE, R. S., 1989. The synthesis and rendering of eroded fractal terrains. Computer Graphics, Volume 23, Number 3, July 1989, pages 41 to 50.

NVIDIA, 2009. Cuda. Available at: http://www.nvidia.com/cuda.

OLSEN, J., 2004. Realtime synthesis of eroded fractal terrain for use in computer games.

OPENGL, 2007. Opengl shading language. Available at: http://www.opengl.org/documentation/glsl/.

PERLIN, K. 1985. An image synthesizer. In *SIGGRAPH*, 287–296.

PRZEMYSLAW, P., AND LINDENMAYER, A., 1990. The algorithmic beauty of plants. Springer-Verlag.

SONY ENTERTAINMENT, 2007. Everquest. Available at: http://everquest2.station.sony.com/.

WANG, T., 2000. Integer hash function. Available at: http://www.concentric.net/ Ttwang/tech/inthash.htm.

WEBER, J., AND PENN, J., 1995. Creation and rendering of realistic trees.

WHITTED, T. 1980. An improved illumination model for shaded display. *Communications of the ACM 23*, 6 (June), 343–349.

Design and implementation of a flexible hand gesture command interface for games based on computer vision

João L. Bernardes[1] Ricardo Nakamura[2] Romero Tori[1]

[1,2]Escola Politécnica da USP, PCS, Brazil [1]Centro Universitário SENAC, Brazil

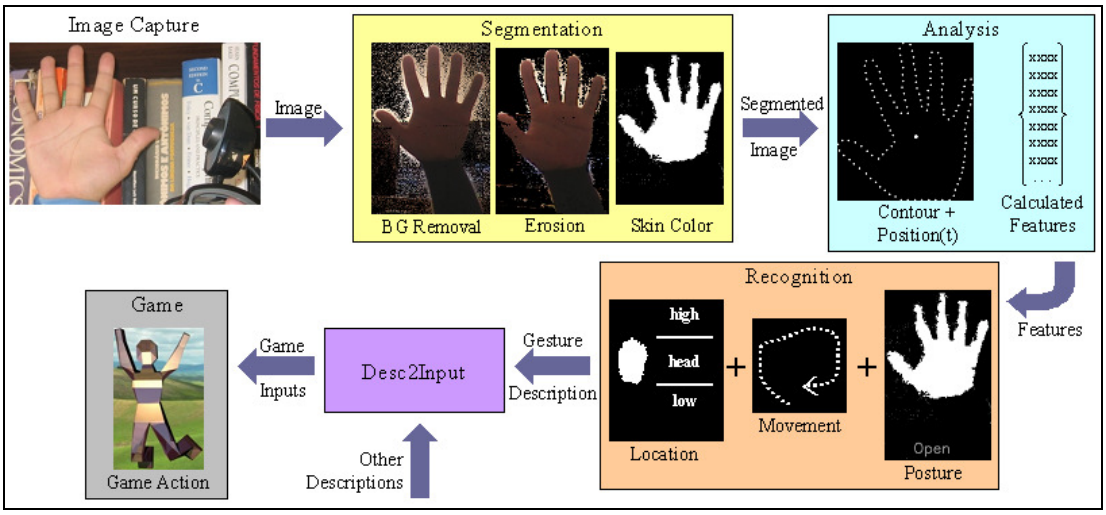


Figure 1: Gestures2Go

Abstract

This paper describes a command interface for games based on hand gestures defined by postures, movement and location. The large variety of gestures thus possible increases usability by allowing a better match between gesture and action. The system uses computer vision requiring no sensors or markers on the user or background. The analysis of requirements for games, the architecture and implementation are discussed, as well as the results of several tests to evaluate how well each requirement is met.

Keywords: computer vision, gesture recognition, human-computer interaction, electronic games

Authors' contact:
{joao.bernardes, ricardo.nakamura}@poli.usp.br, tori@acm.org

1. Introduction

The possibility of relaying commands to a computer system using one's own hands and gestures has interested researches and users for a long time and was one of the first topics in user interface research, partly because it uses well-developed, everyday skills [Bowman 2005]. With the computational capacity available today and widespread use of image capture devices, even in domestic systems it is possible to implement this sort of interaction using computer vision. This brings the benefit of leaving the user's hands free of any gloves, cables or sensors. Gestures2Go, the system described here, provides this functionality and its implementation (in C++, illustrated in figure 1) is focused on electronic games.

Games are an ideal platform to test and popularize new user interface systems, for several reasons, such as an increased user willingness to explore in this medium [Starner et al. 2004]. There are many examples of academic research developing and studying new interfaces with games, particularly incorporating Augmented Reality [Bernardes et al., 2008]. The game industry has also introduced new (or of previously restricted use) interfaces and devices to the public. From the joystick to increasingly complex gamepads and controllers shaped as musical instruments, from datagloves to "pistols" that function as pointing devices and even haptic devices [Novint 2009], many are such examples, to the point that, today, some professionals are encouraged to play games to improve job-related skills [Dobnik 2004].

On the other hand, both the industry and academia acknowledge that new, more natural (and hopefully fun) interfaces are one way to attract new consumers to this economically important but still restricted market [Kane 2005]. And in the past few years, the search for these interfaces has been more widespread, continuous, well-publicized and commercially successful. After a popular gaming platform introduced motion and tilt detection in a simpler controller as its most innovating feature [AiLive 2007], motion detection was quickly added to other platforms and games and continues to be researched and improved upon. Several portable gaming systems, in particular, are taking advantage of motion and tilt sensing, touchscreens and even microphones in their interface. More recently still a project was unveiled to add interaction based on recognition of full-body motion, speech and faces to a popular platform [Snider 2009].

Despite this ebullience in game interfaces, the use of hand gestures, especially leaving the user's hands free, has seen little academic or commercial research in this area and is usually limited to analyzing only hand movement or a small number of hand postures. One of Gestures2Go's objectives is greater flexibility, to allow the use of a greater variety of gestures (currently defined by hand postures, movement or location and using both hands). Another important goal is that it must be easy to use for both players and developers. Gestures2Go should also be usable with existing games (designed for traditional interfaces) and allow multimodal interaction. These and other requirements arose, during system design, from an analysis focusing specifically on gestures and on game applications. Many of the same requirements exist in other applications as well, such as education or virtual and augmented reality, and the authors believe this system may be well suited for these applications, but will leave this discussion outside the scope of this paper.

2. Related Work

A few works have been proposed recently to use free hand gestures in games using computer vision. A multimodal multiplayer gaming system [Tse et al. 2007] combines a small number of postures, their location on a table-based interaction system and speech commands to interact with games and discusses results of using this platform to interact with two popular games. Interpreting movements or postures of the arms or the whole body is also usual. A body-driven multiplayer game system [Laakso & Laakso 2006] uses 8 postures of the two arms viewed from above, plus player location, to design and test the interaction in several games. Going further, tests with both functional prototypes and Wizard of Oz prototypes indicate that body movement patterns (such as running, swimming or flying), rather than specific gestures or trajectories, may be used to trigger similar actions on game characters [Hoysniemi et al. 2005].

Other tools facilitate the use of gesture recognition for applications in general, not only games. ICondensation [Isard & Blake 1998] is a probabilistic framework that allows the combination of different observation models, such as color and contours. HandVu [Kolsch et al. 2004] also uses condensation but provides a simpler interface to track hands in six predefined postures using skin color and a "flock" of Haar-like features. GART [Lyons et al. 2007] provides a high level interface to machine learning via Hidden Markov Models used to train and recognize gestures that consist only of movements (detected by sensors such as a camera, mouse or accelerometers). It is interesting to note that HandVu and GART can be combined to allow robust hand tracking and a larger number of gestures (combining postures and movement, like Gestures2Go) than either one isolated. Finally, EyesWeb [Camurri et al. 2003] is a framework with a graphical programming interface that presents several tools and metrics for segmentation and analysis of full body movements.

The literature regarding gesture recognition in general is vast and a complete review is beyond the scope of this paper, especially since established and comprehensive reviews [Pavlovic et al. 1997] as well as more recent but still comprehensive discussions [Imai et al. 2004] are available. Other works, when relevant to this implementation or future developments, are discussed in the correspondent sections.

3. HCI and Game-specific requisites

Both the use of gestures and having games as an application bring specific requirements to an interface and analyzing these requirements was one of the most important steps in designing Gestures2Go. For gesture-based interfaces, current research [Bowman et al. 2005, Shneidermann et al. 1998] point out the following:

Gestures are most often used to relay singular commands or actions to the system, instead of tasks that may require continuous control, such as navigation. Therefore, it is recommended that gestures be part of a multimodal interface [Bowman et al. 2005]. This also brings other advantages, such as decoupling different tasks in different interaction modalities, which may reduce the user's cognitive load. So, while gestures have been used for other interaction tasks in the past, including navigation [Mapes & Moshel 1995], Gestures2Go's primary requisite is to allow their use to issue commands. Issuing commands is a very important task in most games, usually accomplished by pressing buttons or keys. Often, games feature a limited number of commands, not even requiring all the buttons in a modern gamepad. Since other tasks, especially navigation, are very common as well, another requirement that naturally arises is that the system must allow multimodal interaction. Massively Multiplayer Online games (MMOs), in particular, often have much of their actual gameplay consisting of navigation plus the issuing of several commands in sequence [Fritsch et al. 2005].

Gesture-based interfaces are almost always "invisible" to the user, i.e. they contain no visual indicators of which commands may be issued at any particular time or context. To reduce short term memory load, therefore, the number of possible gestures in any given context, but not necessarily for the entire application, must be limited (typically to 7±2 [Miller 1956], or approximately 5 to 10 gestures). The gestures must also be highly learnable, chosen from the application domain so the gesture matches the intended command. Changing gears in a racing game, for instance, could be represented by pulling a fist towards or away from the user with the hand relatively low, as if driving a stick shift car, and pausing the game could be associated with an open palm extended forward, a well-known gesture meaning "stop". This means that while the system is not required to deal with a large

number of different gestures at any one time (which simplifies the implementation), being flexible by having a large number of possible gestures to choose from, so the interface designer may pick the most appropriate to associate with each user action, is indeed a requirement. Systems that violate either of these two requirements, requiring the memorization of a large number of gestures or limiting the space of possible gestures to only a few postures or movements, make the interface harder to learn and later to remember, reducing its usability.

The examples above (changing gears and stop) also show that the choice of each gesture for the interface depends not only on the application, context and command, but is also heavily culture-dependant, because the cognitive meaning of gestures may vary. In the case of gesture-based games, therefore, and with games being such a global market, localization could also entail changing which gesture is associated with each action [Bernal-Merino 2007]. All this leads to the requirement that the vocabulary of gestures in each context of the interface, while small, must be as simply and quickly modifiable as possible. Systems that require retraining for each set of possible gestures, for instance, could prove problematic in this case, unless such training could be easily automated.

The interface should also accept small variations for each gesture. Demanding that postures and movements be precise, while possibly making the recognition task easier, makes the interaction considerably harder to use and learn, demanding not only that the user remember the gestures and their meanings but also train how to do them precisely, greatly reducing usability.

It could be argued that, for particular games, reducing the usability could actually be part of the challenge presented to the player (the challenge could be remembering a large number of gestures, or learning how to execute them precisely, for instance). While the discussion of whether that is a good game design practice or not is beyond the scope of this paper, Gestures2Go opts for the more general goal of increasing usability as much as possible. This agrees with the principle that, for home and entertainment applications, ease of learning, reducing user errors, satisfaction and low cost are among the most important design goals [Shneidermann et al. 1998].

The system should also allow playing at home with minimal setup time required. Players prefer games where they can be introduced to the action as soon as possible, even while still learning the game and the interface [Hong 2008]. Therefore, the system should not require specific background or lighting conditions, complex calibration or repeated training. Allowing the use of the gesture-based interface with conventional games is also advantageous to the user, providing new options to enjoy a larger number of games. From the developer point of view, the system should be as easy as possible to integrate within a game, without requiring specific knowledge of areas such as computer vision or machine learning.

Finally, processing and response times are important requirements. Despite the growing availability of multi-core gaming platforms, it is still desirable that gesture recognition processing time be as low as possible, freeing processing power to other tasks such as artificial intelligence and physical simulation. It is limited by the acceptable response time, which, in turn, depends on the game. Performing a gesture, for instance, will almost always be slower than pressing a button or key, so this sort of interface is probably not a good choice for reflex-based games such as first person shooters. A genre that has already been mentioned as a good match for this sort of interface is MMOs. Not only much of their gameplay consists of navigation and issuing commands, MMOs use several strategies to deal with network latency [Fritsch et al. 2005] that also result in not penalizing the slower input from gestures, when compared, for instance, with button pressing. Such strategies include reducing the number of commands necessary in a fixed amount of time (for instance, it is common to "enter or exit attack mode", instead of repeating a command for each attack) and accepting the queuing of only one new command while the action triggered by the last one has not finished (and actions are set up to take some time, usually spent with animations or special graphical effects). In the game Everquest 2, for instance, Fritsch et al. report that the use of these strategies, with actions usually taking 1000ms, makes the game playable with latencies of up to 1250ms. A more practical bound, however, pointed after the analysis of several related works, is around 250ms for interactive games [Henderson & Bhatti 2003]. In a setup such as the one described above, that would leave one second to be divided between gesture performance and system response time and this is the parameter that will be used for Gestures2Go. This applies, of course, even for games designed for regular interfaces. When designing a game specifically to explore gestures, similar game design strategies or even new ones could be adopted to compensate for the time the user spends performing the gesture.

4. Gestures2Go

Because one of the requirements for this system was ease of use, both for the player and the developer, it was named Gestures2Go to imply that the gesture recognition is ready to go, to take home, with little extra work. It consists of an abstract framework that divides the system in modules and defines the interface between these modules and, currently, of a single, simple implementation of this framework. It is important to note that the requirements discussed in section 3 apply to the current implementation, which is focused on games, and not to the abstract framework.

The computational task of identifying a gesture from a known vocabulary of possibilities is often divided in gesture modeling, analysis and recognition [Pavlovic et al. 1997].

Gesture modeling consists in how a gesture is defined by the system, from a computational point of view (since definitions of gesture abound in other areas). Gesture2Go's abstract framework defines a gesture as an initial hand posture, an optional movement of the entire hand through an arbitrary path and a final posture, which is optional if the movement is omitted but mandatory otherwise. The starting location of the hand, relative to the user's head (left or right, above, below or roughly aligned with the head), is also an optional parameter of this definition, since it often changes the meaning of a gesture. This means that a gesture may consist of a single posture, of an initial and a final posture or of an initial posture, a movement and a final posture, all depending or not on the initial hand position. It also means that changes of posture during the movement are not taken in consideration, since these changes rarely have semantic meaning [Quek 1994]. While the abstract framework also includes variable parameters in the gesture definition (such as speed or pointing direction), the simple implementation described here does not deal with parametric gestures. Finally, the abstract framework does not specify how each part of the gesture definition is actually modeled (each is identified by a string or numerical ID), so it can vary in each implementation. The hand posture could, for instance, be modeled as a collection of values for the degrees of freedom of a particular hand model, or it could consist of a list of 2D or 3D points of the hand's contour.

During the analysis phase, the gesture's spatial and temporal parameters (which depend on each model) are obtained from sensor data (in this case, from an image or a set of images) and this data is used during the recognition phase to identify the gesture within the vocabulary of possibilities. Analysis and recognition are often, but not necessarily, tightly inter-related.

4.1 The Abstract Framework

Figure 2 shows a UML Activity Diagram representing Gesture2Go's object flow model.

G2gGesture is responsible for the gesture model, while *G2gAnalysis* and *G2gRecognition* define the interfaces for the classes that will implement gesture analysis and recognition. To these activities are added image capture and segmentation. *G2gCapture* provides an interface for capturing 2D images from one or multiple cameras or pre-recorded video streams (mostly for testing). The images must have the same size, but not necessarily the same color depth. A device could provide, for instance, one or more color images and a grayscale image to represent a dense depth map. *G2gSegmentation* should usually find in the original image(s) one or both hands and possibly the head (to determine relative hand position).

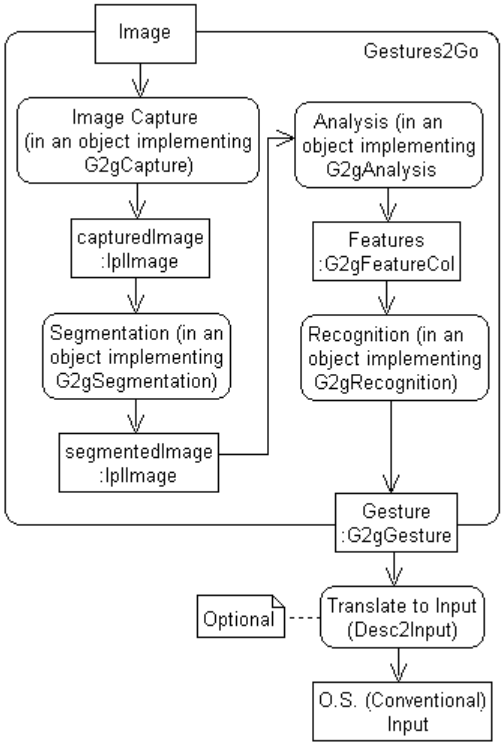


Figure 2: Gesture2Go's Object Flow Model

Figure 2 shows that the usual flow of information in Gestures2Go in each time step is as follows: one or more images serve as input to the image capture model, which makes these images available as an OpenCV's *IplImage* object [OpenCV 2009]. The segmentation uses this image and provides a segmented image as an object of the same class (and same image size, but not necessarily color depth). Based on the segmented image, the analysis provides a collection of features as a *G2gFeatureCol* object which are in turn used by the recognition to output a gesture.

G2gFeatureCol is a collection of *G2gFeature* objects. *G2gFeature* contains a identifier string to describe the feature and either a scalar and an array of values (more often used) or an image (useful, for instance, for features in the frequency domain). *G2gFeature* already defines several identifiers, for those features most often found in the gesture recognition literature, to facilitate the interface between analysis and recognition, but user-created identifiers may also be used.

Desc2Input is an optional module that accompanies but is actually separate from Gestures2Go. It is responsible for facilitating, in a very simple way, both multimodal input and integration with games or engines not necessarily aware of Gesture2Go. It simply translates its input, which is a description (a numerical or string ID or a XML description, for instance) that may be supplied either by Gestures2Go or any other system (and here lies the possibility of multimodal interaction), into another type of input, such as a

system input (like a key down event) or input data to a particular game engine. In one of the tests, for instance, gestures are used for commands and a dancing mat is used for navigation.

Because this architecture consists mostly of interfaces, it is possible to create a single class that, through multiple inheritance, implements the entire system functionality. This is usually considered a bad practice in object orientation (should be avoided) and is actually one of the reasons why aggregation is preferred to inheritance [Eckel 2003]. There are design patterns that could have been used to force the use of aggregation and avoid multiple inheritance, but Gestures2Go opts for allowing it for a reason. Gesture recognition may be a very costly task in terms of processing, and must be done in real time for the purpose of interaction. Many algorithms may be better optimized for speed when performing more than one task (such as segmentation and analysis) together. Furthermore, analysis and recognition are very tightly coupled in some algorithms and forcing their separation could be difficult. So, while it is usually recommended to avoid using multiple inheritance and to implement each task in a different class, making it much easier to exchange one module for the other or to develop modules in parallel and in teams, the option to do otherwise exists, and for good reason.

Finally, all Gestures2Go classes must implement *init( )* and *cleanup( )* methods which are preferred to using the *new* and *delete* operators (the system is implemented in C++) to avoid problems with multiple inheritance and with synchronization.

4.2 Implementation

The requirement analysis pointed that an implementation of the abstract framework described above specifically for games should have the following characteristics: minimum need for setup, low processing demand even though the response time may be relatively high, a high number of possible gestures but with only a small and easily modifiable vocabulary in any one context, tolerance to variations in the execution of gestures, allow multimodal interaction and make development of games using gestures as easy as possible. With these requirements in mind and assuming that a single player in the scene will interact through gestures, this implementation attempts to find the simplest solution for each of the activities shown in figure 2.

Segmentation is based on skin color, to find both hands and the head. A *G2gSimpleSkinSeg2* (a class which implements *G2gSegmentation*) object performs a simple threshold operation on the captured image, in the HSV color space, taking in account both hue and saturation. For most people, skin color lie in a small interval between the red and yellow hues, due to blood and melanin, respectively [Fleck & Forsyth 2009], so using hue is a good way to identify a large range of lighter or darker skin tones, even in different illumination conditions. Saturation is used mostly to remove regions that are either too light or too dark and may end up showing a hue similar to the skin.

At first, fixed average values and tolerances were adopted for the skin's hue and saturation. Testing in different lighting conditions, environments and using different cameras, however, showed large variations for these values in the captured images, either due to different lighting conditions or differences in the white balance [Viggiano 2004] performed automatically by the cameras (and, in most cases, with no "off" option). *G2gSimpleSkinSeg2* was then incremented with methods to accumulate and calculate averages and standard deviations for hue and saturation of several, arbitrary rectangular skin-colored regions. This allows an application to add a quick calibration step so the segmentation may use adequate skin hue and saturation values for the threshold operation.

Finally, after tests in an environment where the background actually has a hue very similar to the skin's, a fixed background removal operation was added as an option. Figure 1 shows a sample result of this operation. Even with a color tolerance of 50 in a 256x256x256 RGB space, about half of the pixels do not match the recorded background (not showing as black), even when this background is far enough that its actual appearance is unlikely to change due to the presence of the hand. This problem is minimized by applying a 3x3 erosion operation after the background removal, also illustrated in figure 1, but due to local corrections imposed by the camera a region around the foreground elements still shows, looking like an "aura" around the color segmented hand images in figure 1.

The system, currently, does not segment the arm from the hand, which imposes the limitation that users must wear long sleeves. This is considered a serious limitation. Even without any information about hand posture, for most of them the arm could be segmented by finding the direction of its major axis, finding the point of minimum width or abrupt change in direction along this axis (the wrist) and segmenting there [Yoon et al. 2006]. This does not work well if only a small length of arm is showing, however, or for certain postures (such as preparing a "karate chop").

Other segmentation strategies that do not require knowledge of the hand's posture were attempted, such as using color histograms and probabilities instead of the simple average and deviation, as well as the use contour information, but so far showed little improvement and more computational cost.

The first step of the analysis activity, implemented in the *G2gSCMAnalysis* class, is to find the connected components in the segmented image. The system does not assume that the background is fixed or that there are no other skin colored regions in the image, but it does presume that the player using gestures is the

closest person to the camera, so it can assume that the three largest connected components correspond to the user's hands and face. There is also a minimum number of pixels for a connected component to be accepted as a region of interest. If only 2 components above this minimum size are found, the system assumes that the missing component corresponds to the user's non-dominant hand and if only one is present, it is assumed to be the head (the head was cropped from figures 1 and 3). To further simplify the identification of the hands and head, this implementation assumes that the left hand is the leftmost region with the head in the middle and the right hand to the right. While this certainly limits user movements and the number of possible gestures, it was considered a valid limitation in this case and, during informal testing, was accepted with no complaint from the users, who obeyed it most of the time even when not informed of it. This first step also reduces noise left after the segmentation and eliminates from the analysis other people who might wander in the background.

Analysis and recognition of the gestures themselves adopt a divide and conquer strategy [Wu & Huang 1999], separating the recognition of hand posture and hand movements. Postures are recognized through estimation by synthesis (ES), i.e. the real hand's image is compared with images synthesized from a 3D hand model so that 3D posture information (the parameters used to model the hand's posture) is obtained comparing only 2D images, instead of trying to match a 3D model to the real image, which can be accurate but computationally expensive and complicated by the presence of postures with self occlusion [Imai et al. 2004]. Unlike most applications of ES methods, however, here it is not necessary to determine hand posture continuously and differentiate between postures with only small differences. Because tolerance of variation in postures is one of the system's requirements, it is not only acceptable but necessary that small differences in posture be disregarded. This implementation, therefore, may sidestep one of the most serious complication of ES methods. It only needs to compare the real hand image with a small number of possible postures, instead of thousands of possibilities. When no acceptable match is found, the system simply assumes the user is not performing a command gesture.

As in other ES methods [Shimada et al. 2001, Imai et al. 2004], the features *G2gSCMAnalysis* provides are based on the hand's 2D contour. The most important feature is a vector of the distances between the hand's centroid and a fixed number of points on the contour. These points are shown in figure 1. This vector is normalized in the analysis, so the maximum distance always corresponds to the same value and the features are scale-invariant, reducing the influence of the distance between the hand and the camera. All features for the vocabulary of possible, modeled postures are pre-calculated so only those for the real hand need to be determined in each execution step. Currently the number of points sampled from the contour in the feature vectors is, somewhat arbitrarily, set at 128. This number has shown to be small enough to allow fast computation and large enough that it is not necessary to worry about choosing points near remarkable contour features (usually local maxima and minima corresponding to tips and bases of fingers).

G2gSCMRecognition implements both posture and movement recognition. Posture recognition consists simply of comparing the feature vector obtained from the real hand's captured image with each vector for all the possible postures and finding the posture that minimizes the mean squared error between these two vectors. If the minimum error is still larger than a tolerance value, no posture is recognized (recognition returns a "not found" constant).

Unlike other ES implementations, however, the observed vector is not made rotation-invariant during recognition (by rotating it during each comparison so extremal points coincide with the model). While some tolerance in posture recognition is desired, rotation-invariance is not. Should this operation prove necessary, to avoid incorrect results due to the accumulation of many small errors caused by a small rotation, it could still be implemented while leaving the algorithm sensitive to rotation because recognition uses yet another feature: the angle between the highest point in the contour and the centroid. This feature, also provided by *G2gSCMAnalysis*, is currently used to speed up recognition by discarding, before the calculation of the mean squared error, any posture with an angle that differs by more than a certain tolerance from the one in the observed image. The highest point (usually a fingertip) is easy to determine because the contour-finding algorithm is implemented in a way to always find this point first. This angle could also be used to account for hand rotation if the vector of distances was made rotation-invariant, but tests so far have not shown the need for this operation.

The analysis also provides the centroid's absolute location in the image and its area (or number of pixels), which are used for movement recognition. Only 12 movements are recognized: left, right, up, down, back, forward, 4 diagonals, clockwise and counter-clockwise approximate rotations. The movement is temporally segmented by the gesture's initial and final postures, so it can be identified as one of these possibilities by a simple set of conditions, similar to a two stage scheme described in the literature [Mammen et al. 2001]. For the back and forward movements, the initial and final posture of the hand must be the same, since this movement is estimated by the variation in area.

In the current implementation, a gesture may be defined by movements and initial relative locations of both hands, but only postures of the dominant one (currently the right hand, but the next version will allow choosing left or right) are identified. There are now 41 postures available. Adding more postures is

quite simple and others were considered and could have been added, but they were either meaningless, quite hard to perform or had the same contour in a 2D image. With this number of available postures and movements, and remembering that a gesture might consist of one or two postures, or a movement bound by two postures that may be different (except when moving back or forward), there are almost 20,000 available gestures for the dominant hand alone, even before considering its location relative to the head or the movement of the other hand.

Finally, *Desc2Input*'s implementation in the current version, for MS Windows only, has only two public methods: *associate* and *sendDesc*. The *associate* method receives a description (a string, representing a gesture or any other event, such as stepping on a dancing mat's "button") and the system input (key press, mouse move or click) and parameters (such as key or position) associated to that description. The *sendDesc* method only receives a description and indicates that *Desc2Input* must generate the associated input (which is broadcast to all windows). A priority for future versions is making this module easier to use, adding alternatives that require little programming (leaving the association of gestures and commands to an external configuration file, for instance).

5. Tests and Results

Four prototype applications were created to test the system in different conditions. The first priority was to verify the posture analysis and recognition strategy, independent of segmentation. To accomplish that, 120 already segmented pictures of hands in different postures were stored and ran through the analysis and recognition modules. These images were segmented using the same algorithm described before but were chosen manually at moments when it worked adequately (as in the examples shown in figure 3).

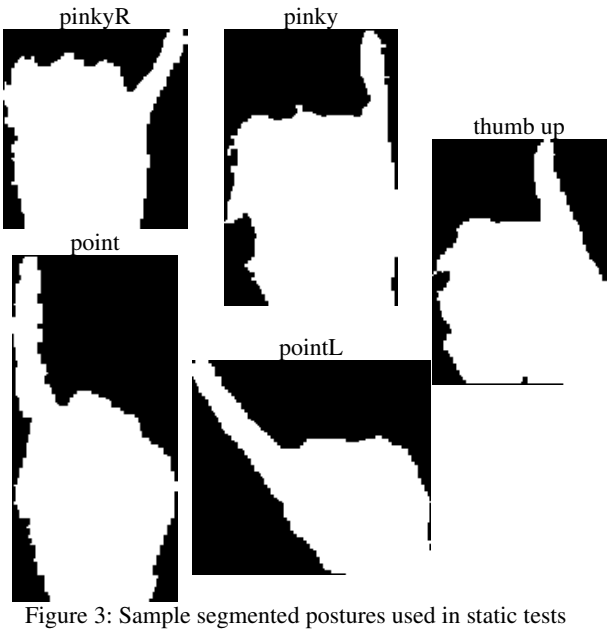


Figure 3: Sample segmented postures used in static tests

To allow the comparison of every posture with every other one, the angle difference between the highest point in each posture was discarded and the mean square error between the distance vectors was recorded. Table 1 shows the results, truncated to the nearest decimal, of one such test, comparing 15 postures. More postures are not shown due to the limited space. This particular test was chosen specifically because it contains similar postures that show problematic results.

In all cases the correct posture was identified (i.e. had the minimum error), as shown by the values with a gray background in table 1. In 8 cases, however, incorrect postures showed a low error as well (shown in bold on white). The system considers error values below 1 as possible matches. So, if "pinkyR" had not been one of the possible postures, for instance, "pinky" would have been accepted by the system as "pinkyR". Figure 3 shows these problematic postures. Two of these cases (pinky and pinkyR, point and pointL) are postures where a single finger is raised and that differ from each other by this finger's angle. Using the angle of the highest point as a feature eliminates these incorrect matches. The other mismatch that might have occurred is between the postures with the pinky up and the thumb up posture, but as seen in figure 3, these postures are actually quite similar. In all these static tests, all postures were recognized correctly but a few similar ones showed possible mismatches. In the test illustrated by table 1, for instance, only 8 comparisons in 225 were possible mismatches, approximately 3.5%.

Table 1: Sample static posture comparison

two	tu	three	pt	pointL	point	pinky	pinkyR	palm	open	openL	hm	hl	fist	claw	
15.9	9.7	13.3	13.5	15.6	16.1	11.1	9.2	9.4	7.4	5.5	14.1	9.7	5.7	**0.8**	claw
14.5	10.2	10.4	16.1	18.0	19.2	11.3	10.3	8.6	9.8	10.1	16.2	12.1	**0.8**	4.3	fist
8.6	2.3	7.3	3.6	3.3	2.9	3.4	3.4	3.2	6.6	9.8	3.3	**0.9**	13.6	12.5	hl
4.1	1.3	9.9	1.4	2.8	3.0	1.8	2.6	4.0	9.3	10.1	**0.4**	2.2	16.8	17.3	hm
10.4	6.4	10.3	8.8	11.4	11.9	7.0	6.0	9.4	3.4	**0.3**	9.8	7.7	11.5	6.1	openL
12.2	5.2	8.5	9.1	9.5	10.0	6.7	6.6	8.6	**0.4**	2.2	8.1	5.6	12.6	8.6	open
7.2	2.6	9.0	4.0	3.8	4.4	2.6	3.3	**0.1**	8.7	9.8	4.6	3.9	7.8	10.5	palm
2.6	**0.4**	5.2	1.7	3.0	3.4	**0.3**	**0.5**	2.5	6.4	7.0	1.8	2.1	10.6	11.1	pinkyR
3.8	**0.5**	5.3	2.3	2.6	3.0	**0.3**	**0.7**	1.9	6.6	7.9	2.2	2.1	10.9	11.8	pinky
7.5	2.8	9.4	1.8	**0.3**	**0.1**	2.3	3.4	3.8	9.7	11.2	2.8	2.9	18.8	17.3	point
7.8	3.0	9.5	2.0	**0.1**	**0.2**	2.5	3.8	3.6	10.1	11.9	3.1	3.4	18.5	17.6	pointL
2.8	1.5	8.2	**0.2**	2.2	2.3	1.4	1.3	3.4	9.6	9.1	1.8	2.7	15.2	14.7	pt
7.1	4.9	**0.4**	7.4	7.9	8.4	5.2	5.3	8.0	7.5	9.7	9.1	7.3	13.7	15.2	three
3.8	**0.2**	5.0	2.3	3.1	3.4	**0.6**	**0.9**	2.5	5.4	7.2	1.6	1.4	10.6	11.1	tu
0.6	3.8	9.4	2.9	8.3	8.1	4.0	3.0	8.5	12.7	11.0	4.5	5.8	17.7	18.0	two

A second test application shows identified postures in real time and allows the verification of the effects of the segmentation. It requires a few seconds for setup,

showing a region on the screen that the user must "cover" with a region of skin so initial averages and deviations for skin color can be determined. While the application allows this to be done several times (to capture, for instance, the colors of the palm and back of the hand as well as face regions), showing either many or any single region of skin have always given similar results during tests. The application also includes the options of recording and removing a known background and can either show a color image of the foreground or a monochrome image of the segmented skin regions. While showing the monochrome image, if a posture is identified the application also displays its description on the bottom of the screen. This application also identifies and displays the 8 possible movements. Actually, a gesture was defined for each movement, all 8 having as both initial and final posture a closed fist (which is very accurately identified by the system). The images labeled as "Erosion" and "Posture" in figure 1 are actually regions from screenshots of this application.

During the tests with this application, analysis and recognition continued to perform well when the images were well segmented. Often, however, a finger, usually the thumb or pinky, would disappear from the segmented image or only parts of the fingers would show, leading to postures not being recognized or for mismatches (such as an open palm identified as mimicking a claw). This was mostly due to problems with the illumination and image capture, such as a bloom showing between the fingers if the open hand was in front of a light source or bright light sources reflecting specularly from large regions of skin. Both make large skin regions show as white. Even in these environments with no controlled (and problematic) illumination, the system identified the right posture most of the time. Another problem that occurred during these tests happened when the long sleeves worn by the subjects slid down the wrist, showing a portion of the forearm. Only 2 or 3 centimeters needed to show to cause a dramatic drop in the recognition's quality. During these tests, the movements were always recognized correctly.

While Gestures2Go should be primarily used to issue commands with gestures, a third application was built to evaluate its use to select objects, replacing the use of the mouse. A posture was associated with moving the mouse and relative changes in hand position while that posture was recognized were mapped to relative positions in the mouse pointer using *Desc2Input*. Two other postures were associated with left and right clicks. The hand moved only in a small region of a 640x480 image while the mouse should move over a 1024x768 region, so the linear mapping between movements increased the hand's vertical and horizontal movements by different constants to apply it to the mouse. The system was still relatively easy to use even to click on smaller objects on the screen.

Finally, postures, movements, using the hand to move the mouse pointer and click and the use of a dancing mat for navigation were put together in a fourth test application which was used to control a popular MMO. Using the hand to move the mouse pointer and clicking was only necessary to manipulate some objects in the scenery. A gesture was associated with the command to select the next possible target and several gestures were associated with different actions to be performed on this target. This interface was especially adequate to this particular MMO because most actions are accompanied by easily identifiable hand motions of the player's avatar, so the mapping between gesture and game action was natural, very visible and enjoyable. To navigate in the game world using the dancing mat, it was connected to the computer's parallel port and a class was created to read its inputs and send them to Desc2Input to be translated as the arrow keys and commands for actions such as jumping. Because in systems derived from Windows NT only applications running in kernel mode can access the parallel port, it was necessary to either write a device driver or use an existing one. Using Inpout32 [logix4u 2009] was the chosen solution. It is a DLL with an embedded driver and functions for reading and writing to the parallel port (inp32 and out32). Up to the time of this writing, unfortunately, permission to use this MMO's name and images had not yet been granted by the publisher.

The performance of each module was also tested, using a 3GHz Intel Core 2 Duo CPU and 2GB of RAM (the test process ran in only one core, however). Table 2 shows approximate average times measured for each task in 375 tests (5 tests of 5s at 15 frames per second).

Table 2: Performance

Activity		**Time (ms)**
Segmentation		13.600
Analysis	Components	0.650
	Moments	0.013
	Features	0.003
Recognition	10 Postures	0.002
	Movement	<0.001

Table 2 shows how segmentation is by far the most costly activity. During analysis, finding the connected components is also the most time consuming task, but still only takes less than a millisecond. Finding the image moments for one hand's connected component takes approximately 13µs only because OpenCV's function calculates up to third order moments, while the system only requires moments of orders 0 and 1, so this operation could be easily sped up, but it is clearly not a priority. Calculating all features needed for recognition and the recognition itself were extremely fast during these tests, at less than 5µs. That's assuming there are 10 possible postures (recognition time increases linearly with possible postures) and a worst case scenario where the angle difference is never above tolerance, so the mean square error between distance vectors is calculated for every possibility. Movement

recognition consists of only a few conditions and happened too fast to get accurate measurements. With these results, the system satisfies the requirement of low processing demand and should it be necessary to make it faster, it is trivial to parallelize the segmentation, either to run in more cores or to be done in the GPU. These processing times, however, indicate that finding a more robust segmentation strategy is much more important than increasing its performance.

6. Conclusion

This current implementation of Gestures2Go, focused specifically on games and other similar applications, satisfies most of the requirements for gesture-based interfaces and games which were studied during the system's design phase.

While there is need of some setup, to record the background and calculate the player's skin color parameters, this setup only takes a few seconds. Each execution step takes less than 15ms in a single 3GHz core, satisfying the requirements for low processing demand, especially considering that in most contexts the system must only differentiate between 5 to 10 gestures. However, combining 41 (or more) postures of one hand and 12 movements and initial hand locations (relative to the head) for both hands creates a vocabulary of thousands of possible gestures, greatly increasing the chance that the interface designer can find an appropriate gesture to associate with an action. Desc2Input facilitates multimodal interaction and the system as a whole is quite tolerant to variations in gesture execution, both for postures and movements.

One requirement cannot be considered satisfied yet, however: simplifying the development of games with gestures. *Desc2Input* should be responsible for this requirement, but currently its interface only allows the association of descriptions and inputs by hard coding them using the associate function. Furthermore, its current version is provided as source code that must be included within the same project as the gesture recognition system and systems for interpreting other modes of interaction (such as the dancing mat used in one of the tests, or speech recognition). This makes the system's use by programmers much more complex than desired. It is a priority for future works, therefore, to develop a better interface for *Desc2Input*. The next system's version will allow the association of descriptions and inputs though an external xml configuration file and *Desc2Input* will be available not only as source code but as a DLL to include in projects as well as a standalone executable that receives descriptions via sockets from different modules responsible for complementary interaction modes. Gestures2Go will also include a standalone application that generates regular system inputs from command gestures so that this sort of interface may be used with any other interactive application simply customizing a configuration file associating gestures to inputs, without requiring a single line of programming. Another standalone application is in development to facilitate this configuration: instead of editing the configuration file directly, the user simply shows initial and final posture to the system and selects, in a graphical interface, movements, locations and which input that gesture must generate. A final improvement in this area is the integration of Gestures2Go with a game engine, but this depends on the engine's architecture and is beyond this paper's scope.

Another priority for future works is improving the segmentation. One of the system's requirements is that it must not demand controlled or special lighting or unusual or expensive equipment and, under those severe limitations, the segmentation actually works considerably well. But it is still the less robust part of the system and causes frequent and noticeable errors under some lighting conditions. Several robust probabilistic solutions exist to track hands and their contours, such as using variations of the condensation algorithm [Isard & Blake 1998]. Most of these solutions require knowledge either of one fixed hand posture, or a small number of postures and a transition model between them [Liu & Jia 2004] which complicates the addition of new postures and gestures. Even these methods often use depth data to aid in segmentation. Other methods do not require a known model for the hand but only track its position, not the contour, which is necessary for Gestures2Go. One promising approach that will be tested as soon as possible within this system is tracking the hand and its contour with no hand model information by using Kalman filters to estimate both the hand's movement and the positions of control points of curves that define hand shape [de Bem & Costa 2006]. This strategy will be adopted if tests show that its performance and accuracy are adequate while tracking enough control points to model a rapidly changing hand contour.

Using depth data [Nakamura & Tori 2008] is another planned improvement to the system, both to the segmentation and to allow a greater number of postures, such as pointing postures. Lastly, formal usability tests must be conducted to determine whether the interaction techniques using Gestures2Go in a MMO are effective in the context of games.

References

AILIVE, 2007. LiveMove White Paper. Available from: http: //www.ikuni.com/papers/LiveMoveWhitePaper_en.pdf [Accessed 24 July 2009].

BERNARDES, J. ET AL, 2008. Augmented Reality Games In: *Extending Experiences: Structure, analysis and design of computer game player experience*. Lapland University Press, p. 228-246.

BERNAL-MERINO, M., 2007. Localization and the Cultural Concept of Play. Available from: http://www. gamecareerguide.com/features/454/localization_and_the_ cultural_.php [Accessed 24 July 2009].

BOWMAN, 2005. 3D User Interfaces: Theory and Practice. Addison-Wesley.

CAMURRI ET AL., 2003. Analysis of expressive gestures in human movement: the EyesWeb expressive gesture processing library. In: *Proc. XIV Colloquium on Musical Informatics*.

DE BEM, R., COSTA, A., 2006. Rastreamento de visual de múltiplos objetos utilizando uma abordagem livre de modelo. In: *Proc. XVI Congresso Brasileiro de Automática*, 2760-2765.

DOBNIK, V., 2004. Surgeons may err less by playing video games. Available from: http://www.msnbc.msn.com/id/ 4685909 [Accessed 24 July 2009].

ECKEL, B., 2003. Thinking in C++. Prentice Hall.

FLECK, M. & FORSYTH , D., 2009. Naked people Skin Filter. Available from: http://www.cs.hmc.edu/ ~fleck/naked-skin.html [Accessed 24 July 2009].

FRITSCH ET AL., 2005. The Effect of Latency and Network Limitations on MMORPGs (A Field Study of Everquest 2). In: *Proc. of NetGames '05*.

HENDERSON & BHATTI, 2003. Networked Games – a QoS-Sensitive Application for QoS-Insensitive Users? In: *Proc. ACM SIGCOMM 2003 Workshops*.

HONG, T., 2008. Shoot to Thrill. In: *Game Developer* 15(9) p. 21-28.

HOYSNIEMI ET AL., 2005. Children's Intuitive Gestures in Vision-Based Action Games. In: *Communications of the ACM* 48(1), p.44-50.

IMAI, A. ET AL., 2004. 3-D Hand Posture Recognition by Training Contour Variation. In: *Proc. Automatic Face and Gesture Recognition 2004*, p. 895-900.

ISARD, M., BLAKE, A., 1998. ICondensation: Unifying low-level and high-level tracking in a stochastic framework. In: *Proc. 5th European Conf. Computer Vision*.

KANE, B., 2005. Beyond the Gamepad panel session. Available from: http://www.gamasutra.com/features/ 20050819/kane_01.shtml [Accessed 24 July 2009].

KOLSCH, ET AL., 2004. Vision-based Interfaces for Mobility. In: *Proc. Intl. Conf. on Mobile and Ubiquitous Systems*.

LAAKSO, S., LAAKSO, M., 2006. Design of a Body-Driven Multiplayer Game System. In: *ACM CCIE* 4(4).

LIU, Y., JIA, Y., 2004. A Robust Hand Tracking and Gesture Recognition Method for Wearable Visual Interfaces and its Applications. In: *Proc. ICIG '04*.

LOGIX4U, 2009. Inpout32.dll for Windows 98/2000/NT/XP. Available from: http://logix4u.net/Legacy_Ports/Parallel_ ort/npout32.dll_for_Windows_98/2000/NT/XP.html Accessed 24 July 2009].

LYONS, H. ET AL., 2007. Gart: The gesture and activity recognition toolkit. In *Proc. HCI International 2007*.

MAMMEN, J.; CHAUDHURI, S. & AGARWAL, T. Simultaneous Tracking Of Both Hands By Estimation Of Erroneous Observations. In: *Proc. British Machine Vision Conference* 2001.

MAPES, D., MOSHEL, J.,1995. A Two Handed Interface for Object Manipulation in Virtual Environments. In: *Presence: Teleoperators and Virtual Environments* 4(4), p. 403-416.

MILLER, G., 1956. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. In: *The Psychological Review* 63, p. 81-97.

NAKAMURA, R., TORI, R., Improving Collision Detection for Real-Time Video Avatar Interaction. In: *Proc. X Symp. on Virtual and Augmented Reality*, p. 105-114.

NOVINT, 2009. Novint Falcon. Available from: http://home.novint.com/products/novint_falcon.php [Accessed 24 July 2009].

OPENCV, 2009. Available from: http://sourceforge.net/ projects/opencvlibrary/ [Accessed 24 July 2009].

PAVLOVIC ET AL., 1997. Visual Interpretation of Hand Gestures for Human Computer Interaction: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19(7), p. 677-695.

QUEK, 1994. Towards a vision-based hand gesture interface. In: *Proc. Virtual Reality Software and Technology Conference 1994*.

SHIMADA ET AL., 2001. Real-time 3-D Hand Posture Estimation based on 2-D Appearance Retrieval Using Monocular Camera. In: *Proc. Int. Workshop on RATFG-RTS*, p. 23–30.

SHNEIDERMANN, 1998. Designing the user interface: strategies for effective human-computer interaction. Addison Wesley, 3. ed.

SNIDER, M., 2009. Microsoft unveils hands-free gaming. In: *USA Today*, 1 June 2009. Available from: http://www. usatoday.com/tech/gaming/2009-06-01-hands-free-microsoft_N.htm [Accessed 24 July 2009].

STARNER ET AL., 2004. Mind-Warping. In: *Proc. ACM SIGCHI Advances in Computer Enternatainment 2004*, p. 256-259.

TSE ET AL., 2007. Multimodal Multiplayer Tabletop Gaming. In: *ACM CIE* 5(2).

VIGGIANO, J., 2004. Comparison of the accuracy of different white balancing options as quantified by their color constancy. In: *Proc. of the SPIE* 5301, p. 323-333.

WU, Y., HUANG, T., 1999. Capturing Articulated Human Hand Motion: A Divide-and-Conquer Approach. In: *Proc. IEEE Int'l Conf. Computer Vision*, p. 606-611.

YOON, T. ET AL., 2006. Image Segmentation of Human Forearms in Infrared Image. In: *Proc. 28 IEEE EMBS Annual International Conf.*, p. 2762-2765.

Development of Games for SET-TOP-BOXES with Brazilian's Middleware GINGA-NCL

Aderbal N. Silva Junior Antônio C. S. Souza Luiz C. S. Machado

Instituto Federal de Educação, Ciência e Tecnologia da Bahia, DTEE, Brazil

Figure 1: Frames illustrating the games that are being produced on this project.

Abstract

This paper aims to present the process of creating games for set-top-boxes with Brazilian's middleware GINGA embedded on it, focusing on GINGA-NCL, since game's pre-production until the end of the demonstrration (DEMO) version developed by the Research Group of Information's Technology on Digital TV of the Instituto Federal de Educação, Ciência e Tecnologia da bahia in conjunction with the Core of Computational Modeling of SENAI / CIMATEC. All games in this text have been programmed in the language Lua coupled to NCL and the tests to evaluate the gameplay were performed using the emulator Ginga-NCL Player and the Virtual Set-Top-Box (STB) Ginga-NCL. Both softwares simulate the needed equipment for televisions that do not have specific tuner for receiving digital signal.

Keywords: digital tv, ginga, set-top-box

Authors' contact:
{aderbalnunes,antoniocarlos,luizcms}
@ifba.edu.br

1. Introduction

The market of electronic games have been highlighted in the last decade on Brazilian economy, being currently responsible for handling 87.5 million reais adding specific hardware and software according to the Brazilian Association of Developers of Electronic Games (ABRAGAMES). It was also discovered in the last survey that the software industry of games in Brazil has grown about 30% per year. Worldwide's economic viability is even more visible since 2003, when the segment of video games exceeded the revenues of the film industry [ABRAGAMES 2004].

The current scenario for the deployment of Brazilian System of Digital Television (SBTVD) is an opportunity to encourage the Brazilian scientific and technological development, since the guidelines and strategies of Decree No. 4901 of November 26th, 2003 of Brazilian's government that establishes the SBTVD [Brazil 2006], are to encourage regional and local industry in the production of digital applications. Looking deeper in the guidelines and strategies of the decree establishing the SBTVD, its inclusion intends to implement three key concepts: portability, connectivity and interactivity, field in which the development of games stands out among all other types of applications. However, being a relatively new technology, the construction of more complex and dynamic applications for SBTVD is still a challenge. The middleware GINGA set for the Brazilian's digital TV system, developed by the Catholic University of Rio de Janeiro (PUC-Rio) and the Federal University of Paraíba (UFPB) has some limitations, showing that still needs to be improved, given that it does not meet all needs of interactive applications' development.

The transmission of free signal of digital TV in Brazil is on implementation's phase and for this reason the majority of Brazilian's cities are not receiving the digital signal. So the feedback from researchers and developers who work in this area and were not part of the team responsible for the creation of middleware is essential for a stronger version of GINGA to be available.

The main objective of this work is to strengthen the activities aimed at production of digital content in Bahia, including its research centers in the area of Digital TV and production of electronic games. The research and development in the Group of Research on Information Technology, Line of Digital TV are

part of the project contemplated by FAPESB, August 2007 edict, for the development of innovative solutions in the field of information and communication technology. A partnership of the Instituto Federal de Educação, Ciência e Tecnologia da Bahia, with SENAI / CIMATEC, joining a team of programmers, designers, writers, specialists in sound effects, masters and doctors in computing area for the strategic positioning of the State of Bahia in the forefront of development and improvement of the transmission of digital TV in Brazil.

2. Detailed Steps of Development

The process of developing software for Digital TV presents peculiarities compared to other architectures such as PCs, cell phones and video game consoles, although the cycle of production is based on three stages set by Schuytema [2008] with pre-and post-production well defined. Below are the detailed steps of the methodology used by the Group of Information Technology - Research Line of Digital TV from Instituto Federal de Educação, Ciência e Tecnologia da Bahia with SENAI / CIMATEC for Digital TV Games Development Project.

2.1 Analysis of Functions Provided by the Middleware

In this case, GINGA was the middleware analyzed. The analysis seeks to exploit the resources of image, sound, storage devices and input and output. Due to possible problems with patents, GINGA-Java (or Ginga-J) had no official forecast for full release when this project started and that is why the development of Java applications for digital TV was discarded. Ginga-Java adopted GEM, a software block, which is part of Multimedia Home Platform (MHP), the european standard, and OpenCable Application Platform (OCAP), the north-american standard. The GEM's adoption to make the system more compatible with the other technologies became a delay factor for GINGA-Java because GEM was patented, as it was described by B4DTV. Then, the SBTVD forum made a deal with Sun Mycrosystems to develop an open version of GEM and that is why all the work described here was designed and implemented under GINGA-NCL using its support for the programming language LUA.

2.2 Game's Specification

Some ideas of casual games were chosen to be worked remembering the platform and physical limitations such as the remote control. These limitations had to be considered because they affect directly the gameplay of certain applications, such as the use of remote controls as joysticks that prevents excessively rapid responses and a greater care in the provision of keys to be used. These specifications and settings will be the Design Document (DD) and the basis for script and analysis of games geared for TV and other platforms that have characteristics in common with the games that are part of the project. This is the first model of the DD because the process of creating games is iterative and this document is in constant improvement in each step what prevents the development team to lose focus of the project.

2.3 Tools' Definition

This step consists in study, evaluation and selection of tools that will be used. After the definition of the game, the designers specified the graphical objects to be used in the game table to table exported in 3D Studio Max (to be in 3D), Adobe Flash (for animation 2D) and Adobe Photoshop (for 2D paintings and illustrations) in order to generate the frames of each object. For programming of game's routines in languages NCL and Lua the selected tools were Lua's Composer to write the code, the emulator GINGA-NCL player to run the first tests and virtual machine Ginga-NCL Virtual STB, that simulates the environment of a STB with the middleware GINGA-NCL embedded on it.

2.4 Implementation

It is conducted around the Development Group (DG) and begins after the definition of the visual identity of the games. Usually the team of arts and design make the first characters and environments for the game, which are turned into a test environment by the programming team. Then, in parallel or in sequence, the whole interface is designed (the main screen and other screens of the game). The features' programming and phases of the game are supervised by the DG. The activities are executed in parallel, making it a cyclic and incremental process, where each cycle implements a new stage, mission or play mode. Below are described the parts of implementation, the responsible staff and their respective functions.

2.4.1 Design and Arts

The game's script and DD will subsidize the illustration and modeling of characters, objects and scenarios which is built up by the group of design and arts. This stage consists of the following sub-steps: decoupage (stage where functionalities of the particular phase are identified and documented), conceptual art, storyboard, illustration or modeling, animation and animations exporting.

2.4.2 Sound

As the design and arts team, the sound team is subsidized by the game's script in the implementation phase, so the soundtrack portrays faithfully the atmosphere suggested by each stage or mission and its peculiarities in terms of effects. The stages of production in the group are decoupage, search and

cataloging of music and sound effects, editing, composition, adjustment and composition and editing with animations.

2.4.3 Programming

Like previous teams, the group will be guided by the stage or mission's script. The activities of this group are: decoupage, model solution planning, implementation and level design - where occurs the integration of products from the previous phase placing characters, objects and the sources of sound and also adding all the programming logic required to make the correct operation of these objects.

The step of testing and adjustments close the development cycle and starts a new one, which will create a new stage or part of the game.

The first cycle of development was used to design the demonstration version of the game. This version is being used to test the usability of the game and, soon players from outside the group of research and development will be called to play it. The players' suggestions will be taken for evaluation and the most important of them will be included as a possible fix on the subsequent cycle.

2.4.4 Beta Version and Final adjustments

This test will be conducted with a group of players after the completion of the final version of the game. Level of motivation and entertainment offered by the game will be considered on this phase. Noticed problems like breaks on the flow, components that maximize the difficulty to the point of discouraging casual gamers and general unexpected errors as well as significant suggestions for improved use of the remote control will be corrected before release to consumers, thereby ending the process of developing the game.

2.4.5 Completion

This stage is the end of the last adjustment. In the completion a fully game's evaluation will be held by the whole project team. There will be discussed and evaluated the produced games and the development process adopted as well. Each DG will present their report of overall development, which contains information such as problems encountered, solutions to problems and methodologies adopted. This meeting will result in a document with the project's analysis that will synthesize the knowledge and suggestions for possible improvements in a new version or sequel for the game developed. The second and third sub-steps from this stage involve the process of distribution and monitoring of the game. As most of the steps is common to many processes of development of electronic games, here was just explored the tool's definition and its implementation which are described in the following section.

3. Definition and Implementation of Tools

As shown previously, there are some differences between the process of software development for Digital TV and the same process on common platforms. In this architecture, as well as cellular phones, emulators are used to simulate the middleware in which the application will run, making the installation of the application in the STB unnecessary, at least while you are doing test procedures of implementation.

The games developed and presented in this article were run in Emulator GINGA-NCL Player, which can be found in the Brazilian Public Portal Software for the operational systems: Windows, Linux and MAC OS. Note that the package Java Runtime Environment (JRE) is a prerequisite.

Ginga-NCL was created by PUC-Rio to provide an infrastructure for submitting applications for multimedia / hypermedia under the declarative paradigm written in language NCL. NCL provides facilities for specification of interactivity aspects, space-time synchronization between media objects, adaptability and support for multiple devices [Brazilian Public Software Portal 2008]. Unlike HTML or XML, the language NCL do not mix the definition of the document's contents with its structure, offering a noninvasive control of both the document's layout (presentation space) and the exhibition time. NCL does not define any objects of media, but only reference these objects semantically together in a multimedia presentation. As Such, as quoted in Souza [2008], the use of language NCL is not sufficient for the development of games, because these applications require a control flow at runtime. Thus, developers of PUC-Rio provided a integration with LUA language (LUA-NCL).

LUA is a light but really powerful programming language, which was designed to extend applications. It combines simple syntax for procedural programming with powerful constructs for data's description based on associative tables and extended semantics. It thus allows the game designer to have some control over the language without having the need to follow the huge learning curve, usually associated with programming, as said by Schuytema [2008]. As said by Celes [2004], Lua is a scripting language, designed to provide meta-mechanisms that enable the construction of more specific mechanisms. In particular, developers of the games can provide adequate abstractions for writer and artist, simplifying the tasks of these.

Lua is dynamically typed and is interpreted from bytecodes to a virtual machine (engine). It has automatic memory management with incremental garbage collection. These characteristics make it an ideal language for configuration, automation

(scripting) and rapid prototyping [Soares 2007]. This language simplicity, efficiency, portability and low impact of inclusion in applications, makes it one of the most used programming languages in the world of entertainment and can be seen today in games from LucasArts, BioWare, Microsoft, Relic Entertainment, Absolute Studios, Monkeystone Games, among others mainstreams producers as cited by Souza [2008].

4. Results

In the second step of the methodology six games were chosen to be produced. Of these six games, one has already a beta version being on implementation's phase and another two are under the demonstration version production. For all these games was adopted a line of creativity where the visual theme overlaps the simple aspect inherent to casual games. The interfaces are more dynamic and interactive, which easily draws the user's attention due to the easy assimilation.

4.1. Sudoku

Sudoku was the first produced game and there is, actually, a demonstration version with most of its features. This is a logical reasoning game where each new game usually lasts 10 to 40 minutes depending of the difficulty level and the player's experience. The puzzle starts with some initial clues and the player has to fulfill the remaining grids with numbers from 1 to 9, so that no digit is repeated in any block, row or column. Despite the numerical appearance the game does not need algebraic knowledge of the player, as said before it is a logic game.

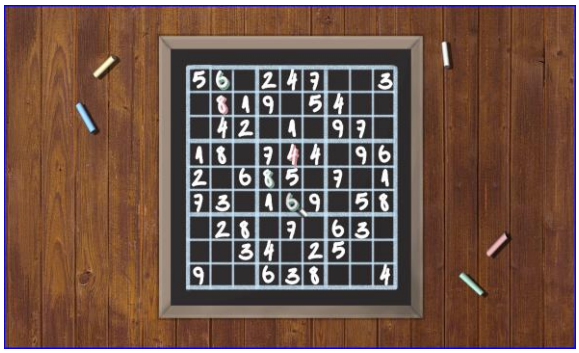


Figure 2: Sudoku's main menu.

The game was developed exploring the idea of using the colored buttons on the remote (as shown on Figure 2), the simplicity of handling, the games of chance inherent in reasoning, the limitation of using the memory emulator that simulates a platform and a design characteristic of board games as can be seen in Figure 3.

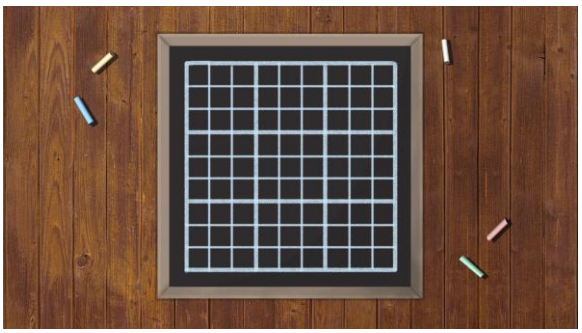


Figure 3: Sudoku's clean board.

Animated videos identified and coordinated as objects in NCL, were used to create the animated menu. The start of the presentation, the loop of animations, the interaction between the player and the screen through colored buttons and the script that describes the game itself (identified as an object of type LUA) were are referenced in NCL. The main code written in NCL, with all those references, will be run by GINGA-NCL, be it on the emulator or in the virtual machine.

Transitions are made through NCL simple connectors like "onEndStart", "onBeginStart" or "onKeySelectionStartStopAbort" that executes literally their descriptions. The trick to the flow dynamism in the NCL controlled part is on the production, merge and timing of the videos and the player's control.

Inside the LUA object called through the main code are the chunks (lines or blocks of code / commands) that describe the paths to load the entire graphical interface within the game and the variables that store the objects (chalk, board, numbers) its original position and size, in addition to the functions that govern the movement, appearance, disappearance and replacement of what is shown on screen, win or lose and life systems.

The win or lose system for board games of this type can be done through a finder function like:

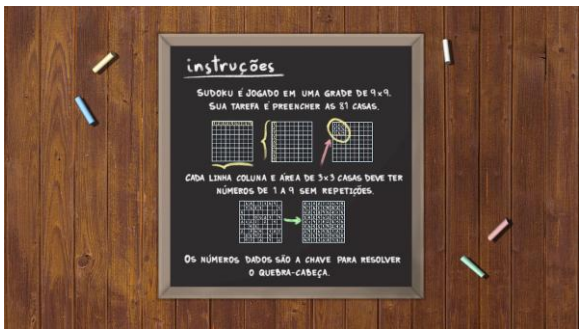
```
function whereami()
x  = math.floor((cursor.x-PIH)/EHQ) + 1
y  = math.floor((cursor.y-PIV)/EVQ) + 1
return y*TQV+(x-TQH)
end
```

Where PIH / PIV represents the initial position of the cursor in the horizontal / vertical, EHQ / EVQ are the spaces that the cursor has to walk in order to reach the next grid in horizontal / vertical and TQH / TQV is the total number of grids in horizontal / vertical. Combined with a generic function to fill the board (and also the array with the new information to be compared with the solution array) the code will be like:

```
function changeimage(target, num)

if target.fix == false and
tonumber(num) ==
rightgame[whereami()]then
target.img =
canvas:new('/imagens/'..num..'_ok.png')
target.num = num

if gameimplayn == rightgame then
event.post('out', { class='ncl',
type='presentation', area='fim',
transition='starts' })
IGNORE = true
end

elseif target.fix == false and num ~=
rightgame[whereami()] then
target.img =
canvas:new('/imagens/'..num..'_no.png')
target.num = num

playerslife= playerslife-1
if playerslifer== 0 then
event.post('out', { class='ncl',
type='presentation', area='fim',
transition='starts' })
IGNORE = true
end
else return end
end
```

As can be seen on this block, there is already attached to the function the change of images, the lose system and the correction with its penalties (reduction of player's life). Also the notification that determines the end of the game, consequently the end of LUA's script, and the return to NCL control which will be verified by the IGNORE on the handler of events that controls the input in the program. Note that the "var = var – 1" works on LUA. You just need to make sure that "var" is not NIL (null value).

Board games for GINGA-NCL as the Sudoku can use the same principle, applied here, where tables (arrays) with pre-defined positions according to their scenario can load random matches each time a player starts a new game. The loading of random files where arrays with different games are could be done through the "require" function. The possibility of loading different games from chunks of the source code allows the possibility to download new and different level games. In this game, the opening animation was made in 3D and all the gameplay (screen background, and cursor numbers) was created with 2D digital paintings and illustrations, including the sub-menus (Figure 4).



.Figure 4: Sudoku's sub-menu.

4.2. Wing Force

Wing Force was based on old shooters of vertical progression where players control an aircraft seen from above, such as River Raid of the extinct console ATARI 2600. The player must destroy enemy aircrafts, tanks, ships and interactive elements of the scenario as he travels through it collecting power-ups (improvements to the machinery of the aircraft) and keeping himself alive until the end of each stage and thus achieve the next level.

The source code of this game is severely more complex than sudoku's code, since you need to control a time flow with constant handling of the scenario, ship, enemies, etc. that should be controlled by co-routines (threads). The problem is that despite the support of co-routines by the programming language Lua, the virtual machine that simulates the STB with the Ginga-NCL does not offer this support installed. The algorithm is improved using the movement of variables to control the appearance and movement of enemies.

Today the enemies on the screen tend to bump against the player at a lower speed than the player's ship what gives the player enough time to evade. Right now the enemy's ships mechanism is being studied for better handling, be it random or fixed, to further minimize the problem of GINGA-NCL lacking of co-routines.

Some objects can explode in case of collisions. The collisions can be done by a comparator function of position in the handler of events.

Figure 4: Sudoku's sub-menu.

The scenarios are 2D paintings and enemy ships and tanks as other scenario's elements were modeled and animated in 3D and then exported frame to frame. The animations were completed in Adobe Flash, retouching the texture in Adobe Photoshop. The soundtrack is very fast, with typical effects of 8-bit consoles and a melody-based on 80s rock.

5. Related Work

Although the SBTVD's game development is something new, all the analysis about limits and possibilities of GINGA made by Souza et al.[2008] was a essential part of this work, as much as Piccolo et al.[2008] study about STBs and Digital TV architecture is particularly recommended for understanding of the operation of STBs and how

Development of games in Lua is perfectly describe in the whole Celes and Ierusalimsch work as much as development of applications in NCL is described in Soares at al.[2007] work.

6. Conclusions

The difficulty of developing more robust games for GINGA-NCL is visibly noticeable. Not just for the question of co-routines, but also by the limitation of the size of imported images and scenarios. By the close time of this article, the latest virtual machine that simulates the STB (GINGA LIVE CD V1.0) still does not support co-routines. The concept that most stood out in the proposed deployment of SBTVD, interactivity, is still not satisfactory because applications that are being produced and disseminated are still quite poor in terms of interaction between the user and the TV. While these application hopes to astonish users for their visual appeal they still provide little immersion an important conception in games.

Although the games presented here are almost finished, this work can still be seen as an initial step in developing applications for Brazilian's digital TV as there is still much to explore with GINGA-JAVA especially on development of applications aimed at interactivity and entertainment. It is presumable now that with the GINGA-JAVA's confirmation a new way to the development of games for STBs with Brazilian's middleware fostering interactivity, and conducting a more robust version of GINGA-NCL with more opportunities and less restrictions.

Acknowledgements

The authors would like to thank Fundação de Amparo à Pesquisa do Estado da Bahia and Ifba's CTPGP for the financial support and for believing in this project. Also it is important to mention B4DTV blog and DEVDTV discussion list the best source for those who want to develop absolutely anything for GINGA or just to keep in touch with everything about Digital TV on Brazil.

References

ABRAGAMES, 2004. Plano Diretor de Promoção da Indústria de Desenvolvimento de Jogos Eletrônicos no Brasil – Diretrizes Básicas, http://www.abragames.org/docs/pd_diretrizesbasicas.pdf, [Accessed 17 July 2009].

B4DTV – Blog for Digital Tv. http://b4dtv.blogspot.com/[Accessed 15 August 2009].

BRASIL. Decreto n 5.820, de 29 de Junho de 2006. Implantação do Sistema Brasileiro de Televisão Digital Terrestre - SBTVD-T. DOU de 27/11/2006. http://www.planalto.gov.br/ccivil_03/_Ato2004-2006/2006/Decreto/D5820.htm, [Accessed 20 June 2008].

Celes, W., Figueiredo, L. H. e Ierusalimschy, R., 2004. "A Linguagem Lua e suas Aplicações em Jogos", http://www.tecgraf.puc-rio.br/~lhf/ftp/doc/wjogos04.pdf, [Accessed 15 May 2008].

Openginga, 2008 ProjetoOpenGING. http://www.openginga.org, [Accessed 15 July 2008].

Piccolo, L. S. G., Melo, A. M., Baranauskas, M. C. C.[2008]. "Accessibility and Interactive TV: Design Recommendations for the Brazilian Scenario", Human Computer Interaction INTERACT 2007 11th IFIP TC 13.

Piccolo, L. S. G., "Arquitetura do Set-top Box para TV Digital Interativa". http://www.grupos.com.br/group/designcefet20051/Messages.html?action=download&year=08&month=5&id=121077537895923&attach=arquitetura%20conversor.pdf, [Accessed 15 May 2008]

Soares, L. F. G., Rodrigues, R. F., Moreno, M. F.,2007. "Ginga-NCL: the Declarative Environment of the Brazilian Digital TV System". Journal of the Brazilian Computer Society, v. 12, p. 37-46, 2007.

Souza, A. C., Machado, L. C. S., Sampaio, R. L. e Raimundo, P. O. , 2008. "Desenvolvimento de Jogos para TV Digital", 3° Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica.

Souza, A. C., Machado, L. C. S., Sampaio, R. L. e Raimundo, P. O. , 2008. "TV Digital: Limites e Possibilidades Tecnológicas", 3° Congresso de Pesquisa e Inovação da Rede Norte Nordeste de Educação Tecnológica.

Schuytema, P. , 2008. Design de games: Uma abordagem prática, Brasil, Ed. Pioneira.

Ginga Game: A Framework for Game Development for the Interactive Digital Television

Diego Cordeiro Barboza Esteban Walter Gonzales Clua

Universidade Federal Fluminense, Instituto de Computação - Media Lab, Brasil

Abstract

With the implantation of Brazilian's Digital Television System, a new software development platform has been created. Applications for the Digital TV are an important part of this new system, which aims, in addition to higher image and sound quality, the creation of an interactivity channel for the viewer. Among all possible applications for this new environment are the digital games, which every year attract a growing audience worldwide. However, game development isn't a simple task, and doing so in a limited platform such as digital receivers could be a complicated process. So, this paper presents a framework for game development for the Digital TV, which allows the developer to focus on content creation only, without concerns about technical issues or common tasks related to game development.

Keywords: Ginga, Digital TV, frameworks, games

Authors' contact:
dbarboza@ic.uff.br, esteban@ic.uff.br

1. Introduction

Brazilian Digital Terrestrial Television System (SBTVD-T) has three guidelines, in addition to its current analogue system: high-definition digital broadcast (HDTV); digital broadcast for fixed, mobile and portable reception; and interactivity [Brasil 2006].

SBTVD-T's interactivity channel allows the system to be expanded through applications built over a reference standard system. The main idea is to allow the digital receiver (set-top box) to run different applications, such as electronic guides, shopping channels, bank and educational services, among others [Barbosa and Soares 2008].

Digital games are an interactive application type that could help Digital TV become popular in the country, since the national industry is in an expansion and growing moment [Ferreira and Souza 2009].

In SBTVD-T, the *Ginga* middleware takes place between applications and execution infrastructure (hardware and operating system) [Ginga 2009]. For this reason, applications for the Brazilian Digital TV must be based on the *Ginga* middleware, using one of its supported programming languages.

This paper's main goal is to present a game development framework for the SBTVD-T, using *Ginga-J* (the procedural part of *Ginga* middleware that uses the Java language). The framework presents an application model and a set of classes that simplifies game development for the Digital TV as well as abstracts the process from a specific platform.

The idea is to use the framework to facilitate the game development for the Digital TV to the process of developing games to personal computers, except for some certain limitations imposed by the platform [ABNT 2008], such as hardware limitations concerning memory and processing capacity, and input issues related to the use of a remote control instead of mouse and keyboard.

The paper is organized as follows: section 2 presents some related publications to this paper's proposal. Section 3 presents the *Ginga* middleware and its structure. Section 4 describes the *Ginga* Game, the framework proposed in this paper, and also presents an example of its application. The last section presents the paper's conclusions.

2. Related Work

There is some related work about game development for the Digital TV. Among these papers, the following could be highlighted: [Ferreira and Souza 2009], that presents a different approach to *Ginga Game*, but with similar purpose; [Lima 2007], that develops a communication protocol for network games within SBTVD-T; and [Junior *et al* 2009], that doesn't use *Ginga-J*, but makes an interesting study about game development for the Digital TV using *Ginga-NCL* with the *Lua* programming language.

In its master degree dissertation, [Valente 2005] presents the study about the development of a framework for computer games. This work isn't directly related to game development for the Digital TV, but has some interesting content about game development frameworks architectures, which some of them are used in this work.

3. *Ginga* Middleware

Ginga is the middleware for running applications on the SBTVD-T. This platform was developed together by the research laboratories *Telemídia* [Telemídia 2009] and LAViD [LAViD], from PUC-Rio and UFPB.

The *Ginga* middleware is subdivided into two applications environments for Digital TV receivers and allows the application development following two distinct programming paradigms: the declarative (*Ginga-NCL*) and the procedural (*Ginga-J*).

The declarative environment allows the programmer to define a set of tasks to be executed without concerning who will perform these tasks and how they will be performed. This way, it is needed only the description of the desired results in a declarative language, instead of an algorithm [Barbosa and Soares 2008]. The SBTVD-T employs the NCL (Nested Context Language) language in its declarative environment [ABNT 2007] together with the Lua language, for non-declarative applications.

The non-declarative environment, or the procedural environment, requires the specification of each step to be performed by the program. In this kind of environment, the programmer has higher level of control over the application and its execution flow, but it's also required a higher language and algorithm knowledge [Barbosa and Soares 2008]. The Java language is employed in the SBTVD-T's procedural environment.

Figure 1 exposes *Ginga* middleware's architecture. The Presenting is the subsystem responsible for processing NCL documents, and the Execution Machine is in charge of processing procedural applications, i.e., Java *Xlets* [ABNT and CEET-00:001.85 2008].

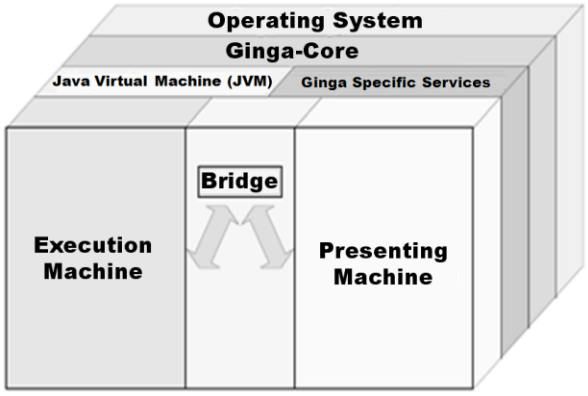


Figure 1 – *Ginga* middleware architecture **[**ABNT and CEET-00:001.85 2008].

This paper's scope doesn't enclose the *Ginga's* declarative environment, focusing solely on *Ginga-J*. For further information on application development using *Ginga-NCL*, it is recommended [Barbosa and Soares 2008].

3.1 *Ginga-J*

Ginga-J, the procedural environment for the *Ginga* middleware, is currently under development and doesn't have an official implementation. On may/2008, a draft version, without normative value, of *Ginga-J* specification was released by ABTN (*Associação Brasileira de Normas Técnicas*) and CEET-00:001.85 (*Comissão de Estudo Especial Temporária de Televisão Digital*) [ABNT and CEET-00:001.85 2008]. This specification defines *Ginga-J*'s architecture and execution environment, and is addressed to applications and digital receivers developers.

Ginga-J's architecture is shown on **Figure** 2. In this architecture, user's applications (called *Xlets*) are placed on the top level and must make use of *Ginga-J*'s standard API (Application Programming Interface). Resident Applications, on the other hand, may use non-standard system resources, available from the operating system or a particular implementation of *Ginga*. This kind of application includes closed captions, system messages, receiver's menus, program guide, and others [ABNT and CEET-00:001.85 2008].

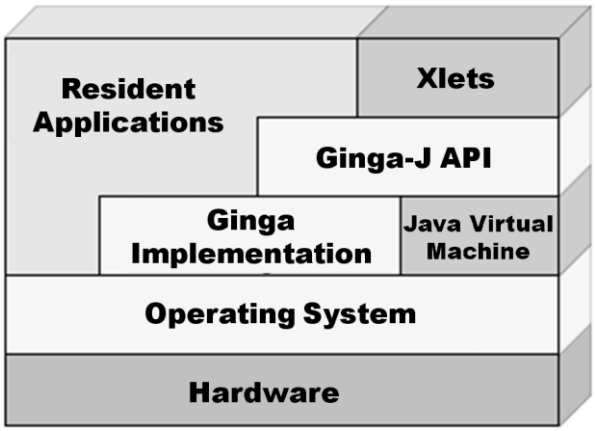


Figure 2 – *Ginga-J* middleware architecture [ABNT and CEET-00:001.85 2008].

The *Ginga-J*'s reference implementation document defines the *Ginga-J* API, a set of Java packages included in *Ginga-J*. This API includes packages from the following APIs:

- *JavaTV* [Sun 2009a]: an extension of Java platform to add support for Digital TV application development. It's main goal is to run applications abstracting the technologies used on the broadcast.;
- DAVIC [DAVIC 1998]: is a set of specifications aims to keep interoperability between platforms involved on execution of broadcasted audio and video;
- HAVi [HAVi 1999]: defines a home network interoperability standard between audio and video devices. HAVi packages' also provides resources for graphical users interface creation, extending Java's AWT 1.1 [Sun 1999];
- DVB [DVB 2009]: packages that extends features from *JavaTV*, DAVIC and HAVi,

and includes other features, such as *Xlets* communications, persistence, and others;
- *Ginga* Extensions: class set that includes channel tuning control, media flow API, and return channel API;
- ARIB STD B-23 [ARIB 2003]: API compatible with japanese Digital TV standard specifications;
- *Ginga-J* Definitions: packages that offer functions to *Ginga* middleware-included devices, such as multi-user interaction, and the bridge with *Ginga-NCL*.

The full list of these packages is available from [ABNT and CEET-00:001.85 2008].

Due issues related to copyrights costs, this reference implementation of *Ginga-J* has not been included in any digital receiver marketed in Brazil so far.

In may/2009, the Board of the Forum of Brazilian Digital Terrestrial Television System (*Conselho Deliberativo do Fórum do Sistema Brasileiro de TV Digital Terrestre*) decided [Fórum SBTVD 2009a] for the implantation of *JavaDTV* [Sun 2009b] in the *Ginga* middleware, instead of the previous published draft [ABNT and CEET-00:001.85 2008]. This API is also based on *JavaTV* and is very similar to the reference implementation, but it replaces some proprietary solutions.

The JavaDTV binaries are unavailable on the time this papers is being writted and only its documentation has been published [Fórum SBTVD 2009b].

This paper's elaboration uses the reference implementation so that it is possible to present a functional version of the project. The framework's structure has been developed in a way that platform specific details are isolated, making easier for the migration process to be done when JavaDTV becomes available.

3.2 Applications for the Digital TV

Applications for the Digital TV are called *Xlets*, just like Java applications for the web and mobile are called *Applets* and *Midlets*, respectively. An *Xlet* life-cycle is shown in Figure 3. As soon as the application is loaded to the set-top box, it will stay on the loaded state until it's started. Then it pass from the paused state to the started state (where it is actually running), and may be eventually paused and resumed again. Finally, the application manager destroys the *Xlet* when it enters the destroyed state [Burlamaqui *et al* 2008].

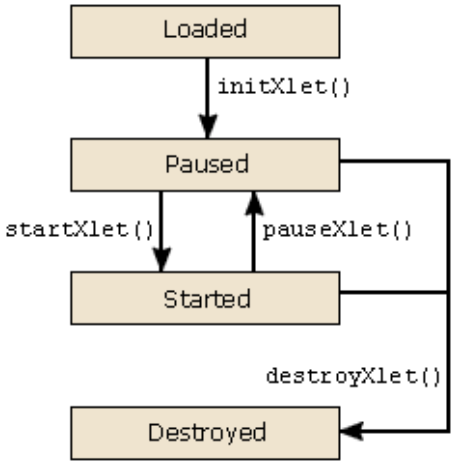


Figure 3 – The *Xlet's* life-cycle [Morris 2005].

An important difference between an *Xlet* and an *Applet* is the possibility to pause and resume an *Xlet*. This is very important in a limited environment like the digital receivers, where many applications could be running at the same time and sharing the limited available resources. This way, it's possible to temporary stop an application that is not visible and release the resources to other applications [Morris 2005].

A Digital TV application is, therefore, an application that implements the *Xlet* interface provided by *JavaTV* [Sun 2006]. The following are the public methods of this interface:

- destroyXlet: signals the *Xlet* must be finalized and enter in the destroyed state;
- initXlet: signals the *Xlet* must be initialized and enter the paused state, which means it's ready to start providing a service;
- pauseXlet: signal the *Xlet* must stop providing a service and enter it's paused state;
- startXlet: signals the *Xlet* must start providing a service and enter it's started state.

The *Xlet* interface allows an application manager to create, initialize, start, pause and destroy an *Xlet*. Due its life-cycle, it's possible that several different *Xlets* are controlled at the same time by the application manager, and the runtime environment chooses which one should be active in a given time.

For the elaboration of this paper, while the JavaDTV is unavailable, the reference implementation [ABNT and CEET-00:001.85 2008] has been used. Tests with the built applications were held with the *XleTView* [Sveden 2004], an *Xlet* emulation software that allows testing applications developed for the Digital TV [Carvalho and Araújo 2009].

4. *Ginga Game*

Ginga Game is a Digital TV game development framework proposed in this paper. Its goal is to provide a structure that makes it easier to develop games for the Digital TV and make this task more similar to the development for personal computers.

The purpose on the creation of a software framework for game development is to avoid that common tasks be implemented again every time a new game is produced. [Valente 2005] presents a similar approach, but focuses on the reuse of software components for computer game development.

Ginga Game provides an application model that automatically performs several recurring tasks concerning game development, such as resources loading and component management, and allows the developer to focus on its game specific code.

This approach is similar to the models provided by XNA [Microsoft 2009] and Unity *3D* [Unity 2009], for instance. These tools provide a complete game structure and the developer must only write the code that defines the behavior of its game components and add them to the game scenes.

Ginga Game is subdivided in three different *Java* packages. This implementation was made in a way that classes that need platform specific resources are in a separate package from the classes that doesn't have this kind of dependency. So, the migration of *Ginga* Game to another platform could be made just by doing the required modifications in only one package, while the others remain unchanged.

The package *GingaGame* provides some abstract interfaces that must be implemented in a platform specific package. In this package are defined basic concepts of the framework, such as game objects and game components, scenes, and the application model that manages these objects.

The package *GingaGame.GameComponent* has a set of ready to use components. These components must be added to objects in a game scene. Among the developed components are *AnimatedSprite* (that allows the drawing of animated images), *StaticSprite* (for drawing static images), and *BoundingBox* (for collision checking using rectangles). More components will be developed over time.

Lastly, the *GingaGameJavaTV* package encloses all platform specific classes, in a *JavaTV* specific implementation. An example of platform specific resource is the window manager. *JavaTV* uses the class *HScene* (from the HAVi package) to access the application's windows. These classes are put apart from the remaining classes of the framework to make it easy to change the execution platform, if it's needed. This way, only the platform specific code should be modified, but the interfaces remain the same.

The UML Package Diagram for *Ginga Game* is shown in Figure 4.

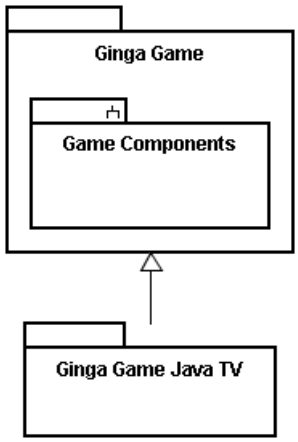


Figure 4 - Ginga Game Package Diagram**.**

4.1 How *Ginga Game works*

Games created with *Ginga Game* are made with a class that extends the framework's *Game* class. This is the main class in the framework, and it works as a starting point on the creation of new games, while it's also responsible for running the application model and the component management.

A game can be decomposed into logical units called scenes that are instances of the *Scene* class. Each scene is independent from other scenes and could be initialized and finalized at any moment by the game. It may also hold several game components and objects that are loaded and removed from the game at the same time of the scene. This division allows a simpler game organization, where each screen, stage or level can be described as a scene.

A scene or the game can hold a game object collection, where a game object is an instance of the *GameObject* class. These entities interact with each other and these interactions are what give life to the game. Characters in an adventure game or cars in a racing game, for instance, could be described as game objects. A game object may be added to a scene or the game (in this case, it will be a persistent object that will not be destroyed when the scene is finalized). A game object may contain one or more game component.

The game components are instances of the *GameComponent* class used to compose a game object. For instance, a car object can be composed by wheels, engine, lights and several other items. Each of these items can be represented as a game component.

It's up to the developer to use the components available from the framework or create its own. The *GameObject* and *GameComponent* classes can be extended so the developer can create its own content.

This approach allows software reuse at various levels. Depending on how the content was created, scenes, objects and component can be easily reused in other games. For example, a game options screen (a scene) can be shared between games that have the same user's options. Likewise, an object used to account player's scores can also be the same in several different games.

Ginga Game has classes to manage game resources and execute certain routine tasks. The *Screen* class is responsible for drawing images and text into the game screen, while the *ContentManager* class is employed to load and manage resources, such as images and fonts.

User interface is made through the *Input* class that must query the remote control keys' state e provide this information to the game components.

A collision manager verifies if a collision has occurred between solid objects in a scene and notifies the objects involved in the collision. This way, a solid object doesn't need to query if a collision has occurred at any time, when it happens an event is triggered and the object can treat it.

In future versions, more components will be available and the common content will be more and more separated from the specific content a game may need.

A simplified UML Class Diagram for *Ginga Game* is shown in Figure 5.

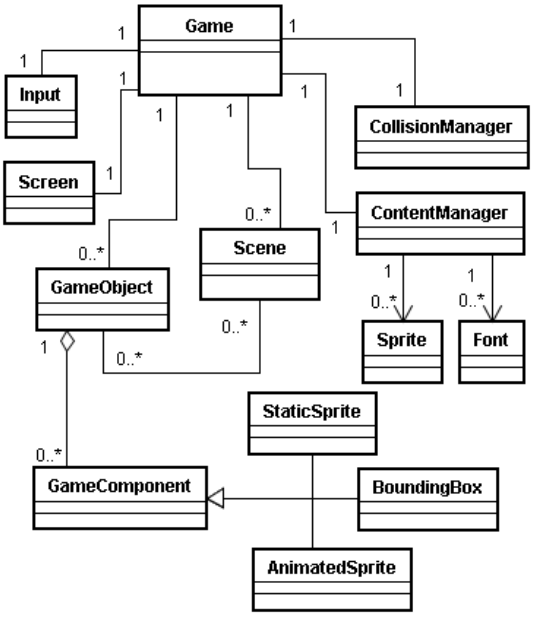


Figure 5 - Ginga Game Class Diagram.

4. Validation of the *Ginga Game*

To illustrate the development of a game with *Ginga Game*, a simple quiz game has been elaborated. While it's a very simple game, it helps to illustrate the use of the framework, since many of *Ginga Game*'s features are employed.

The game works as follows: a question and four answers are shown to the player. The answers appear inside red, green, yellow and blue buttons, using the colors of the remote control button's to make the process more intuitive. For each question the player must press the correspondent button color that he thinks is the right answer. A component accounts how many right and wrong answers the player has given.

Figure 6 shows the game's project, where it's listed its classes and images. Following is a brief description of each class:

- *Botão* (Button): a game object that represents one possible answer to the question. A button is related to an image (a *StaticSprite*) and to a text (the answer);
- *Controle* (Controller): the game object that verifies if a remote control key was pressed and tests if the right answer was chosen. The controller tells the scoreboard if the answer was correct or not and requests a new question to the game;
- *GingaGameQuiz*: the application's starting point. This class creates the *Xlet* and a game instance;
- *Jogo* (Game): an instance of the *Game* class that is responsible for initializing the game scenes, as well as the scoreboard (a global object that controls the player's score);
- *Pergunta* (Question): a game object that represents a question in-game.
- *Placar* (Scoreboard): a global game object created by the game, not a scene, that controls player's score;
- *TelaDePergunta* (Question Screen): each question screen is a scene that contains a question, a controller and four buttons. In order to add more questions to the game, it's needed just to create more instances of this class.

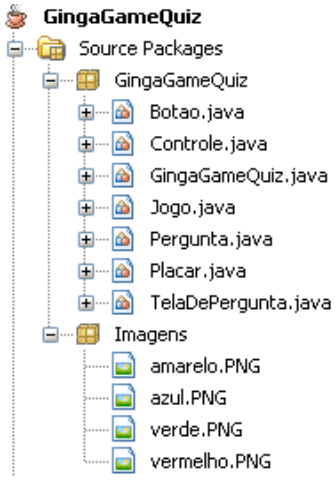


Figure 6 - Quiz Game classes.

This game exemplifies de use of *Ginga Game'*s classes in a very simplified way. In the game the game object, game component and scene concepts are used to divide the game in smaller logical units.

Figure 7 shows a game screen on the *XleTView* emulator. The area within the yellow lines are the game itself (a question on the top, the answer options on the right side using the colors of the remote control, and a scoreboard on the lower-left corner) and the remote control on the left side is provided by the emulator.

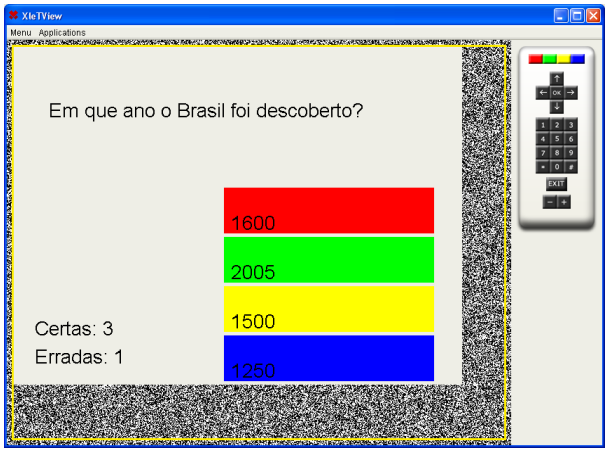


Figure 7 – Sample game developed using *Ginga Game.*

New features could be easily added. A win screen, for instance, can be created through a new scene that could be loaded after the player answers some correct questions.

5. Conclusion

The development of games for the Digital TV in Brazil, using the *Ginga* middleware, either within the procedural environment (*Ginga-J*) or the declarative environment (*Ginga-NCL*) is possible and the appeal the games have may help popularize the interactive content on SBTVD-T.

Software development auxiliary tools has great importance and the creation of frameworks that allows greater code reuse and reduces the needing to rewrite code for common tasks may help to make the process quicker and more intuitive.

With the elaboration of *Ginga Game*, it's expected to make the process of creating games for the Digital TV simpler, providing an environment that abstract the execution platform and allows the developer to focus only on the game's logic. Through its structure, *Ginga Game* proposes an environment that allows a high rate of software component reuse, reducing the creation time for new games, as previously created components can be reused in new projects.

Future versions of the framework could add current unavailable features, such as sound and video playback, as well as the integration with NCL documents.

References

ABNT - Associação Brasileira de Normas Técnicas, 2007. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 2: Gimga-NCL para receptores fixos e móveis - Linguagem de aplicação XML para codificação de aplicações. Sistema Brasileiro de TV Digital Terrestre, NBR 15606-2. Available from: http://www.dtv.org.br/download/pt-br/ABNTNBR15606_D2_2007Vc3_2008.pdf [Accessed 20 April 2009].

ABNT - Associação Brasileira de Normas Técnicas, 2008. Televisão digital terrestre – Receptores. Sistema Brasileiro de TV Digital Terrestre, NBR 15604. Available from: http://www.abnt.org.br/imagens/Normalizacao_TV_Digital/ABNTNBR15604_2007Vc_2008.pdf. [Accessed 20 April 2009].

ABNT - Associação Brasileira de Normas Técnicas and CEET-00:001.85 - Comissão de Estudo Especial Temporária de Televisão Digital, 2008. Televisão digital terrestre - Codificação de dados e especificações de transmissão para transmissão digital – Parte 4: Ginga-J - Ambiente para a execução de aplicações procedurais (VERSÃO DRAFT 05/2008). Available from: http://www.openginga.org/00_001_85_006-4abnt_port-DRAFT-05200.pdf [Accessed 23 April 2009].

ARIB, 2003. Application Execution Engine Platform for Digital Broad Casting – ARIB STD-B23. Available from: http://www.arib.or.jp/english/html/overview/doc/6-STD-B23v1_1-E1.pdf [Accessed 21 June 2009].

Barbosa, S.D.J. and Soares, L.F.G. TV digital interativa no Brasil se faz com Ginga: Fundamentos, Padrões, Autoria Declarativa e Usabilidade. In T. Kowaltowski & K. Brealtman (orgs.) Atualizações em Informática 2008. Rio de Janeiro, RJ: Editora PUC-RIO, 2008. pp.105-174.

Brasil. Decreto n 5.820, de 29 de Junho de 2006. Implantação do Sistema Brasileiro de Televisão Digital Terrestre - SBTVD-T. DOU de 27/11/2006. http://www.planalto.gov.br/ccivil_03/_Ato2004-2006/2006/Decreto/D5820.htm [Accessed 23 June 2009].

Burlamaqui, A., Silva, I. R. M. and Bezerra, D. H. D., 2008. Construção de programas Interativos para TV Digital utilizando o Ginga. Available from: http://gingarn.wikidot.com/local--files/tvdiepoca08/capituloTVDIEPOCAFinal.pdf [Accessed 23 June 2009].

Carvalho, S. R. C. and Araújo, V. T., 2009. Emuladores para TV Digital – OpenMHP e Xletview. Available from: http://www.tvdi.inf.br/upload/artigos/artigo7.pdf [Áccessed 31 March 2009].

DAVIC, 1998. Digital Audio-Visual Concil Davic 1.4. Available from: http://www.davic.org/down1.htm [Accessed 20 June 2009].

DVB, 2009. Digital Video Broadcasting – Standards & BlueBooks. Available from: http://www.dvb.org/technology/standards/ [Accessed 21 June 2009].

Ferreira, D. A. and Souza, C. T., 2009. TuGA: Um Middleware para o Suporte ao Desenvolvimento de Jogos em TV Digital Interativa. Centro Federal de Educação Tecnológica do Ceará. Available from: http://code.google.com/p/tuga-sdk/downloads/detail?name=TuGA_Middleware.Jogos.TVDigital_v1.6.pdf [Accessed 21 May 2009].

Fórum SBTVD, 2009a. Fórum do Sistema Brasileiro de Televisão Digital define o padrão para a interatividade. Available from: http://www.forumsbtvd.org.br/materias.asp?id=127 [Accessed 30 May 2009].

Fórum SBTVD, 2009b. Sun Microsystems entrega especificações Java DTV para Ginga-J sem cobrança de royalties. Available from: http://www.forumsbtvd.org.br/materias.asp?id=74 [Accessed 12 June 2009].

Ginga, 2009. Available from: http://www.ginga.org.br/ [Accessed 20 June 2009].

HAVi, 1999. Technical Background – HAVi, the a/v digital network revolution. Available from: http://www.havi.org/pdf/white.pdf [Accessed 20 June 2009].

Junior, A. N. S., Souza, A. C. S., Santos, L. C. M., Sampaio, R. L. and Raimundo, P. O., 2009. Desenvolvimento de Jogos para o Sistema Brasileiro de TV Digital, I Santa Catarina Games. Available from: http://200.169.53.89/scgames/artigos/08980100014.pdf [Accessed 18 June 2009].

LAViD, 2009. Available from: http://www.lavid.ufpb.br/ [Accessed 24 June 2009].

Lima, F. M., 2007. Protocolo de Aplicação para Jogos de Tabuleiro para Ambiente de TV Digital. Available from: http://www.midiacom.uff.br/~debora/fsmm/trab-2007-2/protocolo.pdf [Accessed 21 April 2009].

Microsoft, 2009. XNA. Available from: http://www.xna.com/ [Accessed 01 June 2009].

Morris, S., 2009. An Introduction To Xlets. Available from: http://www.mhp-interactive.org/tutorials/mhp/xlet_introduction [Accessed 30 May 2009].

Sun, 1999. The AWT in 1.0 and 1.1. Available from: http://java.sun.com/products/jdk/awt/ [Accessed 28 March 2009].

Sun, 2006. Interface Xlet. Available from: http://72.5.124.55/javame/reference/apis/jsr217/javax/microedition/xlet/Xlet.html [Accessed 30 March 2009].

Sun, 2009a. Java ME Technology – Java TV API. Available from: http://java.sun.com/javame/technology/javatv/ [Accessed 28 March 2009].

Sun, 2009b. Java(TM) DTV 1.0 Final Release. Available from: https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=javadtv-1.0-oth-JPR@CDS-CDS_Developer [Accessed 30 May 2009].

Sveden, M., 2004. xleTView. Available from: http://www.xletview.org/ [Accessed 30 May 2009].

Telemídia, 2009. Available from: http://www.telemidia.puc-rio.br/ [Accessed 24 June 2009].

Unity, 2009. Unity: Game Development Tool. Available from: http://unity3d.com/ [Accessed 01 June 2009].

Valente, L., 2005. GUFF: Um Framework para desenvolvimento de jogos, Dissertação (Mestrado) - Universidade Federal Fluminense – Instituto de Computação. Available from: http://guff.tigris.org/docs/Thesis05-pt.pdf [Áccessed 07 March 2009].

GPU Accelerated Path-planning for Multi-agents in Virtual Environments

Leonardo G. Fischer, Renato Silveira, Luciana Nedel
Institute of Informatics
Federal University of Rio Grande do Sul

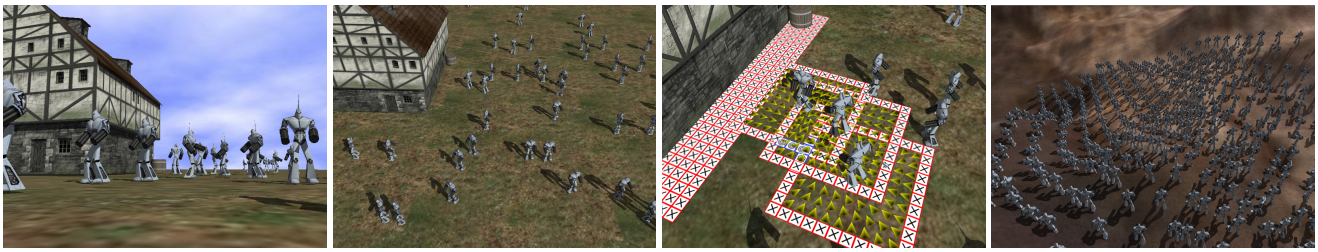


Figure 1: *Virtual characters controlled by the BVP Planner in a virtual environment.*

Abstract

Many games are populated by synthetic humanoid actors that act as autonomous agents. The animation of humanoids in real-time applications is yet a challenge if the problem involves attaining a precise location in a virtual world (path-planning), and moving realistically according to its own personality, intentions and mood (motion planning). In this paper we present a strategy to implement – using CUDA on GPU – a path planner that produces natural steering behaviors for virtual humans using a numerical solution for boundary value problems. The planner is based on the potential field formalism that allows synthetic actors to move negotiating space, avoiding collisions, and attaining goals, while producing very individual paths. The individuality of each character can be set by changing its inner field parameters leading to a broad range of possible behaviors without jeopardizing its performance. With our GPU-based strategy we achieve a speed up to 56 times the previous implementation, allowing its use in situations with a large number of autonomous characters, which is commonly found in games.

Keywords:: Path-planning, GPGPU, NVIDIA CUDA, Agent Simulation

Author's Contact:

{lgfischer,rsilveira,nedel}@inf.ufrgs.br

1 Introduction

Many types of games, specifically First Person Shooters (*FPS*) and Real Time Strategy (*RTS*) are populated by synthetic actors that should act as autonomous agents. Autonomous agents, also called *non-player characters*, are characters with the ability of playing a role into the environment with life-like and improvisational behavior. To behave in such way, the agents must act in the virtual world, perceive, react and remember their perceptions about this world, think about the effects of possible actions and finally, learn from their experience [Funge 2004]. In this complex and suitable context, navigation plays an important role [Nieuwenhuisen et al. 2007]. To move agents in a synthetic world, a semantic representation of the environment is needed, as well as the definition of the agent initial and target position (goal). Once these parameters were set, any path-planning algorithm can be used to find a trajectory to be followed.

However, in the real world, if we consider different persons (all in the same initial position) looking for achieving the same target position, each path followed will be unique. Even for the same task, the strategy used for each person to reach his/her goal depends on his/her physical constitution, personality, mood, reasoning, urgency, and so on. From this point of view, a high quality algorithm to move characters across virtual environments should generate expressive, natural and unexpected steering behaviors.

In contrast, the high performance required for real-time graphics applications compels developers to look for most efficient and less expensive methods that produce yet good and almost natural movements. To illustrate how performance is a crucial problem, it is known that to be playable, a game must run at least at a rate of 30-100 frames per second. This implies in 0.02 seconds per frame. Each frame (or step of an animation) includes the updating of the game status, handling user inputs, graphics processing, physics computations, strategic AI, path-planning, among others. Then, we can easily consider something as one millisecond per step for path-planning (with multi-core architectures, this restriction is relaxed).

Many researchers are working on methods to improve the quality of the steering behavior of synthetic agents with a minimal cost. One way to improve the performance is taking advantage of massively parallel architectures, as multi-core CPUs and GPUs (*Graphics Processing Unit*). In this work we propose a GPU implementation of the BVP Planner recently proposed by us [Dapper et al. 2007]. The BVP Planner is a method based on the numeric solution of the boundary value problem (BVP) to control the movement of pedestrians allowing the individuality of each agent.

Our main contributions in this paper are:

- A parallel version of our previously technique [Dapper et al. 2007], implemented on the GPU using nVIDIA CUDA (Compute Unified Device Architecture) [NVIDIA. 2009]
- A strategy to reduce the number of memory transactions between CPU and GPU
- Several tests showing that the GPU implementation improves up to 56 times the CPU sequential version, allowing the real-time use of this technique even in scenarios with a large number of autonomous characters

Despite *humanoid*, *autonomous agent*, and *behavior* are terms used in many different contexts, in this paper we limit its use in order to match our goals. For the sake of simplicity, we consider *humanoids* as a kind of embodied *autonomous agent* with reactive behaviors (driven by stimulus), represented by a computational model, and capable of producing physical manifestations in a virtual world. The term *behavior* will be used mainly as a synonymous of *animation* or *steering behavior* and intend to refer the improvisational and personalized action of a *humanoid*.

The remaining of this paper is structured as follows. Section 2 reviews some related works on path-planning techniques applied to virtual agents simulation. Section 3 describes the fundamentals of the path-planning method proposed by us. In Section 4 we detail the strategy used to handle the information about the environment and other agents. In Section 5 we present our strategy to implement

this technique on GPU. Section 6 shows our results, including several comparisons between the CPU and GPU version, and exposes considerations about performance. Finally, Section 7 presents our conclusions and some ideas for future works.

2 Related Work

The path-planning problem has been deeply explored in game development. The generation of a path between two known configurations in a bi-dimensional world is a well-known problem in robotics, artificial intelligence, and computer graphics field. However, to find the path is not enough when we want to endow artificial characters with natural and realistic movement similar to the ones found and followed by real human beings. When it comes to a game with many autonomous characters, for instance, these characters must also present convincing behavior. It is very difficult to produce natural behavior by using a strategy focusing on the global control of characters. On the other hand, taking into account the individuality of each character can be a costly task. As a consequence, most of the approaches proposed in computer graphics literature do not take into account the individual behavior of each agent.

An example is the technique proposed by Kuffner [James J. Kuffner 1998]. Kuffner proposed a technique where the scenario is mapped onto a 2D mesh and the path is computed using a dynamic programming technique like Dijkstra. Then, the motion controller is used to animate the agent along the path planned. Kuffner argue that his technique is fast enough to be used in dynamic environments. Another example is the work developed by Metoyer and Hodgings [Metoyer and Hodgins 2004]. They proposed a technique where the user defines the path that should be followed by each agent. During the motion along this path, it is smoothed and slightly changed to avoid collisions using force fields that act on the agent.

The development of randomized path-finding algorithms – specially the PRM (Probabilistic Roadmaps) [Kavraki et al. 1996] and RTT (Rapidly-exploring Random Tree) [LaValle 1998] – allowed the use of large and more complex configuration spaces to generate paths efficiently. Thus, the challenge becomes more the generation of realistic movements than finding a valid path. For instance, Choi et al. [Choi et al. 2003] use a library of captured movements associated to the PRM to generate realistic movements in a static environment, that is, live-captured motions are used insofar the agent tracks the path computed from a roadmap. Despite the fact the path is computed in a pre-processing phase, results are very realistic. Pettré et al. [Pettre et al. 2002] improved this idea adding one more step in this process. This step consists of smoothing the path computed by the PRM using Bézier curves. Hereinafter, the already captured motions are associated to the agent position during the path execution. As in previous works, the motion is also performed on a 2D environment.

Differently, Burgess and Darken [Burgess and Darken 2004] proposed a method based on the *principle of least action* which describes the tendency of elements in nature to seek the minimal effort solution. Authors claim that a realistic path for a human is the one that requires the smallest amount of effort. The method produces human-like movements, through very realistic paths, using properties of fluid simulation.

Tecchia et al. [Tecchia et al. 2001] proposed a platform that aims to accelerate the development of behaviors for agents through local rules that control these behaviors. These rules are governed by four different control levels, where each one reflects a different aspect of the behavior of the agent. Results show that, for a fairly simple behavioral model, the system performance can achieve interactive time.

Pelechano et al. [Pelechano et al. 2005] described a new architecture to integrate a psychological model into a crowd simulation system in order to obtain believable emergent behaviors. The architecture achieves individualistic behaviors through the modeling of the agent knowledge, as well as the basic principles of communication between agents.

Treuille et al. [Treuille et al. 2006] proposed a crowd simulator driven by dynamic potential fields which integrates both global navigation and local collision avoidance. Basically, this technique uses the crowd as a density field, and, for each group, constructs a unit cost field which is used to control people displacement. The method produces smooth behavior for a large amount of agents at interactive rates.

Recently, Reynolds [Reynolds 2006] implemented a high performance multi-agent simulation and animation for the Playstation® 3. Basically, his technique uses a spatial partitioning that divides the simulation into disjoint jobs which are evaluated in an arbitrary order on any number of Playstation® 3 Synergistic Processor Units (SPUs). A fine-grain partitioning suits SPU memory size and provides automatic load balancing. This approach allows a scalable multi-processor implementation of a large and fast crowd simulation, achieving good frame rates with thousand of agents.

In 2008, Bleiweiss [Bleiweiss 2008] implemented the Dijkstra and the A* algorithms using CUDA. Differently from our work, these algorithms are used in the path finding problem with pre-computed graphs. After several benchmarks, he observed that the Dijkstra implementation reached a speed up of 27 times compared to a C++ implementation without SSE instructions. The A* implementation reached a speed up of 24 times compared to the C++ implementation with SSE instructions.

Based on local control, van den Berg [van den Berg et al. 2008] proposed a technique that handles the navigation of multiple agents in the presence of dynamic obstacles. He uses an extended *velocity obstacles* concept to locally control the agents with few oscillation. Kapadia [Kapadia et al. 2009] presented a framework that enables agents to navigate in unknown environments based on *affordance fields* that compute all the possible ways an agent can interact with its environment.

As mentioned above, most of the approaches do not take into account the individual behavior of each agent, his internal state or mood. Our assumption is that realistic paths derive from human personal characteristics and internal state, thus varying from one person to another. As a consequence, we [Dapper et al. 2006; Dapper et al. 2007] recently proposed a technique that generate individual paths. Our path is smooth and is dynamically generated while the agent walks. In the following sections, we will explain the concepts of our technique and our strategy to implement it on the GPU.

3 Path Planner based on Boundary Value Problems

Recently, we [Dapper et al. 2006; Dapper et al. 2007] developed a technique that produces natural and individual behaviors for virtual humanoids. This technique is based on an extension of the Laplace's Equation that produces a family of potential field functions that do not have local minima. This family is generated through the numeric solution of a convenient partial differential equation with Dirichlet boundary conditions, i.e., a boundary value problem (BVP). Boundary conditions are central to the method indicating which regions in the environment are obstacles and which ones are targets. Our method uses the following equation

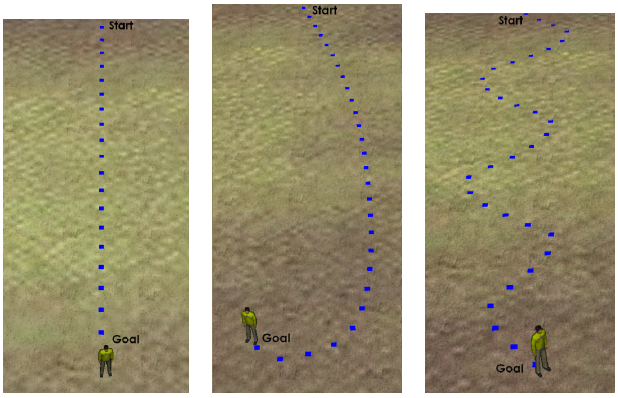
$$\nabla^2 \, p(\mathbf{r}) + \epsilon \mathbf{v}.\nabla p(\mathbf{r}) \; = \; 0 \qquad (1)$$

where $\mathbf{v}$ is a bias unity vector and ϵ is a scalar value.

The use of terms ϵ and $\mathbf{v}$ distort the potential field providing a preferred direction to be followed. This distortion allows the production of individual behaviors for humanoids illustrated through the path followed by each one during navigation tasks.

To generate realistic steering behaviors, we need to conveniently adjust both parameters ϵ and $\mathbf{v}$. The vector $\mathbf{v}$, called *behavior vector*, can be thought as an external force that pulls the agent to its direction always as possible whereas the parameter ϵ can be understood as the *strength* or *influence* of this vector in the agent behavior. The allowed values of parameters ϵ and $\mathbf{v}$ permit to generate an expressive amount of action sequences – *displacement sequences* – that virtual humanoids can use to reach a specific target position.

Figure 2 shows three different paths followed by an agent using the Equation 1 and changing the parameters ϵ and $\mathbf{v}$.



(a) (b) (c)

Figure 2: *Different paths followed by an agent using Equation 1: (a) path produced by harmonic potential, i.e., with $\epsilon = 0$; (b) with $\epsilon = -1.0$ and $\mathbf{v} = (1, 0)$; (c) with $\epsilon = -1.0$ and $\mathbf{v} = (1, sin(0.6t))$*

.

Two action sequences are not statically defined for a same pair ϵ and $\mathbf{v}$, i.e., the path generated vary according to the information gathered by the agent to allow it to dynamically react against unexpected events (e.g. dynamic obstacles). In other words, the configuration of the obstacles has an important role in the generation of the path.

Besides, this pair is not constrained to keep constant during the execution of tasks. They can vary insofar the agent displaces in the environment to obtain the desired behavior. Figure 2(c) shows a situation where the behavior vector varies according to a sin function. It is not natural for human beings to walk based on a sin function. However, the path based on a sin function illustrates the flexibility of Equation 1. Any function can be associated to $\mathbf{v}$ and ϵ to generate a behavior.

When $\epsilon = 0$, Equation 1 reduces to $\nabla^2\ p(\mathbf{r}) = 0$ which corresponds to Laplace's Equation. This equation is used as core of the path planner based on harmonic function developed by Connolly and Grupen [Connolly and Grupen 1993] on Robotics context. This planner produces paths that minimize the hitting probability of the agent with obstacles, i.e., in an indoor environment the agent will tend to follows a path equidistant to the walls, as shown in Figure 2(a). This behavior is not always adequate to simulate humanoid motion since it looks very stereotyped because humans do not always walk equidistant to the walls. Hence the importance of using these parameters ϵ and $\mathbf{v}$.

The common approach to numerically solve a BVP is to consider that the solution space is discretized in a regular grid. Each cell (i, j) is associated to a squared region of the real environment and stores a potential value $p^t_{i,j}$ at instant t. Each cell is distant from each other 1 unit. The Dirichlet boundary conditions previously associate a specific potential value to some cells, before the relaxation process is performed. That is, cells associated to obstacles in the real environment store a potential value equal to 1 (*high potential*) whereas cells containing the target store a potential value equal to 0 (*low potential*). The high potential value prevents the agent from running into obstacles whereas the low potential value generates an attraction basin that pulls the agent. The potentials of the other cells are computed using the Gauss-Seidel relaxation method, as discussed in [Prestes et al. 2002]. By considering the Equation 1, the potentials of the free space cells are updated through the following equation

$$p_c = \frac{p_b + p_t + p_r + p_l}{4} + \frac{\epsilon((p_r - p_l)v_x + (p_b - p_t)v_y)}{8} \quad (2)$$

where $p_c = p^{t+1}_{i,j}$, $p_b = p^t_{i,j+1}$, $p_t = p^{t+1}_{i,j-1}$, $p_r = p^t_{i+1,j}$, $p_l = p^{t+1}_{i-1,j}$ and $\mathbf{v} = (v_x, v_y)$. Figure 3 shows a representation of these cells.

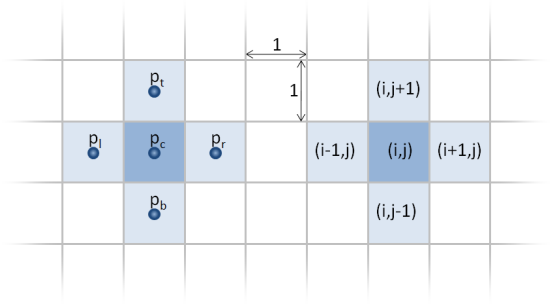


Figure 3: *Representation of p_c, p_b, p_t, p_r and p_l on the grid.*

The parameter $\mathbf{v}$ must be a unit vector and ϵ must be in the interval $(-2, 2)$. Values out of this range generate oscillatory and unstable paths that do not guarantee that the agent will reach the target or will avoid obstacles. This happens because the boundary conditions – that assert the agent is repelled by obstacles and attracted by targets – are violated.

After the potential computation, the agent moves following the direction of the gradient descent of this potential at its current position (i, j),

$$(\nabla \mathbf{p})_{(i,j)} = \left(\frac{p_{i+1,j} - p_{i-1,j}}{2}, \frac{p_{i,j+1} - p_{i,j-1}}{2} \right)$$

This process is an intuitive way to control the agent motion. However, it can easily fail in producing realistic steering behaviors, as observed in real world. One of the reasons is that the agent changes its direction based solely on the gradient descent of its position. For instance, if the field of view of the agent is small, its reaction time will be very short to treat dynamic obstacles[1]. Then, these obstacles will produce a strong repel force that will change the agent direction abruptly. As we can see in Figure 4, if the agent uses only the gradient descent (*dgrad*) it will change its direction in nearly $\pi/2$.

We handle this problem by adjusting the current agent position by

$$\Delta\ \mathbf{d} = \upsilon(\cos(\varphi^t), \sin(\varphi^t)) \quad (3)$$

where υ defines the maximum agent speed and φ^t is

$$\varphi^t = \eta\ \varphi^{t-1} + (1 - \eta)\ \zeta^t \quad (4)$$

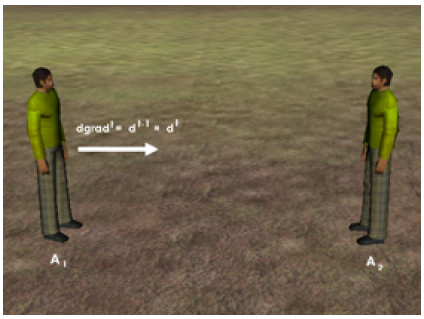
where $\eta \in [0, 1)$ and ζ is the orientation of the gradient descent at current agent position.

When $\eta = 0$, the agent adjusts its orientation using only information about the gradient descent. If $\eta = 0.5$, the previous agent direction (φ^{t-1}) and the gradient descent direction influence equally the computation of the new agent direction. Figure 4(b) shows the vector $\mathbf{d}^t$ with orientation φ^t computed with $\eta = 0.5$. The parameter η can be viewed as an inertial factor that tends to keep the agent direction constant insofar $\eta \to 1$. When $\eta \to 1$, the agent reacts slowly to unexpected events, increasing its hitting probability with obstacles. η is a flexible parameter that the user is able to control. However, a learning strategy could be used to specify what is the best η to a specific situation.

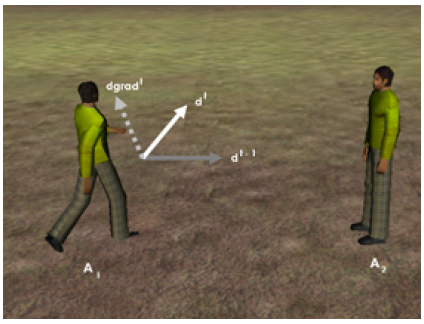
Despite Equation 3 produces good results and smooth paths in environments with few obstacles, when the environment is cluttered with obstacles, the agent behavior is not realistic and collisions can happen. To solve this problem, a speed control was incorporated into this equation,

$$\Delta\ \mathbf{d} = \upsilon\ (\cos(\varphi^t), \sin(\varphi^t))\ \Psi(|\varphi^{t-1} - \zeta^t|) \quad (5)$$

[1]We consider that dynamic obstacles (as other agents) are mapped in the environment only when they are inside the field of view of the agent, which almost corresponds to reality.



(a)



(b)

Figure 4: *Defining agent motion. (a) Situation before the agent A_2 enters in the field of view of A_1. (b) If the agent A_1 follows the direction defined by the gradient descent (*dgrad*), it will changes its direction in nearly $\pi/2$, what is undesirable. However, if the agent uses the vector d, it will achieve a smooth curve, what is more natural and realistic.*

where function $\Psi : \mathbf{R} \rightarrow \mathbf{R}$ is

$$\Psi(x) = \begin{cases} 0 & \text{if } x > \pi/2 \\ \cos(x) & \text{, otherwise} \end{cases} .$$

If $|\varphi^{t-1} - \zeta^t|$ is higher than $\pi/2$, then there is a high hitting probability and this function returns the value 0, making the agent stops. Otherwise, the agent speed will change proportionally to the collision risk. In regions cluttered with obstacles, agents will tend to move slowly. If a given agent is about to cross the path of another, one of them will stop and wait until the other get through. Furthermore, speed control allows the simulation of agents' mood through the variation of the speed magnitude, that is, it is possible to simulate a tired agent making it move slower and an agent that is anxious about its work making it move faster.

4 Implementation Strategy

As previously explained, our motion planning method requires the discretization of the environment into a regular grid. In this section we present the strategy that was used in our previous work [Dapper et al. 2006; Dapper et al. 2007] to implement it by using global environment maps (one for each target) and local maps (one for each agent), as well as the mechanisms used to control each agent steering behavior.

4.1 Environment Global Map

The entire environment is represented by a set of homogeneous meshes, $\{\mathcal{M}_k\}$, in which each mesh $\mathcal{M}_k$ has $L_x \times L_y$ cells, denoted by $\{\mathcal{C}^k_{i,j}\}$. Each cell $\mathcal{C}^k_{i,j}$ corresponds to a squared region centered in environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $\mathcal{P}^k_{i,j}$. The potential associated to the mesh $\mathcal{M}_k$ is computed by the harmonic path planner, through the Equation 2, and then used by agents to reach the target $\mathcal{O}_k$.

In order to delimit the navigation space, we consider that the environment is surrounded by static obstacles. Global maps are built before simulation starts, in a pre-processing phase.

4.2 Agent Local Map

Each agent a_k has one map m_k that stores the current local information about the environment obtained by its own sensors. This map is centered in the current agent position and represents a small fraction of the global map, usually about 10% of the total area covered by the global map.

The map m_k has $l^k_x \times l^k_y$ cells, denoted by $\{c^k_{i,j}\}$ and divided in three regions: the update zone (*u-zone*); the free zone (*f-zone*) and the border zone (*b-zone*), as shown in Figure 5. Each cell corresponds to a squared region centered in environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $p^k_{i,j}$.

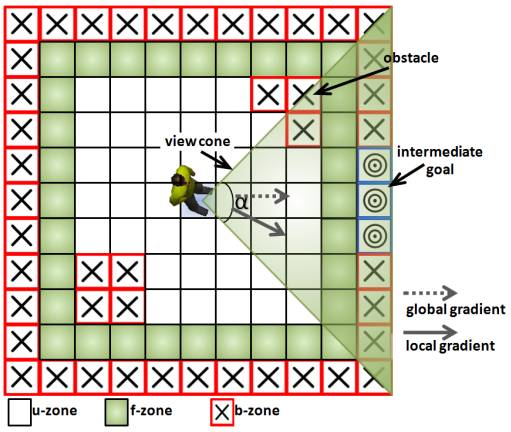


Figure 5: *Agent Local Map. The update (u-zone), free (f-zone) and border zones (b-zone) are shown. Blue and red cells correspond to the intermediate goal and obstacles, respectively.*

The area associated to each agent map cell is smaller than the area associated to the global map cell. The main reason is that the agent map is used to produce refined motion, while the global map is used only to assist the long-term agent navigation. Hence, the smaller the size of the cell on the local map, the better the quality of motion.

4.3 Updating Local Maps from Global Maps

For each agent a_k, a goal $\mathcal{O}_{goal(k)}$[2], a particular vector $\mathbf{v}_k$ that controls its behavior, and a ϵ_k should be stated. The same goal, $\mathbf{v}$, and ϵ can be designated to several agents. If $\mathbf{v}_k$ or ϵ_k is dynamic, then the function that controls it must also be specified.

To navigate into the environment, an agent a_k uses its sensors to perceive the world and to update its local map with information about obstacles and other agents. The agent sensor sets a view cone with aperture α.

Figure 6 exemplifies a particular instance of the agent local map where we can see the obstacles mapped from the global map. The *u-zone* cells $c^k_{i,j}$ which are inside the view cone and correspond to obstacles or other agents have their potential value set to 1. In Figure 7, as there is an agent in the *u-zone* of the agent local map, inside of his view cone, it is mapped as an obstacle into his local map. This procedure assures that dynamic or static obstacles behind the agent (out of his view cone) do not interfere in his future motion.

For each agent a_k, the global descent gradient on the cell in the global map $\mathcal{M}_{goal(k)}$ that contains his current position is calculated. The gradient direction is used to generate an intermediate goal in the border of the local map, setting the potential values of a couple of *b-zone* cells to 0, while the other *b-zone* cells are considered as obstacles, with their potential values set to 1. In Figure 7, the agent calculates his global gradient in order to project an intermediate goal in its own local map. As the agent local map is delimited by obstacles, the agent is pulled towards the intermediate goal using the direction of his local gradient. The intermediate goal helps the agent a_k to reach its target $\mathcal{O}_{goal(k)}$ while allowing it to produce a particular motion.

[2]Function *goal()* maps the agent number k into its current target number

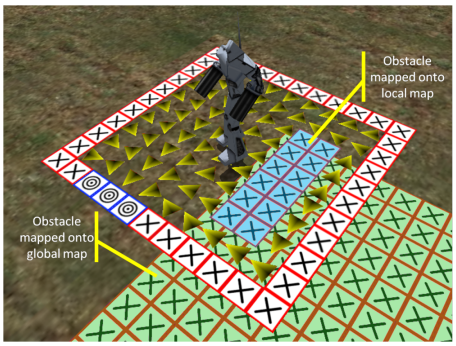


Figure 6: *Global map mapped onto the agent local map.*

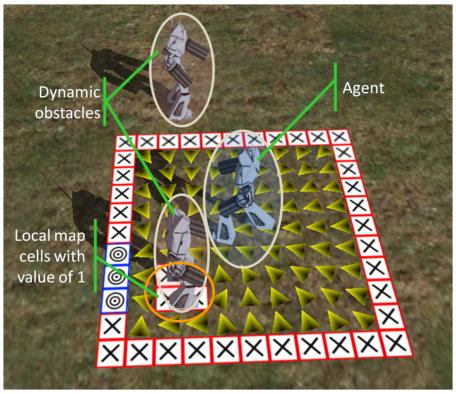


Figure 7: *The cells which are inside the agent's view cone and correspond to obstacles or other agents have their potential value set to* 1.

In some cases, the target $\mathcal{O}_{goal(k)}$ is inside both view cone and *u-zone*, and consequently local map cells associated are set to 0. The intermediate goal is always projected, even if the target is mapped onto the *u-zone*. Otherwise the agent can easily get trapped because it would be taking into consideration only the local information about the environment, in a same way as traditional potential fields [Khatib 1980].

F-zone cells are always considered free of obstacles, even when there are obstacles inside. The absence of this zone may close the connection between the current agent cell and the intermediate goal due to the mapping of obstacles in front of the intermediate goal. When this occurs, the agent gets lost because there is no information coming from the intermediate goal to produce a path to reach it. *F-zone* cells handle the situation always allowing the propagation of the goal's information to the cells associated to the agent position.

After the sensing and mapping steps, the agent k updates the potential value of its map cells using Equation 2 with its pair $\mathbf{v}^k$ and ϵ^k. Hereinafter, it updates its position according to Equation 5 using the gradient descent computed from the potential field stored on its local map in the position $p_x = \lceil l_x^k/2 \rceil$ and $p_y = \lceil l_y^k/2 \rceil$.

5 Implementation on GPU

In the real world, people walking inside a room react to what they perceive from the environment based on their own personality, mood and reasoning, i.e., they think in parallel. So, a technique that handles several agents should be parallelized in the same way.

According to Section 3, during the update phase of our technique, each agent must update its local map with the environment obstacles which are inside this region. Note that, in this step, we consider that for a given agent a_i, each other agent $a_j, i \neq j$, is also an obstacle. Then, each cell in the agent local map inside his view cone is updated as an obstacle, with the potential value equal to 1. After the update of these cells, we update the cells which correspond to the agent goal, with the potential value of 0.

Note that each one of these updates can be made in parallel between the agents. The only dependency here is that obstacle cells must be updated before goal cells. It must be done sequentially, otherwise, if an agent has a goal very close to an obstacle, both obstacle and goal may be mapped to the same cell. In this case, if goal cells are updated before obstacle cells, the agent will become lost, without a goal to achieve. All other cells are updated as free cells.

Afterwards, the Equation 2 is evaluated for each agent local map. Since it is difficult and needs to be evaluated independently for each agent, it is a good candidate for a parallel implementation. The Gauss-Seidel relaxation method (previously used in Equation 2) is not suitable for a parallel implementation because it uses values from the current and previous iterations. In a sequential approach, it is very simple to implement and fast to execute, but a parallel implementation will require some kind of synchronization, which may cause degradation in performance. A better approach for a parallel implementation is to use values only from the previous iteration. This is exactly what the Jacobi method does. The update rule is described below.

$$p_c = \frac{p_b + p_t + p_r + p_l}{4} + \frac{\epsilon((p_r - p_l)v_x + (p_b - p_t)v_y)}{8} \quad (6)$$

where $p_c = p^t_{i,j}$, $p_b = p^t_{i,j+1}$, $p_t = p^t_{i,j-1}$, $p_r = p^t_{i+1,j}$, $p_l = p^t_{i-1,j}$ and $\mathbf{v} = (v_x, v_y)$.

We implemented the parallel version of our technique using the nVIDIA® Cuda [NVIDIA. 2009] language, which allows us to use the graphics processor without using shading languages. In the context of CUDA, the CPU, here called *Host*, controls the graphics processor, called *Device*. It sends data, calls the *Device* to execute some functions, and then copies back its results.

Each graphics processor of a nVIDIA graphics card is divided into several multiprocessors. Cuda divides the processing in blocks, where each block is divided in several threads. Each block of threads is mapped to one multiprocessor of the graphics processor. When the *Host* calls the *Device* to execute a function, it needs to inform how the work will be divided in blocks and threads. Maximum performance is achieved when we maximize the use of blocks and threads for a given graphics processor.

Each of the multiprocessors is a group of simple processors that share a set of registers and some memory (the *shared memory* space). The shared memory size is very small (16KB on graphics cards up to *Compute Capability 1.3*), but it is as fast as the registers. The communication between two multiprocessors must be done through the Device Memory, which is very slow if compared to the shared memory. There is also the *Constant Cache* and *Texture Cache memory*, which has better access times than the Device memory, but it is read-only for the *Device*.

Before the execution of the code in the *Device*, the *Host* must send the data to its Device Memory to be processed later. The memory copy from the Host Memory to the Device memory is a slow process, and should be minimized. Besides, the nVIDIA Cuda Programming Guide [NVIDIA. 2009] says that one single call to the memory copy function with a lot of data is much more efficient than several calls to the same function with a few bytes. We can improve the performance of our application making good use of these restrictions of Cuda.

As previously mentioned, each agent a_k has several attributes: the scalar ϵ_k, the vector $\mathbf{v}_k$, and its current objective $\mathcal{O}_{goal(k)}$. The local map also has some attributes, like its width l_x^k and height l_y^k. All these attributes must be sent at least once to the *Device*. The agent goal and the local map position in the world, for instance, will be frequently updated. To avoid several memory transactions between the *Host* and the *Device*, we store all these attributes in contiguous memory areas, and treat it like an array. At the position k we store an attribute of the agent a_k. Proceeding this way, we avoid several unnecessary copies, improving the overall performance.

Figure 8 shows our data structure for a set of 3 agents. The array m with all local map cells is illustrated with its cell's index. Each position k of the array $\mathbf{s}$ contains an index to the first position in the array $\mathbf{m}$ in which the agent a_k local map information is stored. Each position k of the array $\mathbf{l}, \mathbf{O}, \epsilon, \mathbf{v}$ contains the information of the

local map dimension and goal, as well as the behavioral parameters ϵ and $\mathbf{v}$ of the agent a_k, respectively.

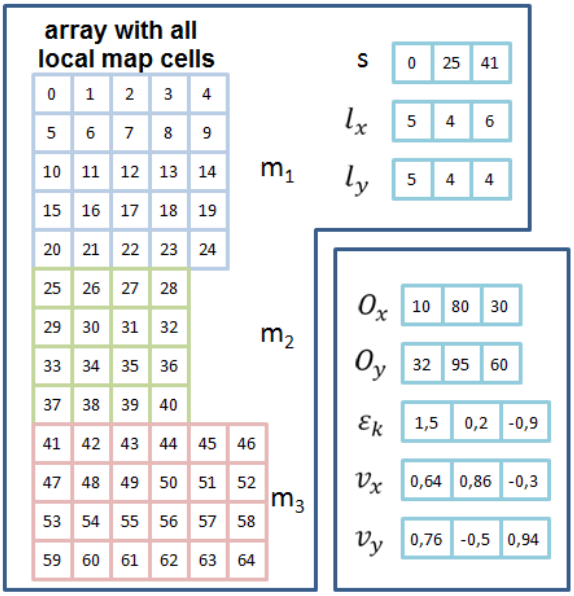


Figure 8: *Data structure used on GPU.*

There are situations in which the size of the agent local map must be changed. Any update on the size of an agent local map will require the modification of the array m, which implies in the entire data structure reconstruction. In these cases, the *Host* must reallocate the entire array in the Host Memory, and send it again to the Device Memory. These attributes should not only be copied once to the Device memory, but they should be sent to the Constant Memory or Texture Memory.

As the *Environment Global Map* is composed only of static obstacles, it can be copied to the Device Memory only once. Then, the update step can be done in the following way. First, each local map is mapped into a block of threads, in which each thread updates one cell of the local map. The thread will find the local map cell corresponding to the Environment Global Map, and will copy the information from the global map cell to the local map cell. This is done only to the cells in the *f-zone*. Figure 6 illustrates this situation.

Afterwards, each local map is mapped to a block of threads, and each thread is associated with a dynamic obstacle. This thread checks whether the obstacle appears inside the view cone. If yes, the local map cells occupied by the obstacle update its potential value to 1. Next, each cell in the *b-zone* is mapped as an obstacle, also updating its potential to 1, except for the ones that are goal cells. The remaining cells are updated as free cells. Then, the Equation 6 can be evaluated, starting one thread to each local map cell. A synchronization must be made between the iterations in order to guarantee that all cells are up to date to the next iteration.

The convergence of the Equation 6 is achieved through several *reads* and *writes* at the Device Memory during several iterations. In order to avoid the high latency of the Device Memory, this must be made in the shared memory of the multiprocessor. An implementation of the Jacobi method will require two copies of the potential map, where at each iteration the values are *read* from one of them and *written* to the other. However, the shared memory size is very limited. Then, we decided to use a combination of the Jacobi method with the Gauss-Seidel. In our implementation, only one copy of the potential map is stored in the shared memory. At each iteration t, a cell $c^k_{i,j}$ may be updated with the potential of the neighborhood cells at the iteration $t-1$ or t. We do not specify whether will be used values from iteration $t-1$ or t. It will depend on how the information will be arranged in the shaders, i.e., the synchronization between cells update is not needed.

6 Results

In order to verify that our parallel implementation can be executed faster than the sequential one, a couple of tests were accomplished. All the tests were executed in an Intel® Core 2 6300 1.86GHz, with 2Gb of RAM memory, a nVIDIA GeForce 9800 GX2 graphics card (the graphics processor has 600 MHz of clock) and Microsoft Windows XP SP3 operating system. We measured how many times per second the algorithm can be executed, and what is the impact of the memory copy between the *Host* and the *Device*, using three different sizes for the local maps.

The tests were executed in the following way. Initially, three sizes of local maps where chosen: 11×11, 16×16 and 21×21. We chose these sizes because previous tests [Dapper et al. 2006] showed that they generate animations with very good quality, being the most interesting for tests. Then, several scenarios were executed using the parallel and sequential versions of the algorithm, changing the number of agents in the scene. For each test, we recorded the frequency at which the algorithm can be executed, and the percentage of time spent in memory copies between the *Host* and the *Device*.

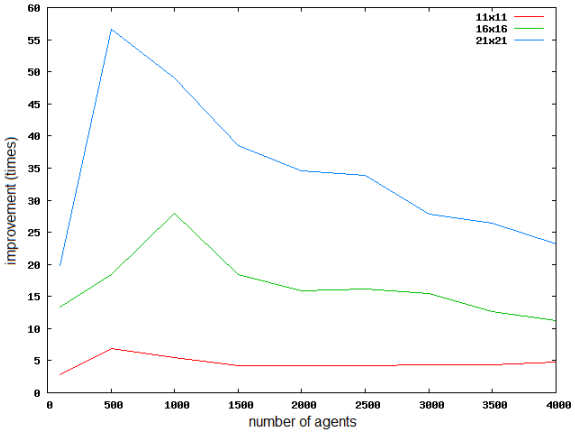


Figure 9: *Speed up achieved using the parallel implementation over the sequential version, with three different sizes of local maps.*

The graphic in Figure 9 shows the speed up achieved using the parallel implementation over the sequential version of the technique. As we can see, in all tests executed the parallel version was above twice faster than the sequential one (exactly the lowest point in the graphic is at 2.85 times). Besides that, the highest point in the graphic occurs at the point 56.60, meaning that in an optimal configuration the parallel version was 56 times faster than the sequential version.

Using bigger local maps means that more threads are needed for each local map in the several steps of the technique. The fact that the multiprocessor offers several running threads at the same time implies in a better use of the resources and in good improvements in performance.

On the other hand, for several reasons, with smaller local maps the speed up is not so high. On the side of the parallel version, a small local map does not make a good use of the resources of each multiprocessor. And on the side of the sequential version, a small local map may fit better in the processor cache. Moreover, the processor clock is three times higher than the graphics processor clock. If we combine all these factors in the same test, the speed up in the parallel version is minimized.

In addition, according to the nVIDIA Cuda Programming Guide [NVIDIA. 2009], the graphics processor cannot handle all the data in a parallel way. The division of the work in blocks of threads lets the graphics processor scheduler run some blocks of thread while others wait for execution. Because of this, the computation of 256 local maps in a parallel way does not give a speed up of 256 times.

To explain what is the cause of the graphics peaks, the nVIDIA Cuda Programming Guide says that each algorithm implemented

with Cuda has an optimal point, in which the amount of blocks and threads uses the most possible number of resources available in the graphics processor simultaneously. In our technique, this point is the one with 500 agents in the scene, each one with a local map of a size of 21×21.

7 Conclusion

This paper presented a strategy to implement on GPU a BVP Planner [Dapper et al. 2007] that produces natural steering behaviors for virtual humans, using a path-planning algorithm based on the numerical solution of boundary value problems.

The guiding potential of Equation 1 is free of local minima, what constitutes a great advantage when compared to the traditional potential fields method. Furthermore, the method proposed is formally complete [Connolly and Grupen 1993] and generates smooth and safe paths that can be directly used in mobile robots or autonomous characters in games. The local information gathered by agent sensors allows treating dynamic obstacles, such as other agents navigating in the environment.

We implemented a parallel version of this algorithm using the nVIDIA® Cuda [NVIDIA. 2009] language, which allows us to use the graphics processor avoiding the use of shading languages. The parallelism was explored, reducing the amount of memory transactions between CPU and GPU.

Our result shown that the GPU implementation improves up to 56 times the sequential CPU version, allowing the real-time use of this technique even in scenarios with a huge number of autonomous characters, which is a common situation often found in games.

As future work, we suggest the exploration of ADI Method [Peaceman D. W. 1995], obtaining a faster convergence of the relaxation process. The ADI Method is suitable to be used on parallel architectures and to explore the use of other shading languages. It would be interesting to compare the possible improvements in performance using other languages.

We have also proposed an extension of this technique to manage the movement of groups of agents in dynamic environments [Silveira et al. 2008]. We intend to implement a parallel version of this extension and release the project over an open source license.

Acknowledgements

The authors would like to thank Edson Prestes and Fabio Dapper for their valuable work on the CPU version of the path-planning algorithm, and Francele Carvalho Rodrigues for helping in the orthographic and grammatical revision of the text. This work was partially supported by grants from CNPq to Leonardo Fischer, Renato Silveira and Luciana Nedel.

References

BLEIWEISS, A. 2008. Gpu accelerated pathfinding. In *GH '08: Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 65–74.

BURGESS, R. G., AND DARKEN, C. J. 2004. Realistic human path planning using fluid simulation. In *Proceedings of Behavior Representation in Modeling and Simulation (BRIMS)*.

CHOI, M. G., LEE, J., AND SHIN, S. Y. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph. 22*, 2, 182–203.

CONNOLLY, C., AND GRUPEN, R. 1993. On the applications of harmonic functions to robotics. *International Journal of Robotic Systems 10*, 931–946.

DAPPER, F., PRESTES, E., IDIART, M. A. P., AND NEDEL, L. P. 2006. Simulating pedestrian behavior with potential fields. In *Advances in Computer Graphics*, Springer Verlag, vol. 4035 of *Lecture Notes in Computer Science*, 324–335.

DAPPER, F., PRESTES, E., AND NEDEL, L. P. 2007. Generating steering behaviors for virtual humanoids using bvp control. *Proc. of CGI*.

FUNGE, J. D. 2004. *Artificial Intelligence For Computer Games: An Introduction*. A. K. Peters, Ltd., Natick, MA, USA.

JAMES J. KUFFNER, J. 1998. Goal-directed navigation for animated characters using real-time path planning and control. In *International Workshop on Modelling and Motion Capture Techniques for Virtual Environments*, Springer-Verlag, London, UK, 171–186.

KAPADIA, M., SINGH, S., HEWLETT, W., AND FALOUTSOS, P. 2009. Egocentric affordance fields in pedestrian steering. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 215–223.

KAVRAKI, L., SVESTKA, P., LATOMBE, J.-C., AND OVERMARS, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Transactions on Robotics and Automation 12*, 4, 566–580.

KHATIB, O. 1980. *Commande dynamique dans l'espace opérational des robots manipulaters en présence d'obstacles*. PhD thesis, École Nationale Supérieure de l'Aéronatique et de l'Espace, France.

LAVALLE, S. 1998. Rapidly-exploring random trees: A new tool for path planning. Tech. Rep. 98-11, Computer Science Dept., Iowa State University.

METOYER, R. A., AND HODGINS, J. K. 2004. Reactive pedestrian path following from examples. *The Visual Computer 20*, 10, 635–649.

NIEUWENHUISEN, D., KAMPHUIS, A., AND OVERMARS, M. H. 2007. High quality navigation in computer games. *Sci. Comput. Program. 67*, 1, 91–104.

NVIDIA. 2009. Nvidia cuda. *http://www.nvidia.com/cuda, last acces at 07/2009*.

PEACEMAN D. W., R. J. H. H. 1995. The numerical solution of parabolic and elliptic differential equations. *Journal of the Society for Industrial and Applied Mathematics 3*, 28–41.

PELECHANO, N., OBRIEN, K., SILVERMAN, B., AND BADLER, N. 2005. Crowd simulation incorporating agent psychological models, roles and communication. In *1st Int'l Workshop on Crowd Simulation*, 21–30.

PETTRE, J., SIMEON, T., AND LAUMOND, J. 2002. Planning human walk in virtual environments. In *IEEE/RSJ International Conference on Intelligent Robots and System*, vol. 3, 3048 – 3053.

PRESTES, E., ENGEL, P. M., TREVISAN, M., AND IDIART, M. A. 2002. Exploration method using harmonic functions. *Robotics and Autonomous Systems 40*, 1, 25–42.

REYNOLDS, C. 2006. Big fast crowds on ps3. In *sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, ACM Press, New York, NY, USA, 113–121.

SILVEIRA, R., PRESTES, E., AND NEDEL, L. P. 2008. Managing coherent groups. *Comput. Animat. Virtual Worlds 19*, 3-4, 295–305.

TECCHIA, F., LOSCOS, C., CONROY, R., AND CHRYSANTHOU, Y., 2001. Agent behaviour simulator (abs): A platform for urban behaviour development.

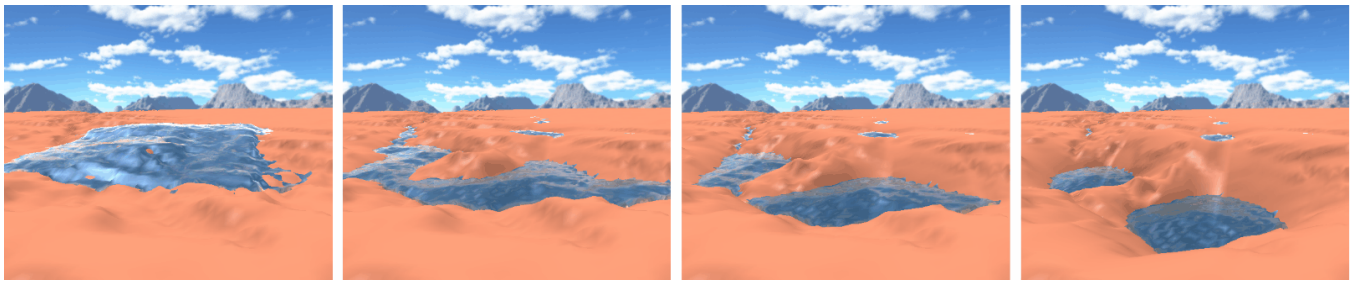
TREUILLE, A., COOPER, S., AND POPOVIĆ, Z. 2006. Continuum crowds. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM Press, New York, NY, USA, 1160–1168.

VAN DEN BERG, J., PATIL, S., SEWALL, J., MANOCHA, D., AND LIN, M. 2008. Interactive navigation of multiple agents in crowded environments. In *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 139–147.

GPU Based Fluid Animation over Elastic Surface Models

*Bruno Barcellos S. Coutinho and Gilson Antonio Giraldi
National Laboratory for Scientific Computing
Petropolis, RJ, Brazil
(lsdelphi, gilson)@lncc.br

Antonio L. Apolinário Jr.
State University of Feira de Santana
Bahia, BA, Brazil
apolinario@ecomp.uefs.br



Keywords:: Real-time Animation, Cellular Automata, Fluid Simulation, Elastic Surfaces, GPU programming.

*Bruno Barcellos now is a master student in the Computer Graphics Laboratory (LCG) at Federal University of Rio de Janeiro (UFRJ-COPPE)

Abstract

In this work, we focus on flow animation in elastic surfaces described by mass-spring models for computer game applications. We propose the combination of an efficient fluid model, that does not require solution of complicated equations, with a mass-spring approach to simulate the deformable surface. Firstly, we describe the fluid model for simulating the flow and its GPU implementation. The simulation method is based on a particle system, that evolves over a lattice. This lattice is defined over the surface domain. A set of local rules determine the interaction between particles. The elastic surface is simulated by a GPU based mass-spring system, geometrically represented by a regular mesh. The fluid particles are guided by the surface topography interacting with the elastic mesh due to external, elastic and damping forces. In the experimental results we emphasize the fact that physically plausible flow/deformation phenomena can be efficiently reproduced and animated in real time by the combined technique.

1 Introduction

Physically-based techniques for the animation of natural elements like fluids (gas or liquids), flood, elastic, plastic and melting objects, among others, have taken the attention of the computer graphics community [Iglesias 2004; Nealen et al. 2005; Deusen et al. 2004]. In particular, techniques in the field of Computational Fluid Dynamics (CFD) have been applied for fluid animation in applications that involve the interaction between the fluid and deformable solids [Müller et al. 2004; Keiser et al. 2005; Genevaux et al. 2003; Chentanez et al. 2006].

A common approach in this area relies on top down viewpoints that use 2D/3D mesh based techniques in conjunction with fluid equations [Chentanez et al. 2006]. Other possibility is to apply mass-spring systems to model the elastic solid [Genevaux et al. 2003]. Mass-spring models are well suited to animation due to its flexibility to handle non-rigid solid properties, its easy manipulation and implementation. Besides, mass-spring models can be faster then their counterpart in continuous mechanics, and so, more suitable for real time applications specially when GPU capabilities are explored [Mosegaard and Sorensen 2005].

However, a common challenge is this area is the cost of the computational animation of the solid-fluid interaction. Recently, it has been demonstrated the advantages of using *bottom up* models for surface water flow simulation which are inspired in Lattice Gas Cellular Automata (LGCA) and LBM techniques [Barcellos et al. 2007; Fan et al. 2005]. These methods are cheaper than the traditional ones for fluid simulation, because there is no need to solve Partial Differential Equations (PDEs) to obtain a high level of description [Frisch et al. 1987]. Convincing animations with real time frame rates can be generated by lattice methods, as demonstrated in [Judice et al. 2008] for computer game applications involving surface flow over terrains.

In this paper we combine the mass-spring model described in [Mosegaard and Sorensen 2005] with an extension of the surface flow simulation proposed in [Barcellos et al. 2007] in order to get a general system for GPU animation of fluids over elastic surfaces. The combined technique is the main contribution of this paper. The advantages of this method over other approaches are its simplicity for implementation and the gain in computational efficiency allowing real time frame rates. The basic data structures of the model are a polygonal representation of the surface and a regular lattice with nodes $(i, j) \in L \times L$, where $L \subset \mathbb{N}$. Up to now, we consider surface flow simulation over deformable $2D$ manifolds which have a global parameterization $\varphi : D \rightarrow \mathbb{R}$ where $\varphi(x, y, t)$ is the elevation of the surface at point (x, y), at time t. So, each lattice node (i, j) is a projection of a surface point $(i, j, \varphi(i, j, t))$. The surface deforms as a mass-spring system, due to internal forces (elastic and damping ones) and external forces (gravitational, etc.). Extensions for more general parametric surfaces will be discussed in section 6. Potential applications in games are erosion effects and deformation of objects, modeled by elastic surfaces, under water accumulation.

The fluid model uses a LGCA approach. Therefore, particles can only move along the edges of the lattice and their interactions are based on local rules. Differently from traditional LGCA approaches, in our model more than one particles may share the same node position (i, j). Particles move according to the surface topography and the fluid configuration nearby. There is a counter in each lattice node used to keep the number of particles in the corresponding (i, j) position. When particles move over the lattice the particles counters change and, consequently, the flow distribution is updated and the surface deforms. The obtained result is a function $f(i, j, t)$ which gives the elevation of the free surface of the fluid, at the point (i, j), in the simulation time t.

The paper is organized as follows. Sections 2 gives a survey of works in interaction between fluids and elastic materials. In Section 3 we present our technique for simulating surface water flow over deformable surfaces. Its GPU implementation is described in Section 4. The experimental results are presented on Section 5. Finally, we present the conclusions and future works on Section 6.

2 Interaction of Fluids with Deformable Objects

The main focus of this work is the animation of fluids interacting with deformable objects. Generally speaking, this subject includes:

(1) Representation of the object geometry; (2) The modeling of mechanical behavior of elastic solid; (3) A suitable model for fluid simulation; (4) A model for interaction of the flow with the object; (5) Visualization and rendering issues.

The $3D$ object geometry is usually represented by mesh based methods that offer the support for Finite Element techniques [Müller et al. 2002; Debunne et al. 2001]. Bidimensional manifolds can be represented by using implicit surfaces [Zhu and Bridson 2005], triangulated meshes or subdivision surfaces with local parameterization for representation [Guendelman et al. 2005; Stam 2003].

The mechanical behavior of elastic objects (item (2)) can be described by the continuum elasticity theory that models how the objects deform under applied forces. In this case, constitutive laws are used for the computation of the symmetric internal stress tensor σ, and a conservation law gives the final PDE that governs the dynamic of the material [Nealen et al. 2005]. Then Finite Element methods, Boundary Element Method (BEM), Finite Differences (FDM) or SPH techniques, are applied to solve the PDE [Müller et al. 2002; James and Pai 1999; Terzopoulos and Witkin 1988; Debunne et al. 2000; Desbrun and Gascuel 1996]. Other possibility is to apply discrete models, based on mass-spring systems [Genevaux et al. 2003; Mosegaard and Sorensen 2005]. In this case, the object geometry is represented by a mesh and its nodes are treated like mass points while each edge acts like a spring connecting two adjacent nodes. The relation between mass-spring models and the continuum elasticity theory was examined in the reference [Delingette 2008]. The conclusion is that methods that are based on the continuum mechanics are more realistic than their discrete counterparts. However, mass-spring models are simple to implement and can be faster then the continuous ones, and so, more suitable for real time applications [Volino and Magnenat-Thalmann 2000; House and Breen 2000; VanGelder and Wilhelms 1997; Etzmuss et al. 2003].

The item (3) involves numerous works that can be coarsely classified in non-physically and physically based models [Iglesias 2004; Deusen et al. 2004]. Our work belongs to the later class, which can be subdivided in PDEs and Lattice based techniques [Frisch et al. 1987; Iglesias 2004].

PDEs methods involve continuous fluid equation, like the Navier-Stokes ones, and numerical techniques based on discretization approaches that can be Lagrangian (Smoothed Particle Hydrodynamics (SPH) [Liu and Liu 2003], method of characteristics [Stam 1999], Moving-Particle Semi-Implicit [Premoze et al. 2003]) or Eulerian (Finite Element) ones [Foster and Metaxas 1997]. The former class uses particle systems for discretization while in the latter model properties are computed for a set of stationary points, usually connected in a mesh, and updated to get the time evolution of the continuous media. PDE based models are called *top down* approaches because continuous mechanics concepts are applied to derive the PDE, which governs the continuous dynamics, while the particle view appears as a consequence of discretization methods.

Alternatively, lattice methods comprised by the Lattice Gas Cellular Automata (LGCA) - FHP and HPP are the most known ones - and Lattice Boltzmann (LBM) can be used [Frisch et al. 1987; Wei et al. 2004; Ye et al. 2004]. The basic idea behind these methods is that the macroscopic dynamics of a fluid is the result of the collective behavior of many microscopic particles. The LGCA will follow this idea but simplifying the dynamics through simple and local rules for particles interaction and displacements while LBM constructs a simplified kinetic model, a simplification of the Boltzmann equation, that incorporates the essential microscopic physics so that the macroscopic averaged properties obey the desired macroscopic equations [Chen and Doolen 1998]. Therefore, lattice methods do not apply PDEs to simulate fluids which can reduce the computational cost of the animation. Recently, it was demonstrated the advantages of using the philosophy behind FHP and HPP models for computer graphics applications and for surface water flow simulation [Barcellos et al. 2007]. Besides, despite of some intrinsic limitations, multiscale techniques were applied to demonstrated that the FHP and LBM models can reproduce Navier-Stokes behaviors [Frisch et al. 1987].

Particularly, the FHP model has a number of advantages over more traditional numerical methods, particularly when fluids mixing and phase transitions occur [Rothman and Zaleski 1994]. The simulation is always performed on a regular grid and can be efficiently implemented on a massively parallel computer. Solid boundaries and multiple fluids can be introduced in a straightforward manner and the simulation is performed equally efficiently, regardless of the complexity of the boundary or interface [Buick et al. 1998]. In addition there are not numerical instability issues because the evolution follows integer arithmetic. However, system parameterization (viscosity, for example) is a difficult task in such lattice models and they are less realistic than PDE based models.

The item (4), interaction between deformable solids and fluids, can be addressed by hybrid methods (fluid is a continuum medium and the solid is represented as a discrete one), SPH based techniques and variational approaches [Genevaux et al. 2003; Solenthaler et al. 2007; Batty et al. 2007]. In [Genevaux et al. 2003] the interaction problem is addressed by an hybrid technique in which the deformable solid is represented through a mass-spring network and the fluid is simulated by Navier-Stokes equations and an Eulerian method. The key idea is to apply spring forces to mass-less marker particles in the fluid and the nodes of the mass-spring network. In [Müller et al. 2004] authors proposed another hybrid method, in which the fluid is represented by a SPH approach and the solid is represented by polygonal meshes. To model the solid-fluid interaction it is used virtual boundary particles which are placed on the surface of the solid objects according to Gaussian quadrature rules. Such approach allows the computation of smooth interaction potentials that yield stable simulations at interactive rates.

A subclass of hybrid methods deals with the interaction between fluids and bidimensional manifolds modeled by a lower dimensional (moving) triangulated surface. The fluid model is a continuous one, simulated by Navier-Stokes plus SPH or grid based techniques. These approaches deals with the specific problem of preventing the leaking of fluid across the thin solids. In [Guendelman et al. 2005] it is proposed a ray cast based visibility method to perform this task and a new technique for properly enforcing incompressibility near the triangulated surface. When using the SPH method, robust point face collisions detection algorithms must be used to prevent fluid leaking [Bridson et al. 2002]. In addition, fluid flows can be simulated on $2D$ manifolds, represented by (continuous) subdivision surfaces or unstructured meshes, following traditional [Stam 2003] or LBM approaches [Fan et al. 2005]. Interaction between Navier-Stokes fluids and digital terrain models is another subclass of fluid-surface interaction [Ye et al. 1996]. Fluid equation on height fields, like shallow water equations [Thürey et al. 2006; Kass and Miller 1990], where applied for surface flow simulation. These methods and our technique share the idea of modeling the terrain and water surface as height fields. Besides, a hybrid particle and implicit surface approach to simulating water was proposed in [Foster and Fedkiw 2001], which led to the particle level set method of [Enright et al. 2002].

The interaction fluid-solid can be seen as a simplified case of two-phase systems. This is explored in [Solenthaler et al. 2007] where it is presented an unified SPH framework for the simulation of melting and solidification which can be straightforward adapted for interaction between fluids and deformable solids. The technique uses the SPH method for the simulation of liquids, deformable as well as rigid objects, which eliminates the need to define an interface for coupling different models. Additionally, a new surface reconstruction technique, based on considering the movement of the center of mass, is proposed to reduce rendering errors in concave regions.

Variational approach follows the usual philosophy for strong variational techniques: a functional (the Lagrangian) is defined such that the governing equation is the Euler-Lagrange equation for minimizing that functional. In [Batty et al. 2007] the governing equation is the pressure PDE and the functional computes the total kinetic energy of the system. The solution in this formulation is the divergence-free fluid field and compatible solid velocities that minimizes the total kinetic energy.

Finally, visualization and rendering techniques must be applied to ensure the desired level of realism or visual effect. Realistic ren-

dering methods can properly account for this task through several algorithms including path tracing, bidirectional path tracing [Heckbert 1990], Metropolis light transport [Veach and Guibas 1997], and photon mapping [Jensen and Christensen 1998]. The interested reader is also encouraged to browse interesting reviews in this area [Adabala and Manohar 2002; Iglesias 2004].

In this paper we focus on the interaction between fluids and deformable $2D$ manifolds modeled, respectively, by a lattice based technique and mass-spring systems. The work is based on a fluid model proposed in [Barcellos et al. 2007] that shares the basic features of the HPP and FHP models. Therefore, we do not apply PDEs, LBM or shallow water methods and the fluid evolution is computed by integer arithmetic. The mass-spring system is simulated by the GPU implementation presented in [Mosegaard and Sorensen 2005]. Next, we describe the proposed model and its GPU implementation.

3 Particle-Based Simulation Model

In [Barcellos et al. 2007] we propose a particle based model for flow simulation over static surfaces which is inspired in a lattice gas model [Frisch et al. 1987]. In this paper we modify that model for surface flow simulation over deformable $2D$ manifolds which have a global parameterization $\varphi : D \rightarrow \mathbb{R}$ where $\varphi(x, y, t)$ is the elevation of the surface at point (x, y), at the time t. The extension for general manifolds is discussed in Section 6.

The fluid-solid interaction model is composed by three basic elements: (1) A discretization of φ, composed by a regular lattice and the value of φ in each lattice node (i, j); (2) A mass-spring system to model the elastic surface; (3) A particle based water flow simulation.

The fluid model uses a cellular automaton approach and is based on the $2D$ regular lattice, a particle system and local rules for particles displacements. The Figure 1 illustrates the surface domain and the regular lattice which resolution is the same of that one used in the φ discretization. In this figure we highlight a (i, j) node of the lattice and its 4 neighbors given by $(i-1, j)$, $(i, j-1)$, $(i+1, j)$ and $(i, j+1)$, numbered V_1, V_2, V_3 and V_4, respectively. Besides, we can estimate the surface slope at point $(x, y) \in D$ through the projection of the surface normal over the domain D.

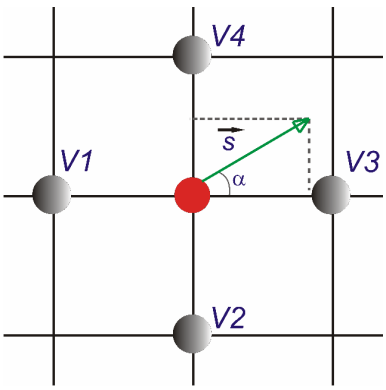


Figure 1: *Neighborhood of a lattice node.*

The particles movement are discrete in space and time; that means, each particle moves from one lattice node to another one in the neighborhood, in the time step $\triangle t$. Specifically, from the lattice topology in Figure 1, we have four directions to consider for the red node (i, j), which can be indexed as $\{1, 2, 3, 4\}$. So, we score these directions, as described by Algorithm 1. The output of this algorithm is the list *Flow Directions*, which receives the four directions sorted according to the priorities for the flow; that means, we have the first, second, third and forth flow direction.

A particle moves from a node (i, j) to a nearest neighbor $V_k = (m, n)$ if k is the first element of *Flow Directions* that satisfies $\mathbf{f}(V_k, t) < \mathbf{f}(i, j, t)$. Besides, a four bit string $n_1(i, j)$, $n_2(i, j)$,..., $n_4(i, j)$ is assigned to each node (i, j) of the lattice. We set

Algorithm 1 : Build Flows Directions ()

```
Initialization:
    L = {1, 2, 3, 4}
while L not null; do
    m ← arg min_k {arc(s⃗, V_k − (i, j)), k ∈ L};
    Remove m from L;
    Insert m in Flow Directions;
end while
```

$n_k(i, j) = 1$ if the node (i, j) has one particle to send to the neighbor k. In the actual implementation each lattice node can send at most one particle at a time, and consequently, can receive at most 4 particles in each simulation time.

For each grid node (i, j) a particle counter is associated, which is used to define the high of the free surface of the flow. The particle system is used in order to update the field of counters. The result is a function

$$f(i, j, t) = \varphi(i, j) + \beta \cdot counter(i, j, t), \tag{1}$$

which gives the elevation of the free surface flow $\mathbf{f}$, at the point (i, j) in the simulation time t (β is a scale parameter).

In this work, the surface deforms as a mass-spring system, subject to internal forces (elastic and damping ones) and external forces. Therefore, the surface elevation φ and slope field $\vec{s}$ are non-stationary fields (as well as the $\mathbf{f}$ in Expression (1)).

The mass-spring system follows the reference [Mosegaard and Sorensen 2005]. The surface nodes works as masses and the edges defines the linear springs with damping. So, given a particle i with mass m_i and position vector $\mathbf{x}_i$, the force system is composed by the elastic ($\mathbf{f}^i_{elastic}$), gravitational ($\mathbf{f}^i_{grav}$) and damping ($\mathbf{f}^i_{damp}$) forces, defined respectively, by:

$$\mathbf{f}^i_{elastic} = \sum_{j=1}^{4} k_{ij} \left(l_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\|\right) \frac{(\mathbf{x}_i - \mathbf{x}_j)}{\|\mathbf{x}_i - \mathbf{x}_j\|}, \tag{2}$$

where k_{ij} is the stiffness of the spring linking the nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ and l_{ij} the spring rest length;

$$\mathbf{f}^i_{grav} = (\beta_i counter(i))\mathbf{z} + m_i \mathbf{g}, \tag{3}$$

$$\mathbf{f}^i_{damp} = \gamma_i \dot{\mathbf{x}}_i, \tag{4}$$

where β_i is a scale parameter, $\mathbf{g}$ is the gravity field, γ_i is the damping factor and $counter(i)$ holds the number of particles accumulated in the corresponding position. Following Newton's Laws, we get the following evolution equation:

$$m_i \ddot{\mathbf{x}}_i = \mathbf{f}^i_{elastic} + \mathbf{f}^i_{damp} + \mathbf{f}^i_{grav}. \tag{5}$$

This system of ordinary differential equations can be efficiently solved by the Verlet integration technique [Mosegaard and Sorensen 2005]:

$$\mathbf{x}_i\,(t + h) = 2\mathbf{x}_i\,(t) - \mathbf{x}_i\,(t - h) + \ddot{\mathbf{x}}_i\,(t)\,h^2. \tag{6}$$

Now, let us put the fluid model and the mass-spring together to render the basic algorithm. At the initialization, the particle counters field, the initial surface geometry and velocity of the nodes must be defined. Then, in the simulation loop, the forces in Expressions (2)-(4) are computed and the differential Equation (5) is integrated through the Verlet scheme in Equation (6). Next, the new normal field is computed and projected to get the new field $\vec{s}$. Then, the Algorithm 1 is applied to obtain the *Flow Directions* field. Based on the free surface high and on the *Flow Direction*, the n_k field is built. This field is used to compute the number of particles that each node

(i, j) sends and receives from the neighbors, and, consequently, the particles counter field is updated. Finally, the deformable surface and the surface flow are visualized and the next simulation step starts. At each interaction, we re-compute the free surface elevation, given by expression (1), in order to mimic the transference of velocity from the terrain to the particles.

Besides, we can include physical effects like evaporation $E(i, j, t)$ and precipitation $P(i, j, t)$ by using an intuitive balance model given by:

$$counter(i, j, t) \leftarrow counter(i, j, t) + (P - E). \qquad (7)$$

This model is based on local rules which try to simulate some aspects of overland flows [Ye et al. 1996].

We shall emphasize that the fluid simulation algorithm does not suffer from numerical stability issues because the floating-point operations are simple (normal field and computation of expressions (1)) and the update of the counter field is based on simple comparisons and integer arithmetic. Besides, volume conservation is straightforward verified because all operations are conservative with respect to the number of particles.

4 GPU Implementation

The algorithm described in Section 3, for surface flow simulation over deformable surfaces, is based on a regular lattice and local rules for surface deformation and fluid animation. Therefore, it may be very suitable for a GPU implementation. The main idea is to encode the information needed in the simulation as textures into the video memory. The Algorithm 2 gives an overview of the whole GPU processing. The initialization step computes the initial values to the nodes particle counters, the surface geometry and the initial velocity to each node. These values are computed in the CPU and stored in different matrixes. Then, the simulation loop starts, performing, for each time step, the simulation and visualization processes. The simulation process update of the surface geometry, based on the mass-spring simulation, and the fluid simulation updates the fluid particles distribution among the lattice nodes. Finally, the visualization step begins, drawing the surface and the water surface. The loop process, involving simulation and visualization is completely performed on the GPU.

Algorithm 2 : GPU Simulation and Visualization()

```
Initialization:
   Compute
      Particles Counters.
      Surface Geometry.
      Velocity.
   Transfer Data to GPU.
loop
   Simulation:
      GPU-Based Mass-Spring Simulation();
      GPU-Based Fluid Simulation();
   Visualization:
      Draw Mass-Spring Surface;
      Draw Water Free Surface;
end loop
```

Like others GPU-based algorithms, all data needed to simulation and visualization processes are stored in GPU memory, mapped as textures. The data flow of algorithm 2 is illustrated by Figure 2.

The GPU-based mass-spring simulation follows the reference [Mosegaard and Sorensen 2005]. At each interaction of the main loop a $2D$ texture, represented in Figure 2 as **Properties texture**, encoding the automaton configuration is generated as follows. To each texture point (i, j) it is associated the index k, such that each color channel represents a different data: the terrain elevation $\varphi(i, j)$ and a particle counter, respectively channels R and B. The connectivity among masses can be implicitly represented because surface samples are organized in a regular lattice with a simple fixed 4-connected neighborhood rule. **Surface** textures stores the particles positions. In fact, as long as Verlet integration method is used (equation 6)), three surface stages must be stored during the simulation, each one representing a different time step: current (t), backward $(t-1)$ and forward $(t+1)$. The output of the spring-mass simulation includes the new surface geometry $(t + 1)$ and the **flow** texture. The **Flow** data encodes the four flow directions in each node (i, j) of the lattice, as described in section 3 and algorithm 1. Based on the flow directions and the particle counters from previous simulation step, the fluid simulation begins.

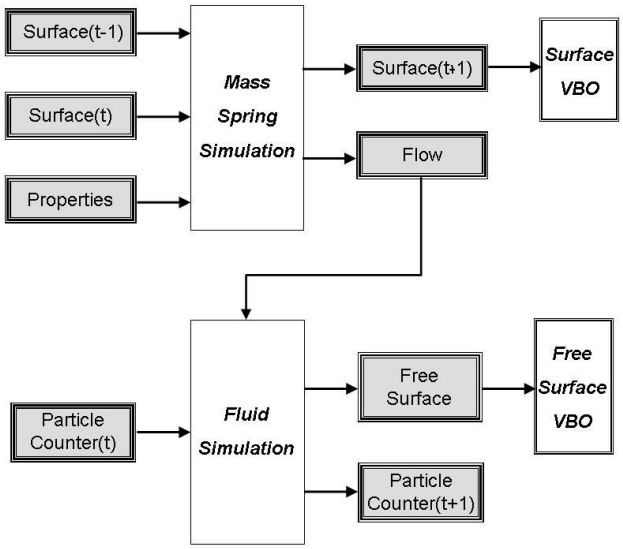


Figure 2: *The data flow involved in the simulation and visualization processes. The gray boxes represent data mapped in GPU using texture memory.*

During the fluid simulation an **Upward Flow** is build, where the $n_k(i, j)$ bit string associated with each lattice node is generated as described in section 3. In this process, given a lattice node (i, j), we sequentially check the Flow Direction texture values and take the first one, say k, that satisfies $\mathbf{f}(V_k, t) < \mathbf{f}(i, j, t)$.

Using the **Upward Flow** and the **Particle Counter** at interaction t, a new **Particle Counter** is generated at time $t + 1$. The elevations of the free surface flow $\mathbf{f}(i, j, t)$ at each (i, j) node of the lattice is also calculated, as Figures 2 illustrate.

The particle counters are updated evaluating the number of particles that the node (i, j) receives and sends, respectively. The value of received particles in a node (i, j) is computed by adding the suitable values of the **Upward Flows** at the neighbors (V_1, V_2, V_3, V_4). The number of particles the node (i, j) sends is obtained by adding the $n_k(i, j)$ values.

The elevation of the free surface at a lattice node (i, j) is finally obtained by adding the terrain elevation and the particle counter value, weighted by a scale factor at that node (equation 1). The new elevation is used to generate the **Free Surface** texture.

The two textures **Surface**$(t+1)$ and **Free Surface**, which contains, respectively, the terrain and the water surfaces, must be sent to an *Frame-Buffer Object (FBO)* [Woo et al. 1999], in order to be used to update data for visualization process.

It is important to emphasize that we need a new **Particle Counter Texture** to encode the configuration at time $t + 1$ due to the restriction of the shader-based implementation, using GLSL (OpenGL Shading Language) [Rost 2004]. In context of a shader, a texture can be strictly *read-only* or *write-only*, and the GPU parallel architecture uses several processing units to compute a field. Also, the fragment processors are able to update more than one texture in parallel through the technique called *Multiple Render Target* [Rost 2004].

The **Surface** and the **Free Surface** textures are used as input to the *Vertex Buffer Object* (**VBO**). The **VBO** will be managed as a *Vertex Array* to render the final visualization of the free surface of the

fluid. Therefore, the whole computation (simulation and visualization) is performed in the GPU. This scheme allows us to minimize the information flow between the CPU and the GPU and improves the performance of the application.

5 Experimental Results

We developed a software in $C++$ language that allows a user to deposit densities on the surfaces as well as to control parameters of the simulation. The pictures included in this paper are snapshots obtained from that program. The corresponding videos can be found in: `http://virtual01.lncc.br/~barcellos/videosSBGames2009.zip`. The rendering is implemented by standard OpenGL calls and the shaders with OpenGL Shading Language. The experiments were performed in a Intel Core 2 Duo 2.66 GHz, with 4 GB of RAM and a Video Card NVidia GeForce 8800 GTX, running Windows XP.

In these experiments we highlight aspects of our model that can be useful for computer graphics applications: simple to simulate complex configurations and computational efficiency.

We apply 2 kinds of external forces: vertical ones and wave-like forces (Section 5.1). Surface configurations include multiply connected domains (Section 5.2) and complex geometries (Section 5.3). Besides we simulate effects like evaporation and precipitation (Section 5.3) and present the CPU and GPU performance comparison (section 5.4).

The lattice resolution is 256×256 and the number of particles used in each test is given in Table 1. The intensity of the vertical forces are scaled according to the requirements of each example. The masses and springs located at the boundary of the domain are kept rigid and we set $m = 1.0$ everywhere. The time step h of the simulation was set to $h = 0.1$. We render the terrain and the water surface as height fields. The visualization was implemented in OpenGL. To increase the scene realism we applied the environment mapping technique and Fresnel effects to incorporate the physical laws of reflection and refraction for the water free surface rendering. The shaders was implemented in OpenGL Shading Language.

Table 1: *Number of particles for the experiments of Sections 5.1, 5.2, 5.3.*

Configuration	Particles
Figure 4	2377092
Figure 5	2005169
Figure 6	218700
Figure 7	218700
Figure 8	364500
Figure 9	211200
Figure 10	53550
Figure 11	218700

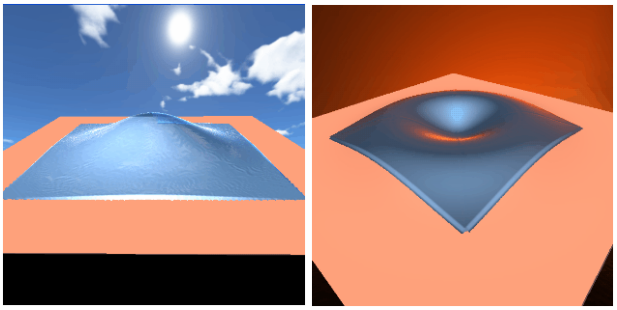
5.1 Vertical Forces

In this section we consider elastic surfaces over the influence of vertical external forces, like the gravitational one, that are applied during a time interval $[t_1, t_2]$ along the simulation, as follows:

$$\mathbf{f}_v(t) = \begin{cases} T \cdot \mathbf{z} & : \quad t_1 \leq t \leq t_2 \\ \mathbf{0} & : \quad otherwise \end{cases} \tag{8}$$

where $\mathbf{z}$ is the unitary vector in the vertical direction and T is a scale factor. The Figures 3.a-b show the initial configurations used in the experiments. Each spring has a spring rest length l_{ij} (see Expression (2)) equal to its length at the initialization.

The Figure 4 shows some snapshots of the animation which is generated starting from the initial configuration pictured in Figure 3.a. In this case, we apply a force given by Expression (8) with $T = 0.025$ $t_1 = 0$, $t_2 = 3200$. Then, we turn off this force and apply a descending one, given by $T = -0.025$ for $t \geq 3201$. The mass-spring parameters are: $\gamma = 0.05$ (damping); $k = 1.0$ (stiffness). Besides force (8) we add a gravitational force computed by:



(a) (b)

Figure 3: *Initial configurations that are used in the computational experiments: (a) Flat surface with a gaussian fluid distribution. (b) Mexican Hat surface with a uniform fluid distribution.*

$$\mathbf{f}^i_{grav} = (\rho * counter(i))\mathbf{z}. \tag{9}$$

with $\rho = -0.00005$. At the beginning of the simulation, we observe that the fluid remains at the center of the deformable surface because the surface deformation generates a lake (Figure 4.a). The force $\mathbf{f}_v$ deforms the surface in the upward direction generating a pyramid-like surface due to the fact that the surface boundary is rigid, as it can be seem in Figures 4.a-b. The descending force is applied and soon the surface starts a contraction process. The Figure 4.c shows the configuration at time $t = 4650$ in which the fluid spills out the top of the surface and deforms the surface at the same time. We observe a fully coupled interactions between three-dimensional deformation and fluid evolution. The surface geometry is modified resulting a fluid redistribution which then feed back and influence subsequent deformation. For instance, in Figure 4.d we observe the system configuration at $t = 5950$ showing new regions of fluid accumulation due to surface deformation. In these examples, the fluid particles are not allowed to go out the simulation domain. So, particles accumulate in the boundary nodes until the surface slope points inside the domain. That is why we observe some accumulations of fluid in the boundary of the surface.

Figure 5 shows the evolution of the same initial configuration depicted on Figure 3.a. However, in this case we set the damping zero and apply the force (8) at time $t = 750$ with $T = -20.0$. Each spring has a spring rest length l_{ij} (see Expression (2)) equal to its length at the initialization and $k = 50.0$.

Figures 5.a-b depicts the time step in which the fluid volume is suspended by the surface generating interesting effects of transparency and downhill flows. This configuration needs special comments. The mechanical behavior acts like a viscous fluid, such a gelatin, which flows slowly and generates pics of accumulations like the one observed. This is a consequence of our heuristic for fluid particles motion which imposes a constant horizontal velocity for particles displacements: the velocity of the particle projection is equal the ratio between the lattice edge and the time step which are constants in the model. Figure 5.c depicts the system configuration at interaction $t = 3440$. We observe a lake formation at the center of the surface as well as a large portion of fluid going down towards the central lake. The configuration shown in Figure 5.d occurs at time step $t = 4200$ and presents a portion of the fluid volume suspended by the surface, similarly to Figure 5.a. In fact, the video sequence shows periodic formations as a consequence of the symmetries in the initial configuration, the fact that we do not consider friction between the fluid and the surface and that the damping is zero in this case.

The Figure 6 shows a sequence of snapshots generated from the initial configuration depicted in Figure 3.b. We applied the force (8) with $T = -30.0$ at time $t = 1000$ and $T = 35.0$ at time $t = 3200$. The mass-spring parameters are: $\gamma = 0.075$ (damping); $k = 20.0$ (stiffness), l_{ij} is the length at the initial step. Besides

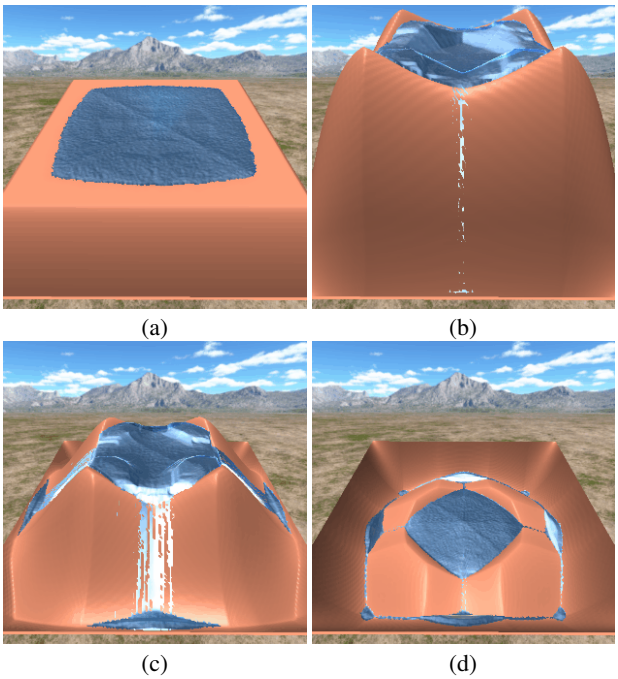


Figure 4: *(a) Flat surface with a gaussian fluid distribution at time step $t = 1400$. (b) The spring-mass system evolves generating a pyramid-like figure due to the fact that springs at the domain boundary are kept rigid and the force given by Expression (8) is applied. (c) Down hill surface flow at simulation time $t = 4650$. (d) Configuration at time $t = 5950$.*

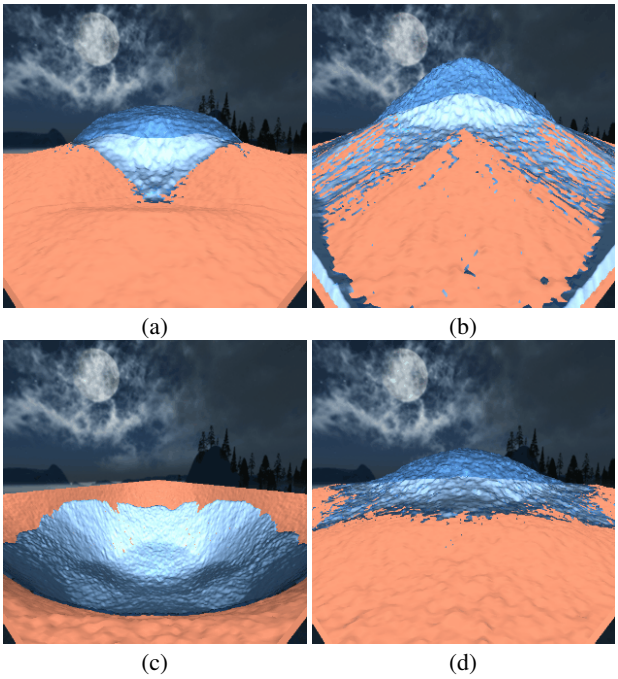


Figure 5: *Evolution of the initial configuration depicted in Figure 3, but without damping. (a) Frame at time step $t = 1910$. (b) Configuration with the fluid volume suspended by the elastic surface (time $t = 2610$); (c) Lake formation at time $t = 3440$. (d) System evolution at time $t = 4200$. Energy conservation and symmetries of the initial configuration implies periodic structures during the evolution.*

force (8) we add a gravitational force computed by expression (9) with $\rho = -0.0000025$.

The Figure 6.a shows a portion of the fluid flowing towards the boundary and another portion that generates a lake in the valley of the surface. The Figure 6.b shows this configuration some time further ($t = 448$) now depicting the fluid accumulation nearby the surface boundary and at the center. The system evolves and achieves the configuration shown in Figure 6.c in which the fluid is concentrated nearby the boundary and in the valley. Then, at the time $t = 1000$ the force (8) is applied with $T = -30.0$ deforming the system which achieves the configuration pictured on Figure 6.d. When the mass-spring system achieves zero kinetic energy, it starts to go up achieving the configurations shown in Figure 6.e and 6.f at times $t = 4648$ and $t = 7084$, respectively.

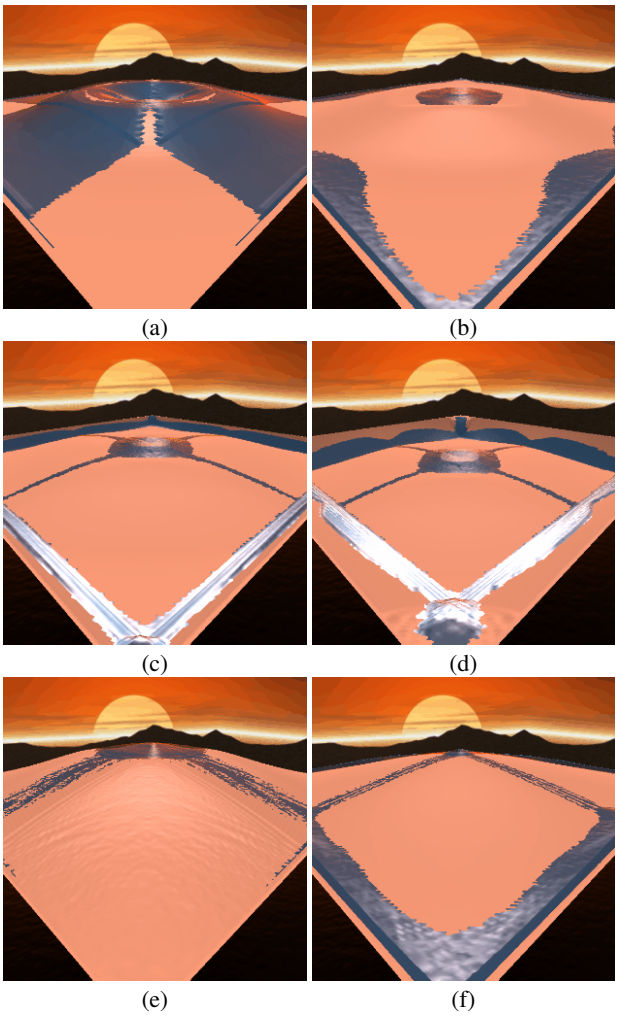


Figure 6: *(a) Mexican hat surface with a uniform fluid distribution at time step $t = 84$. (b) Configuration at time $t = 448$ showing the deformation of the surface nearby the surface valley; (c) Fluid concentration nearby the domain boundary and in the valley at time 1092. (d) Configuration at $t = 1232$ generated after the descending force is applied. (e) System in ascending movement: configuration at time $t = 4648$. (f) Snapshot obtained at time step 7084 of the simulation.*

The Figure 7 shows some snapshots for the evolution of the initial configuration depicted in Figure 3.b, but without damping. In this case we set the spring rest length $l_{ij} = 1.0$ at the initialization. The mass-spring parameters are: $\gamma = 0.0$ (damping); $k = 50.0$ (stiffness). We apply the force (8) with $T = -40.0$ at time $t = 1500$ and we add a gravitational force computed by equation (9) with $\rho = -0.00025$.

Once the rest length of the springs is $l_{ij} = 1.0$, the elastic energy at time $t = 0$ is not null. That is way we observe a little hill formation at the center in Figure 7.a, which dissipates as shown in Figure 7.b.

At time $t = 1500$ it is applied the force (8) with $T = -40.0$ generating the configuration pictured on Figure 6.c at time $t = 2260$. When the kinetic energy becomes null, the system starts ascending movement generating the configuration pictured on Figures 7.d.

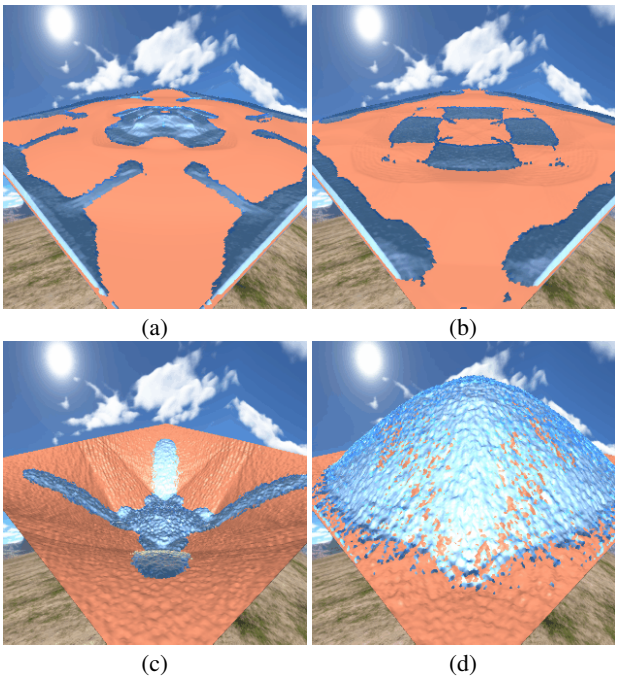


Figure 7: *(a) Mexican hat surface with a uniform fluid distribution at time step* 364*. (b) Configuration at time 560; (c) 2260. (d) 2772.*

The next example applies a force given by a periodic wave form namely a harmonic wave:

$$f_w(x, y, t) = A_0 \sin(bx - \omega t), \qquad (10)$$

where b is the wave number and ω is the angular frequency. This expression represents an unaltered propagation through a linear media in the electromagnetic theory [Corson and Lorrain 1970]. The parameters are: $\gamma = 0.2$ (damping), $k = 20.0$ (stiffness), $b = \omega = 0.1$ and $A_0 = 0.375$. We turned off the gravitational force ($\rho = 0.0$ in expression (9)).

The obtained patterns of the surface flow can be seen in the Figure 8. The initial configuration is similar the one pictured on Figure 3.a but now we take a uniform fluid distribution at the time $t = 0$. As we turned off the influence of the weight of the fluid in the spring-mass system we avoid surface deformation due the fluid weight. So, the fluid is carried by the surface waves spreading a bit due to fluid concentration (Figure 8).a. In this case the fluid is allowed to flow out the surface domain. So we do not have mass conservation, as it can be observed on Figure 8.b,d.

5.2 Complex Domains

Let us consider the simulation technique for a multiply connected domain, like the one pictured in Figure 9.a. To define the initial configuration it was generated a circular hole with radius 35.0 on the surface and an uniform fluid distribution composed by 211200 particles. We use a texture channel in order to distinguish the lattice nodes that are inside the surface domain from the outside nodes.

In this case, we have a boundary with two disconnected components. It is not required any extra mathematical machinery to deal with such topology because system rules do not undergo modifications. The parameters are: $\gamma = 0.125$ (damping), $k = 5.0$ (stiffness). We apply the force (8) with $T = -30.0$ at time $t = 200$, $T = 45.0$ at time $t = 1200$, $T = -45.0$ at time $t = 2900$ and we add a gravitational force computed by expression (9) with $\rho = -0.0000025$.

Figure 9.b shows the fluid distribution at time $t = 1120$ when system is going down. We observe a lake formation and the spring

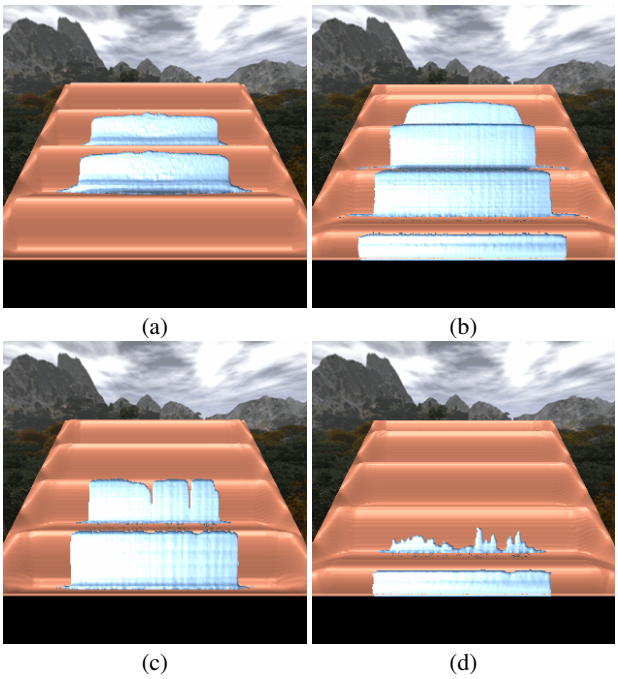


Figure 8: *(a) Flat surface with a uniform fluid distribution and harmonic waves at time step* 220*. (b) Configuration at time* 1150 *with fluid going out the domain; (c) Surface flow pattern at simulation time* 5040*. (d) Snapshot at time step* 6104*.*

mass deformation. Then, we apply the ascending force, at the time $t = 1200$. The Figure 9.c pictures a snapshot of the corresponding sequence showing a surface flow towards the surface hole. In this case the fluid is not allowed to go out the surface. Finally, at time $t = 2900$ we apply the force (8) with $T = -45.0$. As a result, the fluid nearby the inner boundary flows out and lake formations are observed, as it can be seen in Figure 9.d. If we want to predict such effects, we need to consider Navier-Stokes equations with suitable boundary conditions. However, if the aim is to explore the visual effect, we can just simulate and take the desired result at its time.

5.3 Erosion and Deformation

This section illustrates the erosion effects that can be obtained with the proposed technique. The mechanical behavior of the inviscid substrate under erosion is modeled by the spring-mass system coupled with the surface flow. The system dynamics generates a fluid redistribution that indicates high consequence areas and influence subsequent erosion modeled as deformation. A similar approach was presented in [Simpson 2004] but using a continuous thin-plate formulation.

The Figure 10 illustrates an example of erosion simulation caused by a constant rainfall. The original surface topography is pictured on Figure 10.a. We model the rainfall as a precipitation distribution over the terrain, or some part of it, as depicted in Figure 10.b. The precipitation is constant and applied during the first time step ($P(i, j, 0) = 12$ in Equation (7)). There is no evaporation ($E(i, j, t) = 0$ in expression (7)) and the parameters are: $\gamma = 0.15$ (damping), $k = 1.0$ (stiffness).

We observe in Figures 10.c-e the evolution of the terrain topography and the formation of flooded areas over the terrain. The Figure 10.f pictures the surface topography at time $t = 5300$. We can compare it with the initial configuration in Figure 10.a and observe the deformation generated.

The model can predicts high erosion potential at the lower, concave parts of hillslopes. If an almost flat area is encountered then the water simply spreads out into the flat area, deforming it, without any extra machinery. We observe this effect in the region nearby the left-hand corner of Figures 10.c-d. We shall observe that the particle model does not incorporates dissipative forces between the flow

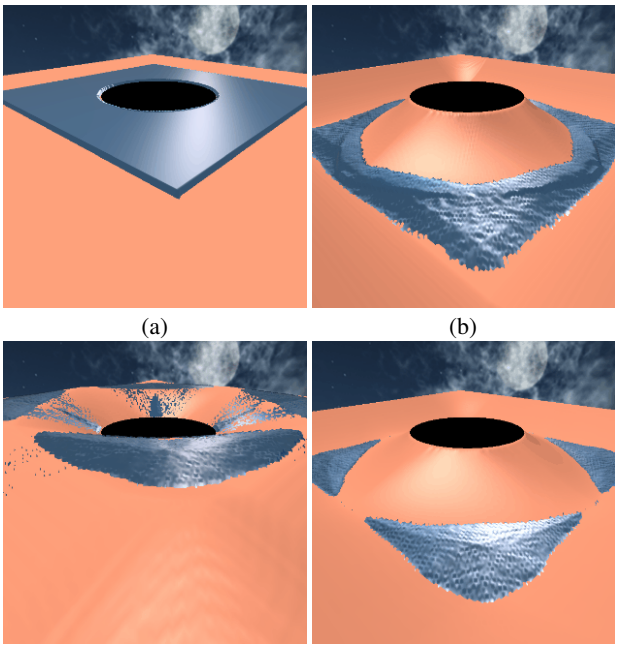


Figure 9: *(a) Multiply connected domain: Initial configuration. (b) System is going down at time step* 2380 *due to the application of force (8) with* $T = -30.0$*; (c) Configuration at time step* 2380 *when system is going up. (d) Snapshot at time step* 4676 *when system is going down again due to application of force (8) with* $T = -45.0$*.*

and the terrain. Besides, mass transport effects are not considered in our model.

Also, we can apply Expression (7) to include evaporation in the simulation. The Figure 11 shows six snapshots of such animation over the Grand Canyon topography, depicted in Figure 11.a. The evaporation is constant ($E(i, j, t_0 + \Delta t) = 1$), for all $(i, j) \in D$, with $t_0 = 2800$ and $\Delta t = 40$. The precipitation is modeled like in the previous example. The Figure 11.b is the initial step of the simulation. Figures 11.c-e show the fluid evolution and the surface deformation. In this case, there is no mass conservation due to the evaporation. So, the simulation stops when the fluid mass is null, that means, in the time step t^* in which the particle counters are zero: $counter(i, j, t^*) = 0$. Figure 11.f shows a snapshot closer to the final step. The parameters are: $\gamma = 0.15$ (damping), $k = 1.0$ (stiffness). In this example as well as in the Figure 10 each spring has a rest length given by its length at the time $t = 0$. When the fluid volume becomes null, the spring-mass will evolve towards its equilibrium configuration. That is way we stop the evolution when the fluid vanishes. More efficient mechanisms must be implemented to control the deformation in order to mimic erosion more efficiently.

5.4 CPU versus GPU Animation and Real Time

In this section we compare the performance of the CPU implementation with the performance of the GPU implementation of the animation algorithm. The Table 2 shows the rate of frames per second (**FPS**) obtained through these implementations, for the above examples. We observe a substantial improvement in the performance (more than 13 times better in all cases) with frame rates suitable for real time applications.

A major concern about using graphics hardware for general computation is the accuracy. The graphics hardware used supports 4 bytes per color channel which fits the requirement on the accuracy of the computation.

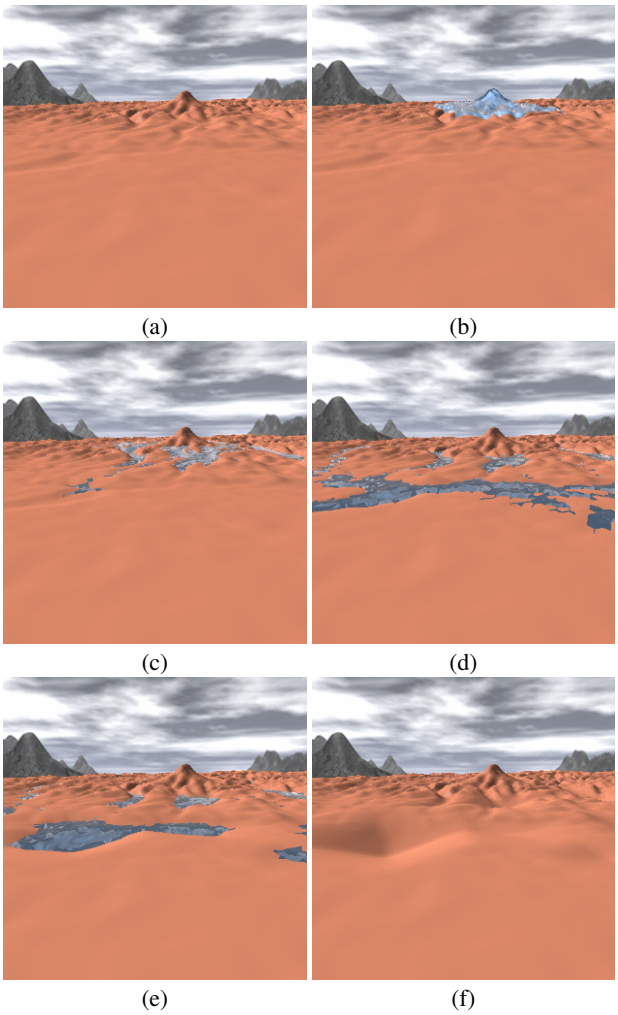


Figure 10: *(a) Puget Sound Model. (b) Fluid precipitation; (c) Modified terrain topography and surface flow at time* 196*. (d) Configuration at time step* 1500*. (e) Configuration at time step* 3800*. (f) Snapshot at time step* 5300 *showing the deformed surface topography.*

Table 2: *Frames per second rates (FPS) and number of particles for the experiments of Sections 5.1, 5.2, 5.3.*

Configuration	Particles	FPS in CPU	FPS in GPU
Figure 4	2377092	5.7	81.6
Figure 5	2005169	5.6	78.3
Figure 6	218700	5.5	83.2
Figure 7	218700	5.4	81.6
Figure 8	364500	5.3	81.6
Figure 9	211200	5.7	82.0
Figure 10	53550	5.7	78.4
Figure 11	218700	5.4	83.2

Table 3: *This table shows the FPSs for different numbers of particles when we use the same configuration of Figure 5.*

Configuration	Particles	FPS in CPU	FPS in GPU
Figure 5	2138306	5.5	79.9
Figure 5	706754	5.5	80.0
Figure 5	229522	5.5	80.1

6 Conclusions

In this work, we focused on surface flow animation in deformable surfaces described by mass-spring models for computer graphics applications. We proposed the combination of an efficient particle model for fluid simulation with a mass-spring approach to perform the animation. Both the CPU and GPU implementations were con-

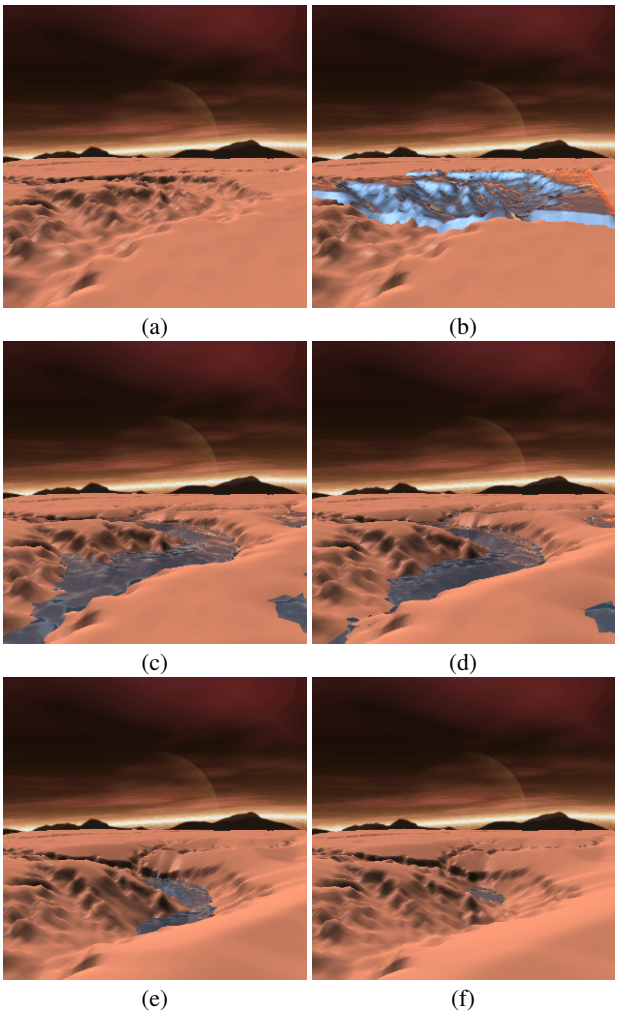


Figure 11: *(a) Grand Canyon model. (b) Initial precipitation area; (c) Snapshot at simulation time* 1568 *showing fluid distribution and surface deformation. (d) Configuration at time step* 2500*. (e) Snapshot at time step* 4060*: we observe the loss of fluid volume due to evaporation. (f) Terrain topography when fluid mass almost becomes null.*

sidered. In the experimental results we emphasize the abilities of our animation technique and the real time capabilities of the GPU implementation. Besides, we notice that the bottleneck of the whole simulation is the mass-spring system. In fact, as we can observe in Table 3, when the number of particles used is 2138306, 706754 or 229522, for the configuration picture on Figure 5, the measured FPS is almost the same.

Future directions in this work are the extension of the combined technique for general $2D$ manifolds represented by subdivision surfaces with local parameterization with special care to handle cross-patch boundary conditions. Besides, the introduction of random variables, to incorporate viscosity, and the comparison with a PDE-based technique will be performed soon.

Acknowledgements

The authors would like to thank the support provided by PCI-LNCC for this work.

References

ADABALA, N., AND MANOHAR, S. 2002. Techniques for realistic visualization of fluids: A survey. *Comput. Graph. Forum 21*, 1, 65–81.

BARCELLOS, B., GIRALDI, G. A., SILVA, R. L., APOLINRIO, A. L., AND RODRIGUES, P. S. S. 2007. Surface flow animation in digital terrain models. In *SVR 2007 - IX Symposium on Virtual an Augmented Reality*.

BATTY, C., BERTAILS, F., AND BRIDSON, R. 2007. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. 26*, 3.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. In *SIGGRAPH '02*, 594–603.

BUICK, J. M., EASSON, W. J., AND GREATED, C. A. 1998. Numerical simulation of internal gravity waves using a lattice gas model. *Int. J. Numer. Meth. fluids 26*, 657–676.

CHEN, S., AND DOOLEN, G. D. 1998. Lattice boltzmann method for fluid flows. *Annual Review of Fluid Mechanics 30*, 329–364.

CHENTANEZ, N., GOKTEKIN, T. G., FELDMAN, B. E., AND O'BRIEN, J. F. 2006. Simultaneous coupling of fluids and deformable bodies. In *SCA '06: Proc. of the 2006 ACM SIGGRAPH/Eurographics*, Eurographics Association, 83–89.

CORSON, D., AND LORRAIN, P. 1970. *Electromagnetic fields and waves*. W.H. Freemann, New York, USA.

DEBUNNE, G., CANI, M.-P., DESBRUN, M., AND BARR, A. 2000. Adaptive simulation of soft bodies in real-time. In *CA '00: Proc. of the Comp. Animation*, IEEE Computer Society, 15.

DEBUNNE, G., DESBRUN, M., CANI, M.-P., AND BARR, A. H. 2001. Dynamic real-time deformations using space & time adaptive sampling. In *SIGGRAPH '01*, 31–36.

DELINGETTE, H. 2008. Triangular springs for modeling nonlinear membranes. *IEEE Trans. on Vis. and Comp. Graph. 14*, 2, 329–341.

DESBRUN, M., AND GASCUEL, M.-P. 1996. Smoothed particles: a new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, Springer-Verlag New York, Inc., New York, NY, USA, 61–76.

DEUSEN, O., EBERT, D. S., FEDKIW, R., MUSGRAVE, F. K., PRUSINKIEWICZ, P., ROBLE, D., STAM, J., AND TESSENDORF, J. 2004. The elements of nature: interactive and realistic techniques. In *ACM SIGGRAPH 2004 Course Notes*, 32.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. *ACM Trans. Graph. 21*, 3, 736–744.

ETZMUSS, O., GROSS, J., AND STRASSER, W. 2003. Deriving a particle system from continuum mechanics for the animation of deformable objects. *IEEE Trans. on Vis. and Comp. Graph. 9*, 4, 538–550.

FAN, Z., ZHAO, Y., KAUFMAN, A., AND HE, Y. 2005. Adapted unstructured lbm for flow simulation on curved surfaces. In *Proc. of the 2005 ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.*, 245–254.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *SIGGRAPH '01*.

FOSTER, N., AND METAXAS, D. 1997. Modeling the motion of a hot, turbulent gas. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 181–188.

FRISCH, U., D'HUMIÈRES, D., HASSLACHER, B., LALLEMAND, P., POMEAU, Y., AND RIVET, J.-P. 1987. Lattice gas hidrodynamics in two and three dimension. *Complex Systems*, 649–707.

GENEVAUX, O., HABIBI, A., AND DISCHLER, J.-M. 2003. Simulating fluid-solid interaction. In *Graphics Interface*.

GUENDELMAN, E., SELLE, A., LOSASSO, F., AND FEDKIW, R. 2005. Coupling water and smoke to thin deformable and rigid shells. In *SIGGRAPH '05*, 973–981.

HECKBERT, P. S. 1990. Adaptive radiosity textures for bidirectional ray tracing. In *SIGGRAPH '90*, 145–154.

HOUSE, D. H., AND BREEN, D. E., Eds. 2000. *Cloth Modeling and Animation*. A K Peters, Ltd.

IGLESIAS, A. 2004. Computer graphics for water modeling and rendering: a survey. *Future Gener. Comput. Syst. 20*, 8, 1355–1374.

JAMES, D. L., AND PAI, D. K. 1999. Artdefo: accurate real time deformable objects. In *SIGGRAPH '99*, ACM Press/Addison-Wesley Publishing Co., 65–72.

JENSEN, H. W., AND CHRISTENSEN, P. H. 1998. Efficient simulation of light transport in scences with participating media using photon maps. In *SIGGRAPH '98*, ACM, 311–320.

JUDICE, S. F., BARCELLOS, B., AND GIRALDI, G. 2008. A cellular automata framework for real time fluid animation. In *Proceedings of SBGames 08: Computing Track.*, 169–176.

KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90*, 49–57.

KEISER, R., ADAMS, B., GASSER, D., BAZZI, P., DUTRE, P., AND GROSS, M. 2005. A unified lagrangian approach to solid-fluid animation. In *Point-Based Graphics, 2005. Eurographics/IEEE VGTC Symposium Proceedings*, 125–148.

LIU, G. R., AND LIU, M. B. 2003. *Smoothed particle hydrodynamics : a meshfree particle method*. World Scientific, New Jersey.

MOSEGAARD, J., AND SORENSEN, T. S. 2005. Gpu accelerated surgical simulators for complex morphology. In *VR '05: Proceedings of the 2005 IEEE Conference 2005 on Virtual Reality*, IEEE Computer Society, Washington, DC, USA, 147–154, 323.

MÜLLER, M., DORSEY, J., MCMILLAN, L., JAGNOW, R., AND CUTLER, B. 2002. Stable real-time deformations. In *SCA '02: Proc. of the 2002 ACM SIGGRAPH/Eurographics*, 49–54.

MÜLLER, M., SCHIRM, S., TESCHNER, M., HEIDELBERGER, B., AND GROSS, M. 2004. Interaction of fluids with deformable solids: Research articles. *Comput. Animat. Virtual Worlds 15*, 3-4, 159–171.

NEALEN, A., MULLER, M., KEISER, R., BOXERMAN, E., AND CARLSON, M. 2005. Physically based deformable models in computer graphics. In *Eurographics 2005, State of The Art Report (STAR)*.

PREMOZE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A., AND WHITAKER, R. 2003. Particle-based simulation of fluids. *Computer Graphics Forum 22*, 3.

ROST, R. J. 2004. *OpenGL(R) Shading Language*. Addison-Wesley Professional, Boston, MA, USA.

ROTHMAN, D. H., AND ZALESKI, S. 1994. Lattice-gas models of phase separation: Interface, phase transition and multiphase flows. *Rev. Mod. Phys 66*, 1417–1479.

SIMPSON, G. 2004. A dynamic model to investigate coupling between erosion, deposition, and three-dimensional (thin-plate) deformation. *JOURNAL OF GEOPHYSICAL RESEARCH 19*.

SOLENTHALER, B., SCHLAFLI, J., AND PAJAROLA, R. 2007. A unified particle model for fluid-solid interactions. *Comput. Animat. Virtual Worlds 18*, 1, 69–82.

STAM, J. 1999. Stable fluids. In *Siggraph 1999*, Addison Wesley Longman, 121–128.

STAM, J. 2003. Flows on surfaces of arbitrary topology. In *SIGGRAPH '03*, 724–731.

TERZOPOULOS, D., AND WITKIN, A. 1988. Physically based models with rigid and deformable components. *IEEE Comput. Graph. Appl. 8*, 6, 41–51.

THÜREY, N., RÜDE, U., AND STAMMINGER, M. 2006. Animation of open water phenomena with coupled shallow water and free surface simulations. In *Proc. of the 2006 ACM SIGGRAPH/Eurographics Symp. on Comp. Anim. (SCA '06)*, 157–164.

VANGELDER, A., AND WILHELMS, J. 1997. Simulation of elastic membranes and soft tissue with triangulated spring meshes. Tech. rep.

VEACH, E., AND GUIBAS, L. J. 1997. Metropolis light transport. In *SIGGRAPH '97*, 65–76.

VOLINO, P., AND MAGNENAT-THALMANN, N. 2000. *Virtual Clothing: Theory and Practice*. Springer.

WEI, X., LI, W., MUELLER, K., AND KAUFMAN, A. 2004. The lattice-boltzmann method for simulating gaseous phenomena. *IEEE Trans. on Vis. and Comp. Graphics 10*, 2, 164–176.

WOO, M., DAVIS, AND SHERIDAN, M. B. 1999. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

YE, Z., MAIDMENT, D., AND MCKINNEY, D. 1996. Map-based surface and subsurface flow simulation models: An object-oriented and gis approach. Tech. rep., Center for Research in Water Resources, University of Texas at Austin, Tec. Report: http://www.crwr.utexas.edu/reports/pdf/1996/rpt96-5front.pdf.

YE, X. W., ZHAO, Y., FAN, Z., LI, W., QIU, F., YOAKUM-STOVER, S., AND KAUFMAN, A. 2004. Lattice-based flow field modeling. *IEEE Trans. on Vis. and Comp. Graphics 10*, 719–729.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Trans. Graph. 24*, 3, 965–972.

GpuWars: Design and Implementation of a GPGPU Game

Mark Joselli
UFF, Medialab

Esteban Clua
UFF, Medialab

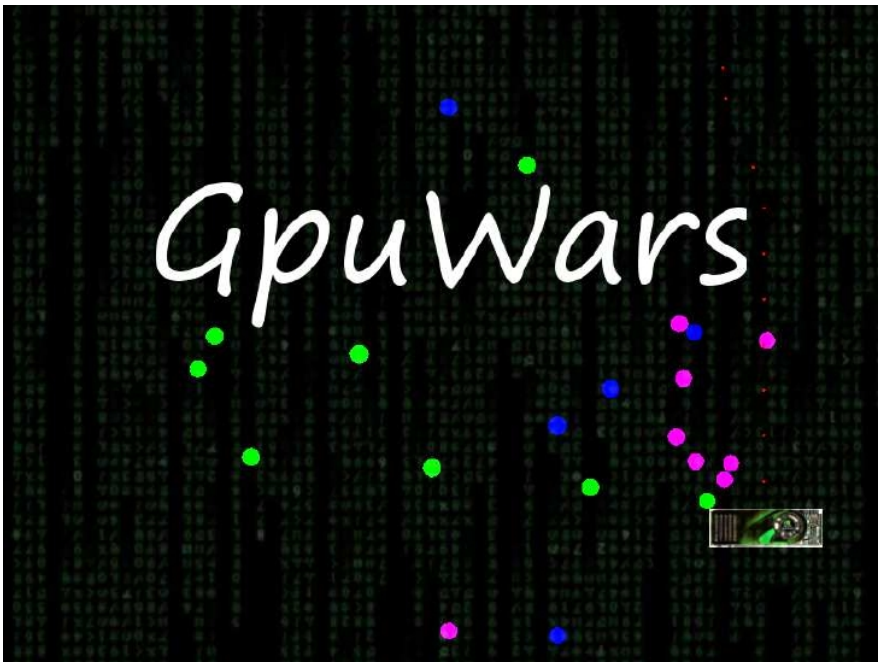


Figure 1: *Teaser of the GpuWars Game.*

Abstract

The GPUs (Graphics Processing Units) have evolved into extremely powerful and flexible processors, allowing its usage for processing different data. This advantage can be used in game development to optimize the game loop. Most GPGPU works deals only with some steps of the game loop, allowing to the CPU to process most of the game logic. This work differ from the traditional approach, by presenting and implementing practically the entire game loop inside the GPU. This is a big breakthrough on game development, since the CPUs are evolving to multi-core, and future games will need similar parallelism as the GPUs programs.

Keywords:: Digital Games, Game Architecture, GPGPU, Game Physics, Game AI

Author's Contact:

{ mjoselli, esteban } @ic.uff.br

1 Introduction

The increase of the level of realism in games depends not only on the enhancement of modeling and rendering effects, but also on the improvement of different aspects such as animation, artificial intelligence of the characters and physics simulation.

Computers, new video game consoles (such as the Microsoft Xbox 360 and the Sony Playstation 3) and GPUs feature multi-core processors. For this reason, paralleling the game tasks is getting more and more important. This work has make a game with its tasks execution in parallel, with the sequential execution kept to a minimum.

The development of programmable GPUs has enabled new possibilities for general purpose computation (GPGPU) which can be used to enhance the level of realism of virtual simulations. Some examples of works in GPGPU that address these issues are Quantum Monte Carlos [Anderson et al. 2007], finite state machines [Rudomn et al. 2005] and ray casting [Muller et al. 2007].

A lot of games and works that uses GPGPU to process some parts of its tasks in the GPU and another on the CPU. This causes limitation on the simulation, because it requires a lot of data transfers between the CPU and GPU, and this can be the bottleneck of the simulation [Krueger 2008]. This work implements all the methods of the game entirely on the GPU with the use of CUDA architecture keeping the GPU-CPU communication to a minimum.

This work is particular important in order to present a paradigm that can be used in currently GPUs and video games (Xbox 360 and Playstation 3), but also in future CPU architectures [Intel 2009], where a massively cores are available.

The paper is organized as follows: Section 2 presents the GPGPU concepts. Section 3 presents some related works on GPGPU that can be applied to games. Section 4 presents the design of the GpuWars game. Section 5 presents the architecture and section 6 present the physics aspects of the architecture. Section 7 presents the game logic aspects of the architecture and section 8 presents the results. Finally section 9 presents the conclusions and future works.

2 GPGPU

GPUs are powerful processors dedicated to graphics computation which are much faster than CPU when considered all the parallel processors available. A nVidia 8800 ultra [NVIDIA 2006], for instance, can sustain a measured 384 GFLOPS/s against 35.3 GFLOPS/s for the 2.6 Ghz dual core Intel Xeon 5150 [NVIDIA 2008b].

GPUs are very good for processing applications that require high arithmetic rates and data bandwidths. Because of the SIMD parallel architecture of the GPU (the nVidia GeForce 9800 GX2 [nVidia 2009b], for example, has 256 unified stream processors), the development of this kind of application requires a different programming paradigm than the traditional CPU sequential programming model.

Nvidia and AMD/ATI are implementing unified architectures in their GPUs. Each architecture is associated with a specific language: Nvidia has developed CUDA (Compute Unified Architecture) [nVidia 2009a] and AMD developed CAL (Compute Abstrac-

tion Layer) [AMD 2008]. One main advantage in the use of these languages is that they allow the use of the GPU in a more flexible way (both languages are based on the C language) without some of the traditional shader languages limitations such as "scatter" memory operations, i.e. indexed write array operations, and others that are not even implemented as integer data operands like bit-wise logical operations AND, OR, XOR, NOT and bit-shifts [Owens et al. 2007]. Nevertheless, the disadvantage of these architectures is that they are only available for the vendors of the software, i.e., CUDA only works on Nvidia and CAL only works on AMD/ATI cards. In order to have GPGPU programs that work on both GPUs it is necessary to implement them in shader languages like GLSL (OpenGL Shading Language), HLSL (High Level Shader Language) or CG (C for Graphics) with all the vertex and pixel shader limitations and idiosyncrasies. In the near future it will be possible to use OpenCL (Open Computing Language) [Group 2009] which is available in beta for both nVidia and AMD graphics cards at the moment of the writing of this paper.

In addition, Intel has recently presented a new architecture for GPUs called Larrabee [Seiler et al. 2008]. It is made up of several x86 processors in parallel which can be used to process both graphics and non-graphics data. The advantage of this architecture is that it does not need a special language, just plain C. Nevertheless, it will only be available in 2010.

3 Related Work

There are a lot of works that deals with the GPGPU field, but the application of these works on game fields are mostly concentrated on the game physics.

Physics on the GPGPU is a potential field and many works could achieve considerable speedup by taking the physics calculations from the CPU and processing on the GPU. All the major physics engine for games in the market has make, or is making, attempts to use of the GPU to process its calculations. The work of Green [Green 2007] presents an implementation on the GPU of some methods of the commercial physics engine called Havok FX which was being constructed to be a GPGPU version of Havok Physis [Havok 2009]. The Havok FX was discontinued when Havok was bought by Intel, but there are rumors that it will be continued with the release of Intel new architecture for GPU [Seiler et al. 2008]. Also the PhysX of nVidia [NVIDIA 2009c] is a physics engine that uses the CUDA architecture to optimizate its calculation [Harris 2009]. Also Bullet [Coumans 2009], an open source physics engine, is also investing in porting it to the GPU and has release some demos with some aspects of the engine running on the GPU. Also in [Joselli et al. 2008b] a hybrid physics engine that has some of its calculations on the GPU is present. Besides the physics engines, there are other works related to the implementation of physics simulation processes on the GPU like: particle system [Kipfer et al. 2004], deformable bodies system [Georgii et al. 2005], and collision detection [Govindaraju et al. 2003].

Physics simulation works very well on the GPU because of the high performance of the stream processors, which allows high parallelism of the physics problems that can be solved in this structure. With that, it is possible to have faster physics simulation on games, and also more physics realistic games.

Another field that could be implemented in the GPGPU and can be used by game is the game AI or game logic. This field is not very explored and there are very simple works on the field. There are implementation of finite state machine on the GPU [Rudomn et al. 2005], but this work implements very primitive behavior that cannot be used for games.

Another field that can be used for game that explores GPGPU is crowd simulation, like the works [Shopf et al. 2008; Passos et al. 2008; Silva et al. 2008; Chiara et al. 2004]. Crowd simulation can be used in games for simulating: the behavior of herbs of animals [Passos et al. 2008; Silva et al. 2008], people walking on the street [van den Berg et al. 2008], soldiers fighting in a battle [Jin et al. 2007], spectators watching a performance [nVidia 2008c] and also to populate game worlds [Shopf et al. 2008], like a GTA game [North 2008]. These works are particularly important since they propose a simple AI model implementation into a GPU architecture.

There are also some works that deals with the distribution of task between the CPU and GPU, like [Zamith et al. 2007; Zamith et al. 2008; Joselli et al. 2008a; Joselli et al. 2008b; Joselli et al. 2009]. These works concentrate on the GPU most the physics tasks of the game and these tasks can be distributed to the CPU. Even though these works presents some aspects of the game tasks inside the GPU, the present work differs from the latter, since it presents all the game tasks that needs to be processed developed inside the GPU.

There are no available work on the literature that use the GPU to process the entire game logic, like the one present in this work, just some tasks of the game.

4 The Design of the Game

The GpuWars is a massive 2D prototype shooter with a top-down 2D perspective. The game is similar to a 2D shooters like Geometric Wars [Creations 2009] and E4 [Inc. 2009]. The main enhancements of GPUWars is that it uses GPU to process its calculations, allowing to process and render thousands of enemies, while similar games only process hundreds.

The game play is very simple: the player plays as a GPU card (which is called "GPUship") inside the "computer universe", and he needs to process (by shooting them) polygons, shaders and data (the enemies) from a game. Every time the "GPUship" make physical contact with a enemy it looses time and in consequence it looses FPS. The objective is to process the maximum number of data in the smaller amount of time, and keep the game interactive with a minimum 12 frames per second.

The GpuWars uses the keyboard as the input device of the game, one set of controls are used to control the movement of the "GPUship", and another set to control the direction of the shots.

5 The Architecture

Computer games are multimedia applications that employ knowledge of many different fields, such as Computer Graphics, Artificial Intelligence, Physics, Network and others [Valente et al. 2005]. More, computer games are also interactive applications that exhibit three general classes of tasks: data acquisition, data processing, and data presentation. Data acquisition in games is related to gathering data from input devices as keyboards, mice and joysticks. Data processing tasks consist on applying game rules, responding to user commands, simulating Physics and Artificial Intelligence behaviors. Data presentation tasks relate to providing feedback to the player about the current game state, usually through images and audio. In this architecture practically all game logic is processed in the GPU, i.e all the data processing tasks, only using the CPU for tasks that need to make use of CPU like data acquisition.

This architecture was implemented using CUDA technology [nVidia 2009a] for GPGPU processing; OpenGL for rendering; GLSL (OpenGL Shading Language) for shaders; and GLUT (OpenGL Utility Toolkit) for window creation and input gathering.

The game loop of the GpuWars work as follows. First the CPU gather the input and sends it to the GPU. The GPU treat this data, making the necessary adjustments,i.e, the transformation of the player's position and the creation of the players shots. The GPU starts updating the bodies by applying the physics behavior on them and their logic behavior, which corresponds to the artificial intelligence step. These updates are put on a VBO (Vertex Buffer Object) and sended to the shaders for rendering. The GPU also sends variables to the CPU in order to tell if it should terminate the application. This game loop is illustrated in figure 2.

To resume, the CPU is responsible for:

- creating a window;

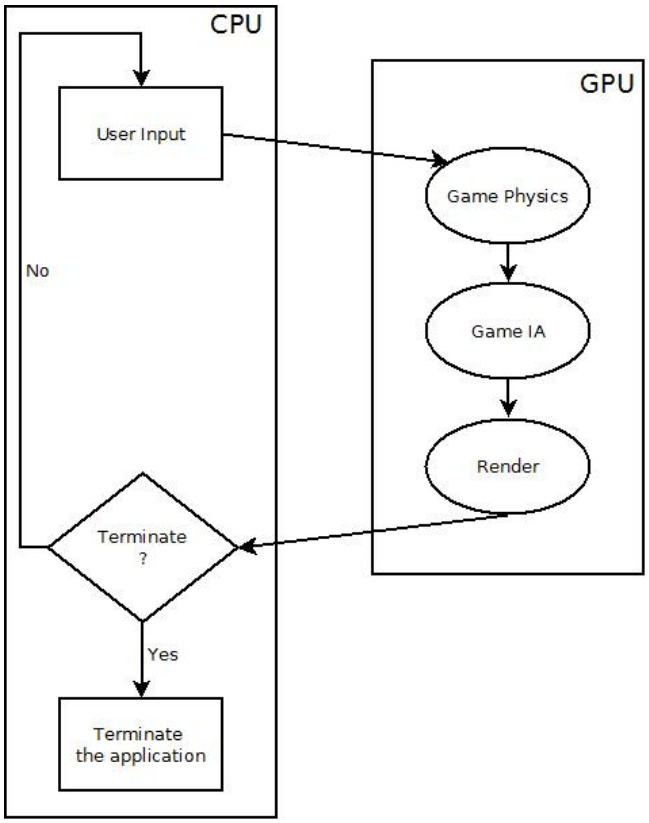


Figure 2: *Game Loop of GpuWars.*

- gather the players input and send it to the GPU;
- make the GPU calls;
- execute the music and sound effects;
- and terminate the application, i.e, destroy the windows and release the data.

While the GPU is responsible for:

- applying the physics on the bodies;
- process the artificial intelligence;
- determinate the game status, like the player scores;
- and determinate the end of the game.

The data that is exchanged between the CPU and GPU is encapsulate in special structure, in order to keep the communication between the CPU and the GPU to a minimum, since this process can be a bottleneck of any simulation that has communication between CPU and GPU [Krueger 2008].

GPGPU programs are divided in threads. In order to process the main game logic which needs to be executed sequentially, the proposed architecture have a special CUDA thread which is responsible for it, and is the same that treats the "GpuShip" data and inputs. This processing includes: update the position of the "GpuShip" accordingly to the input; creation of shots, which are created in other CUDA threads; determinate the scores; determinate the game over; and determinate the creation of new enemies. The others threads are responsible for updating the enemies and the shots, like collision detection and response and the individuals behavior. The positions and type are put in a VBO and sent to a vertex shader in order to render the individuals without using the CPU. Also to deal with the creation of the shots and enemies, the architecture keeps a list with the values to indicate available positions for individuals creation. Using this structure the GPU processes some empty threads, threads that practically does not process anything, and also different codes in different threads, which can affect the performance because of the threads synchronization inside the CUDA block. In order to avoid this, the architecture groups similar threads together in a CUDA block, avoiding the lost in performance caused by the thread synchronization. Figure 3 illustrate the process of the different threads.

GPGPU programs does not have native pseudo random number generation. In order to fulfill that need this work developed a pseudo-random number generation based on nVidia demo [Podlozhnyuk 2007].

In order to implement this architecture some data structure are needed, these are the data that are required for each individual:

- one vector with the individual position;
- one vector with the individual force;
- one vector with the individual direction/orientation;
- one integer as the individual type, which can be player, shot or enemy types;
- one integer with the individual energy;
- one float for the individual mass;

This architecture is build in a way that it can be also used, with proper modifications, in 3D games. In the next sections the most important steps there are processed on the GPU, the physics step and the AI step, are present.

6 Physics Step

This step is responsible for the physics behavior, i.e, how the bodies process and resolve all bodies collisions and response. The physics of this architecture is based on the physics on particle systems [nVidia 2008a; Microsoft 2007; Kipfer et al. 2004] and in a hybrid physics engine [Joselli et al. 2008b].

Collision detection is a complex operation. For n bodies in a system, their must be a collision detection check between the $O(n^2)$ pairs of bodies. Normally, to reduce this computation cost, this task is performed in two steps: first, the broad phase, and second, the narrow phase. In the broad phase, the collision library detects which bodies have a chance of colliding among themselves. In the narrow phase, a more refined algorithm to do the collisions tests are performed between the pairs of bodies that passed by the broad phase.

The physics step is responsible for:

- Make the broad phase of the collision detection;
- Calculate the narrow phase of the collision detection, i.e, apply the collision in each body;
- Forwarding the simulation step for each body by computing the new position and velocities according to the forces and the time step, i.e., integrating the equations of motion;

6.1 The broad phase

This phase is responsible for avoiding the n^2 comparison between all the individuals, and also avoid doing a narrow phase of the collision detection between the n^2 individuals which is normally done by spatial hashing.

There are many ways to do a spatial hashing for the broad phase of the collision detection. This work uses a uniform grid, which has a constant building cost (which makes the simulation more constant) and is very suitable for the parallel structure of the GPU. Also this structure is used in the AI step in order to determinate the vision of the bodies.

This work has based its implementation on the spatial hashing with sort of the nVidia particles demo [nVidia 2008a] and the CUDA broad phase implementation [Le Grand 2007]. This work differs from such implementation because it is adapted and optimize the structure and methods to be used with the GPGPU game loop process and to fill the requirements of the GpuWars game, which needs bigger grids and larger number of objects in the grid in order to be

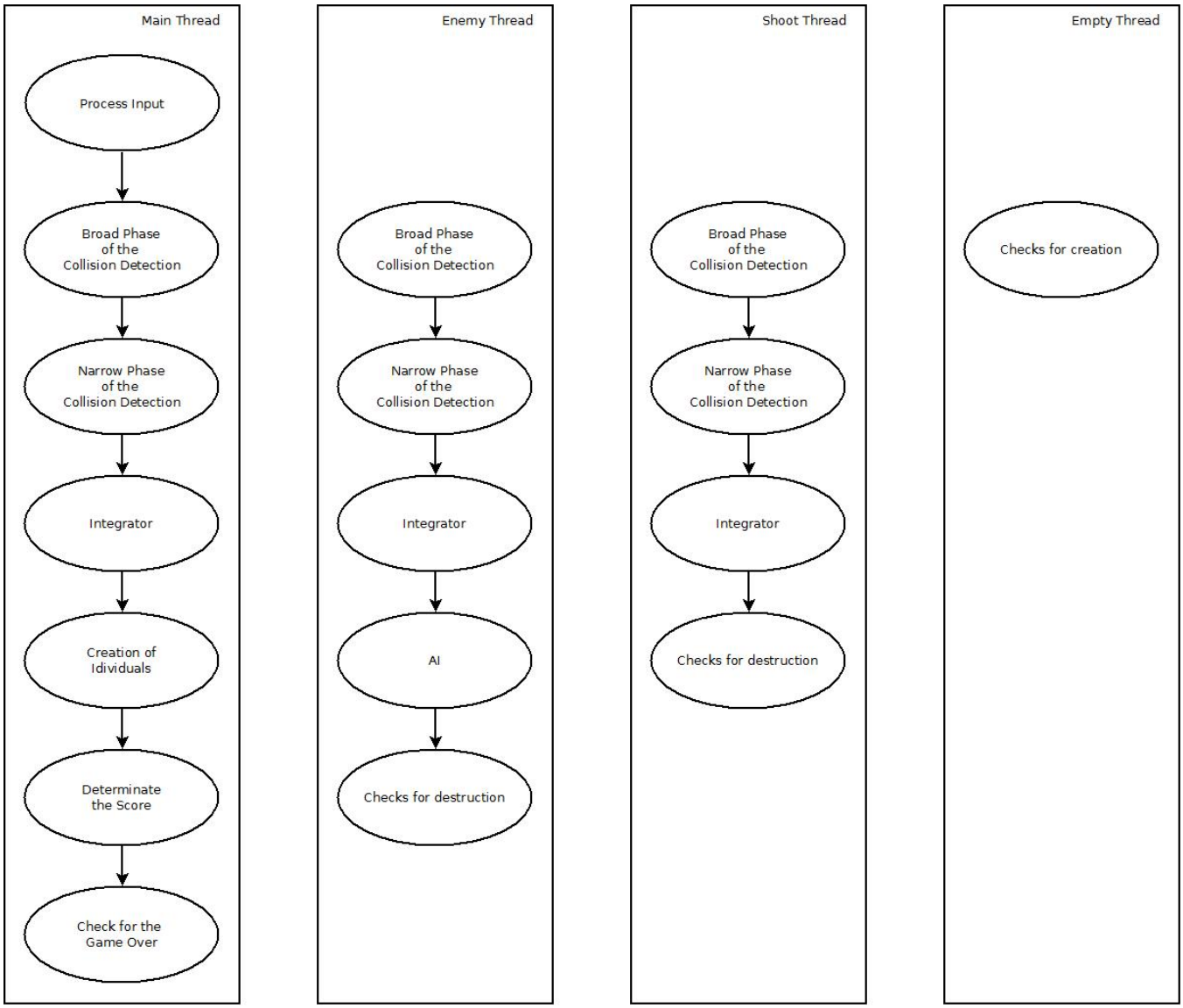


Figure 3: *The Different Process of the CUDA Threads of GpuWars.*

faster for the AI steps, which uses the grid to simulate the vision of the enemies.

6.2 The narrow phase of the collision detection

The narrow phase of the collision detection is responsible for doing the collision detection among the rigid bodies. In this work, instead of doing the collision check between all the polygons of the individuals, it is implemented a basic primitive area element, that complex models are put inside.

There are two types of bounds that this work implements, used to surround every model, simplifying the narrow phase of the collision detection: a circle bounds and a bounding rectangle. The circle bound is used whenever is possible. This is done in order to save memory, since the circle bound only needs the position vector and a radius, while the bounding rectangle needs four variables.

6.3 The Integrator

This method is responsible for integrating the equations of motion of a rigid body [Eberly 2004]. In this work it consist on a simple step, since it does not takes into account the angular velocities and torque. This method updates crowd individual velocity based on the forces that are applied to it, which are sent to the integrator, and then it updates the position based on its velocities, using an integration method based on Euler integration. Euler integration is one of the simplest form of integration. Mathematically, it evaluates the derivative of a function at a certain time, and linearly extrapolate based on that derivative to the next time step.

7 AI Step

Game AI is used to produce the illusion of intelligence in the behavior of non-player characters (NPC), and in the case of GpuWars, the enemies. There are a lot of ways to implement the game AI such as finite state machines, fuzzy logic, neural networks, and many others [Bourg and Seemann 2004]. This work uses finite state machine (FSM). Finite state Machines are powerful tools used in many computer game implementations [Dybsand 2000; Rankin and Vargas 2009; Li and Woodham 2009], like the NPC behavior, the characters animation states and the game menu states.

A finite state machine is a model of behavior composed of a states, the transitions between those states, and the actions. This work implements 3 different behaviors using FSM, the kamikaze, group and tricky behaviors, which are present in the next subsections.

The behaviors are affected by the size of vision (which uses the grid made by the broad phase of the collision detection), velocity and energy, which are variables available for each type of enemy. With the modification of these values, this work implements seven different types of enemies.

7.1 Kamikaze Behavior

The kamikaze approach is a behavior that simulates suicidal attacks. It uses a state machine that has only four state, wandering, attacking, checking energy and dead, and can be seen on figure 4.

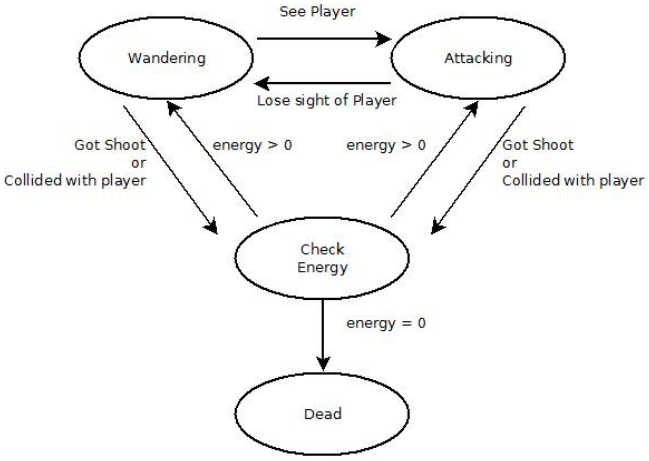


Figure 4: *The Kamikaze State Machine.*

The kamikaze is a very simple behavior. It wanders until it sees the "GPUShip", then it goes attacking it by throwing itself against it. This approach is well suited for a GPU architecture, since few information about the scene is necessary.

7.2 Group Behavior

The group behavior is a behavior that make groups, avoid bullets and attacks. It has a state machine that has six state, wandering, grouping, attacking, checking energy, avoiding bullets and dead, and can be seen on figure 5.

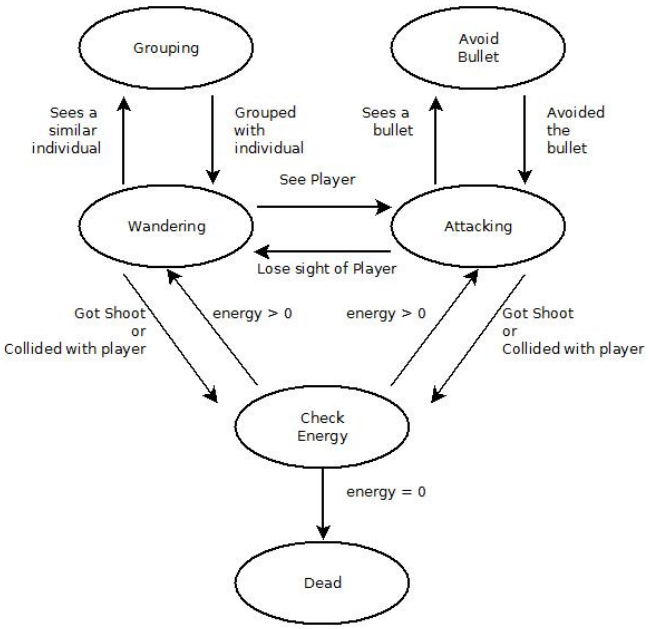


Figure 5: *The Group State Machine.*

This behavior is also very simple. The individual wanders trying to find similar individuals, i.e, individuals of the same type, and the "GPUShip". If it sees a similar individual, it goes close to it and make a group. And if it can see the player, it attacks the player by throwing itself against it. If the individual sees a bullet coming in its direction it tries to avoid it.

7.3 Tricky Behavior

The tricky behavior is the most complex behavior of the game. This behavior tries also groups similar individuals and it is the only that recoveries energy. It has a state machine that has seven states, wandering, grouping, attacking, avoiding bullets, checking energy, escaping and dead, and can be seen on Figure 6.

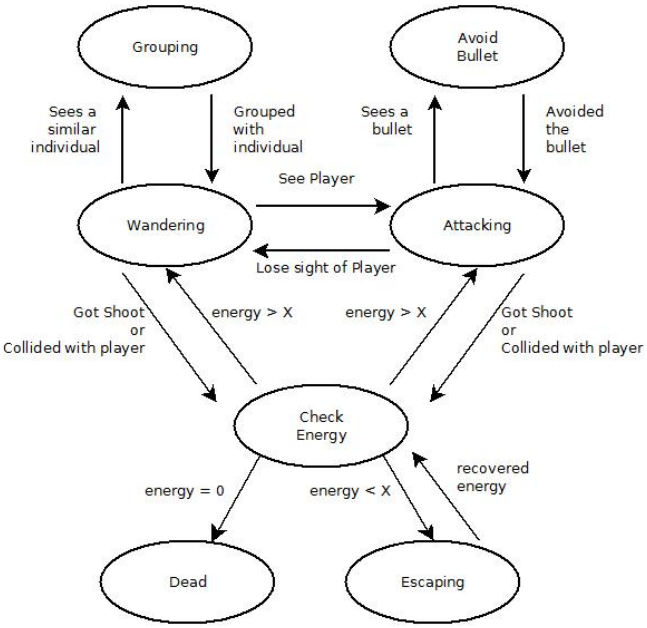


Figure 6: *The Tricky State Machine.*

The enemy wanders trying to find the "GPUShip" or similar individuals. If it sees a similar individual, it goes close to it and make a group. If it is seeing the player, it throws itself against it. If the individual sees a bullet coming in its direction it tries to avoid it. If it has little energy it tries to scape to recover the lost energy.

8 Results

This work has decided to make the tests in the minimum hardware that can run CUDA, a notebook with an AMD Turion Dual-core with 3GB RAM memory and equipped with nVidia Geforce mobile 8200M GPU card (which has only 8 stream processors), running on Windows Vista.

The number of enemies determines the performance of the game. This work has decided to have a maximum bound of 8192 enemies. A screenshot of the game can be seen of figure 7.

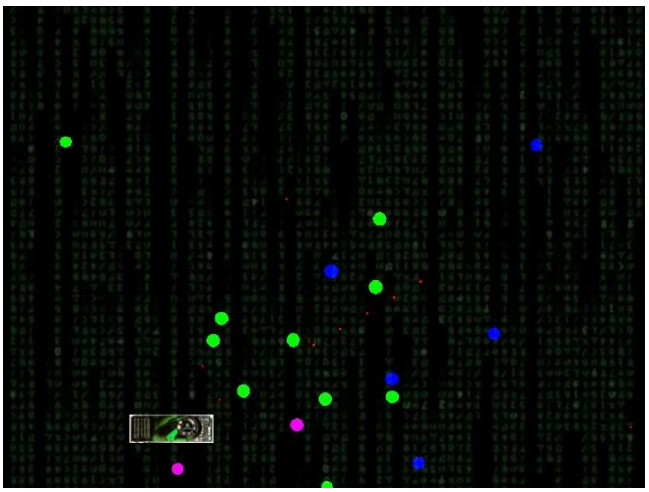


Figure 7: *A Screenshot of the game.*

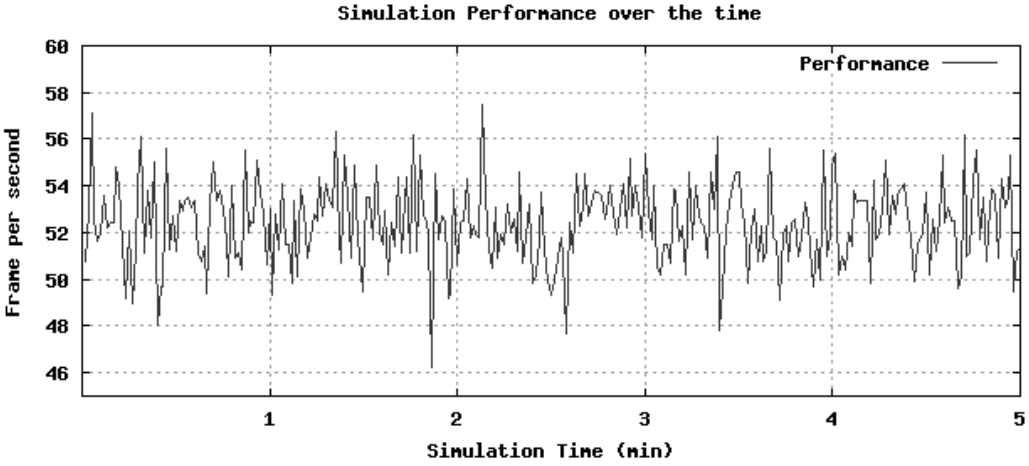


Figure 8: *Performance of the game*

To better view the performance, figure 8 show a graph with the performance in FPS of the game in 5 minutes of the application.

From this figure can be seen that the performance of the game ranges from 45 to 58 frames per second. This performance is considered optimal in a game [Joselli et al. 2009].

The game was also tested with a more powerful hardware, a quad-core with a nVidia GeForce 8800GS GPU card (which has 96 stream processors), with similar results but with a speedup of three times (the FPS ranges from 130 to 170).

9 Conclusions and Future Work

The GPUs have evolved and can be used to process different tasks of the game loops. Most works deals with some aspects of the game loop, with more focus on the game physics. This work differ from the related GPGPU works, presenting a game that has all the game logic inside the GPU. This can make a new trend on game development.

Future works will focus on creating more complex behavior of enemies, by implementing other game AI techniques, like hierarchical state machines, fuzzy logic and neural networks. Also the authors will proceed by evolving the architecture so it can be used in other type of games.

References

AMD, 2008. Amd stream computing. Avalible at: http://ati.amd.com/technology/streamcomputing/firestream-sdk-whitepaper.pdf. 20/02/2008.

ANDERSON, A., III, W. G., AND SCHRDER, P. 2007. Quantum monte carlo on graphical processing units. *Computer Physics Communications 177(3)*.

BOURG, D. M., AND SEEMANN, G. 2004. *AI for Game Developers*. O'Reilly Media, Inc.

CHIARA, R. D., ERRA, U., SCARANO, V., AND TATAFIORE, M. 2004. Massive simulation using gpu of a distributed behavioral model of a flock with obstacle avoidance. In *Vision, Modeling, and Visualization (VMV)*, 233–240.

COUMANS, E., 2009. Bullet physics library. Disponvel em: http://www.bulletphysics.com.

CREATIONS, B., 2009. Geometry wars retro evolve. Avalible at: http://www.bizarrecreations.com/games/geometry_wars_retro_evolved/.

DYBSAND, E. 2000. A finite state machine class. *Game Programming Gems*, 237–248.

EBERLY, D. H. 2004. *Game Physics*. Morgan Kaufmann.

GEORGII, J., ECHTLER, F., AND WESTERMANN, R. 2005. Interactive simulation of deformable bodies on gpu. In *Proceedings of Simulation and Visualization 2005*, 247–258.

GOVINDARAJU, K. N., REDON, S., LIN, M. C., AND MANOCHA, D. 2003. CULLIDE: interactive collision detection between complex models in large environments using graphics hardware. In *Graphics Hardware 2003*, 25–32.

GREEN, S., 2007. Gpgpu physics. Siggraph07 GPGPU Tutorial.

GROUP, K., 2009. Opencl - the open standard for parallel programming of heterogeneous systems. Avalible at: http://www.khronos.org/opencl/.

HARRIS, M., 2009. Cuda fluid simulation in nvidia physx. Siggraph Asia 2009: Beyond Programmable Shading course.

HAVOK, 2009. Havok physics. Avalible at: http://www.havok.com/content/view/17/30/.

INC., Q. E., 2009. Every extend extra extreme. Avalible at: http://www.qentertainment.com/eng/2007/09/every_extend_extra_extreme.html.

INTEL, 2009. Intel multi-core technology. Avalible at: http://www.intel.com/multi-core/.

JIN, X., WANG, C. C. L., HUANG, S., AND XU, J. 2007. Interactive control of real-time crowd navigation in virtual environment. In *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 109–112.

JOSELLI, M., ZAMITH, M., VALENTE, L., CLUA, E. W. G., MONTENEGRO, A., CONCI, A., FEIJÓ, B., DORNELLAS, M., LEAL, R., AND POZZER, C. 2008. Automatic dynamic task distribution between cpu and gpu for real-time systems. *IEEE Proceedings of the 11th International Conference on Computational Science and Engineering*, 48–55.

JOSELLI, M., CLUA, E., MONTENEGRO, A., CONCI, A., AND PAGLIOSA, P. 2008. A new physics engine with automatic process distribution between cpu-gpu. *Sandbox 08: Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, 149–156.

JOSELLI, M., ZAMITH, M., VALENTE, L., CLUA, E. W. G., MONTENEGRO, A., CONCI, A., AND FEIJÓ, PAGLIOSA, P. 2009. An adaptative game loop architecture with automatic distribution of tasks between cpu and gpu. *Proceedings of the VII Brazilian Symposium on Computer Games and Digital Entertainment*, 115–120.

KIPFER, P., SEGAL, M., AND WESTERMANN, R. 2004. Uberflow: a gpu-based particle engine. In *Graphics Hardware 2004*, 115–122.

KRUEGER, J. 2008. A gpu framework for interactive simulation and rendering of fluid effects. *IT - Information Technology 4*, (accepted).

LE GRAND, S. 2007. Broad-phase collision detection with cuda. In *GPU Gems 3*, H. Nguyen, Ed. Addison Wesley Professional, August, ch. 32.

LI, F., AND WOODHAM, R. J. 2009. Video analysis of hockey play in selected game situations. *Image Vision Comput. 27*, 1-2, 45–58.

MICROSOFT, 2007. Advanced particles. Siggraph 2007: Real-Time Rendering in 3D Graphics and Games course.

MULLER, C., STRENGERT, M., AND ERTL, T. 2007. Adaptive load balancing for raycasting of non-uniformly bricked volumes. *Parallel Computing 33(6)*, 406–419.

NORTH, R. G., 2008. Grand theft auto iv, rockstar games. Avalible at: `http://www.rockstargames.com/IV/`.

NVIDIA. 2006. Geforce 8800 gpu architecture overview. tb-02787-001_v0.9. Technical report, NVIDIA.

NVIDIA, 2008. Cuda particles. Avalible at: `http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/particles/doc/particles.pdf`.

NVIDIA. 2008. Nvidia - cuda compute unified device architecture. Programming guide, NVIDIA.

NVIDIA, 2008. Skinned instancing. Avalible at: `http://developer.download.nvidia.com/SDK/10/direct3d/Source/SkinnedInstancing/doc/SkinnedInstancingWhitePaper.pdf`.

NVIDIA, 2009. Nvidia cuda compute unified device architecture documentation version 2.2. Avalible at: `http://developer.nvidia.com/object/cuda.html`.

NVIDIA, 2009. nvidia geforce 9800 gx2 specification. Avalible at: `http://www.nvidia.com/object/product_geforce_9800_gx2_us.html`.

NVIDIA, 2009. Nvidia physx. Avalible at: `http://www.nvidia.com/object/nvidia_physx.html`.

OWENS, J. D., LEUBKE, D., GOVINDARAJU, N., HARRIS, M., KRGER, J., LEFOHN, A. E., AND PURCELL, T. J. 2007. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum 26(1)*, 80–113.

PASSOS, E., JOSELLI, M., ZAMITH, M., ROCHA, J., MONTENEGRO, A., CLUA, E., CONCI, A., AND FEIJÓ, B. 2008. Supermassive crowd simulation on gpu based on emergent behavior. In *Proceedings of the VII Brazilian Symposium on Computer Games and Digital Entertainment*, 81–86.

PODLOZHNYUK, V., 2007. Parallel mersenne twister. Avalible at: `http://developer.download.nvidia.com/compute/cuda/sdk/website/projects/MersenneTwister/doc/MersenneTwister.pdf`.

RANKIN, J. R., AND VARGAS, S. S. 2009. Fps extensions modelling esgs. In *ACHI '09: Proceedings of the 2009 Second International Conferences on Advances in Computer-Human Interactions*, IEEE Computer Society, Washington, DC, USA, 152–155.

RUDOMN, T., MILLN, E., AND HERNNDEZ, B. 2005. Fragment shaders for agent animation using finite state machines. *Simulation Modelling Practice and Theory 13(8)*, 741–751.

SEILER, L., CARMEAN, D., SPRANGLE, E., FORSYTH, T., ABRASH, M., DUBEY, P., JUNKINS, S., LAKE, A., SUGERMAN, J., CAVIN, R., ESPASA, R., GROCHOWSKI, E., JUAN, T., AND HANRAHAN, P. 2008. Larrabee: A many-core x86 architecture for visual computing. *ACM Transactions on Graphics 27*, 3.

SHOPF, J., BARCZAK, J., OAT, C., AND TATARCHUK, N. 2008. March of the froblins: simulation and rendering massive crowds of intelligent and detailed creatures on gpu. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, ACM, New York, NY, USA, 52–101.

SILVA, A. R., LAGES, W. S., AND CHAIMOWICZ, L. 2008. Improving boids algorithm in gpu using estimated self occlusion. In *Proceedings of SBGames'08 - VII Brazilian Symposium on Computer Games and Digital Entertainment*, Sociedade Brasileira de Computação, SBC, 41–46.

VALENTE, L., CONCI, A., AND FEIJÓ, B. 2005. Real time game loop models for single-player computer games. In *Proceedings of the IV Brazilian Symposium on Computer Games and Digital Entertainment*, 89–99.

VAN DEN BERG, J., PATIL, S., SEWALL, J., MANOCHA, D., AND LIN, M. 2008. Interactive navigation of multiple agents in crowded environments. In *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, 139–147.

ZAMITH, M., CLUA, E., PAGLIOSA, P., CONCI, A., MONTENEGRO, A., AND VALENTE, L. 2007. The gpu used as a math co-processor in real time applications. *Proceedings of the VI Brazilian Symposium on Computer Games and Digital Entertainment*, 37–43.

ZAMITH, M. P. M., CLUA, E. W. G., CONCI, A., MONTENEGRO, A., LEAL-TOLEDO, R. C. P., PAGLIOSA, P. A., VALENTE, L., AND FEIJÓ, B. 2008. A game loop architecture for the gpu used as a math coprocessor in real-time applications. *Comput. Entertain. 6*, 3, 1–19.

gRmobile: A Framework for Touch and Accelerometer Gesture Recognition for Mobile Games

Mark Joselli
UFF, Medialab

Esteban Clua
UFF, Medialab

Abstract

Mobile phone games are usually design to be able to play using the traditional number pads of the handsets. This is stressfully difficult for the user interaction and consequently for the game design. Because of that, one of the most desired features of a mobile games is the usage of few buttons as possible. Nowadays, with the evolution of the mobile phones, more types of user interaction are appearing, like touch and accelerometer input. With these features, game developers have new forms of exploring the user input, being necessary to adapt or create new kinds of game play. With mobile phones equipped with 3D accelerometers, developers can use the simple motion of the device to control the game or use complex accelerated gestures. And with mobile phones equipped with the touch feature, they can use a simple touch or a complex touch gesture recognitions. For the gesture to be recognized one can use different methods like simple brute force gestures, that only works well on simple gestures, or more complex pattern recognition techniques like hidden Markov fields, fuzzy logic and neural networks. This work presents a novel framework for touch/accelerometer gesture recognition that uses hidden Markov model for recognition of the gestures. This framework can also be used for the development of mobile application with the use of gestures.

Keywords:: Mobile Games, Gesture Recognition, Motion Sensors, Touch Phones,Tangible User Interfaces

Author's Contact:

{ mjoselli, esteban } @ic.uff.br

1 Introduction

Digital games are defined as real-time multimedia applications that have time constraints to run their tasks. If the game is not able to execute its processing under some time threshold, it will fail [Joselli et al. 2008]. Mobile games are also real-time multimedia application that runs on mobile phones that have time constraints and many others constraints [Chehimi et al. 2008], when compared to PC or console games, like: hardware constraints (processing power and screen size); user input, (buttons, voice, touch screen and accelerometers); and different operating systems, like Android, iPhone OS, Symbian and Windows Mobile. This makes streamilly difficult for the design and development of mobile games.

On the other hand, mobile games can have unique characteristics, making unique type of games: location based games [M1ndLab 2007; De Souza E Silva 2009], voice based games [Zyda et al. 2008], accelerometer based games [Chehimi and Coulton 2008], camera based games [Park and Jung 2009] and touch based games [Rohs 2007]. In order to develop good mobile games, they must be design to take advantages of such unique characteristics into gameplay [Zyda et al. 2007].

Mobile game phones are a growing market [Koivisto 2006] and in 2010 the sales of smartphones is expected to suppress the laptops sales [Oliver 2008]. More than 10 million of people worldwide play games on mobile phones and handheld devices [Soh and Tan 2008] and the world-wide mobile gaming revenue is expected to reach $9.6 billion by 2011 [Gartner 2007]. These are important motivations for game developers and designer to create blockbusters games.

One special characteristic of most mobile phones is that the user interaction is made mostly through number input [Chehimi and Coulton 2008; Gilbertson et al. 2008]. Because of that, the design of games must deal with this fact and design the game to use as few buttons as possible, like just one button games [Nokia 2006] and no-buttons at all games [Gilbertson et al. 2008].

The evolution of mobile phones increase the processing power of such devices and also new forms of input, like touch screen devices and devices equipped with accelerometers. With the development of touch phones, like Motorola a1200, Htc Diamond, Sony Ericson w960i, Samsung Ultra Smart F520 and Nokia N810, new forms of user interaction has appeared through the use of the finger or pen. This innovation has led to change the way users interact with the mobile operation system and with the mobile games.

With the popularization of the use of accelerometer by the Nintendo Wiimote, the major mobile phone manufactures had also equipped their hardware with accelerometer, like Nokia N95, Sony ericson F305, Samsung Omnia and Motorola W7, among others. But this new form of user interaction has not led to major change on the interaction. This is mostly because programs/games only uses the accelerometer data as orientation.

The iPhone was one of the first devices that is equipped with touch screen and accelerometer that has mostly of the user input made though touch or motion, soon others companies followed this tendency, like: RIM Blackberry Storm, Nokia 5800, LG Arena, and many others. They basically use touch for user interaction, and the use of the accelerometer data is restricted for orientation, just like the others phones equipped with accelerometer. This paper tries to fulfill a gap on user interaction by providing a framework for gesture recognition though touch input or motion input, that can be used for games or programs.

The gesture recognition is a type of pattern recognition and can be made by different ways like: brute force [Wobbrock et al. 2007], fuzzy logic [Anderson et al. 2004], Gabor wavelet transform [Mena-Chalco et al. 2008], hidden Markov model [Westeyn et al. 2003], Support Vector Machine [Prekopcsák et al. 2008a] and neural networks [King et al. 2004; Bailador et al. 2007]. This work has developed a framework that can be used for gesture recognition using hidden Makov model. In order to generate and recognize the gestures database the proposed framework is divided in two parts: one for database constriction and another for the gesture recognition.

Summarizing, this work provides the following contributions:

- A novel architecture for touch/motion gesture recognition on mobile phones;
- Presentation of performance and tests of the framework showing that it can be used in real-time;
- Recognition test showing a high accuracy rate;

The paper is organized as follows: Section 2 presents some related works on the mobile development and gesture recognition on devices equipped with touch screen and devices equipped with accelerometers. Section 3 presents and explain the gesture recognition framework. Section 4 present and discuss some results of the use of the framework. Section 5 presents the conclusions and future works.

2 Related Work

Since the gRmobile has two kinds of gestures recognition (thought touch input or accelerometer motion input), this section is divided in two subsections: one for touch screen devices were user interaction and gesture recognition for this kind of device works; and

another for accelerometer devices presenting user interaction works and gesture recognition approaches.

This work does not cover the related work on mobile game development. For this purpose the authors suggest the works [Capin et al. 2008; Chehimi et al. 2008] which covers state of the art for this topic.

2.1 Touch Devices

Nowadays, more and more devices are coming with touch screen, and most of this is because of the decreased in the respective price [Vaughan Nichols 2007]. Touch screen phone devices has the characteristics of having very few buttons and most of its users input interfaces are made through touch by finger or pen. For example the Blackberry Storm has about only 8 buttons and almost all of its user interaction is made by touch.

Most touch screen devices can have two kinds of input: dragged and pressed. The first is used when the user touches softly and can be used as a mouse being dragged. The second is when the user press hard on the device, and can be used as a mouse buttons pressed. Also modern devices has multi-touch screen devices like iPhone, Android T-G1 and Blackberry Storm, among others.

Some of these devices have used some of this types of input as gestures to enable friendly user interaction: like dragging for changing the web page, zooming options on photo view and many others features. But in third-party mobile software and games this use is very restrict.

Narayanaswamy et al [Narayanaswamy et al. 1999] shows an implementation of handwrite recognition on PDAs using Hidden Markov Model. Wei et al. [Wei et al. 2008] presents a study about using pen gestures instead of buttons in a mobile FPS game. They showed that users have little preference in using buttons over gestures and sometimes prefer the use of gestures. The gRmobile also could be used for handwrite recognition and gesture for games but a gesture database must be constructed in order to have the this functionality in the framework (in the case of handwrite recognition all the alphabet must be in the database).

Since the touch screens have similar behavior as the mouse, mouse gesture recognizer [Moyle and Cockburn 2002; Lombardi and Porta 2007; Buckland 2002] could be adapted in order to be used by touch screen devices. But this could be very hard since they are not adapted for the low processing power and memory constraints of such devices. The gRmobile is very adapted to such devices having a very good performance as will be showed in the performance evaluation section.

Even tough touch devices enables much more freedom when compared with buttons based phones, the input error by those devices are higher, as shows by Hoggan et al. [Hoggan et al. 2008] using keyboard input tests.

2.2 Accelerometer Devices

Accelerometer devices are getting more and more popular. This allows the interaction though the form of gestures recognition, being the Nintendo Wii game console the most prominent example of this new form of interaction. This approach allows users to become more engaged to video games [Crampton et al. 2007], whose experience is not only affected by button pressing and timing but also by movement. Also Sony's Playstation 3 has a controller that is equipped with accelerometers.

Nowadays, most smartphones, and also some mobile phones, come equipped with accelerometer. This allows the use of motion and gestures as user input, but very little has been done in the field. Most mobile operating systems [Oliver 2008] only uses the motion to choose the screen orientation. And also most games only uses the "tilt" of the screen and not gestures as input, like the works [Gilbertson et al. 2008; Chehimi and Coulton 2008; Valente et al. 2008] represents.

There are many relevant work related to gestures recognition for accelerometer devices. With the usage of the Wiimote can be highlighted the works [Mlch 2009; Schlömer et al. 2008]. These works use Hidden Markov Models (HMM) as the recognition algorithm and the [Mlch 2009] presents a lower recognition of 66 % and [Schlömer et al. 2008] shows a lower recognition rate of 84 %. This work also uses HMM as the recognition algorithm, but it is adapted for the lower processing power of mobile devices.

Accelerometer gesture recognition requires an intensive task to be achieved on a mobile phone. The work by Choi et al. [Choi et al. 2005] presents a Bayesian network algorithm with its computation performed in the PC to recognize numbers written on the air by accelerometer mobile phones with an average recognition rate of 97 %. Also [Pylvänäinen 2005] shows a HMM recognition algorithm also implemented on the PC with the gestures done by mobile phones with a recognition rate of more than 99 %. There are also some work in development like [Prekopcsák et al. 2008a; Prekopcsák et al. 2008b] shows two recognition algorithms implemented on the PC, a HMM and a Support Vector Machine, which can have an average recognition rate of 96 %.

Without the use of the PC, MobiToss [Scheible et al. 2008] presents the use of mobile phone to use simple gestures to interact with large public displays with all the processing made on the phone.

3 Framework Overview

The framework is build using Java language. The choice to use Java was to achieve a higher number of devices with the same framework, allowing its usage by any accelerated/touch device that can handle the acceleration data and/or touch data, and that has a Java virtual machine, like many devices from different manufacturers: Blackberries, Nokias, Sony Ericson, Motorola, Android, LG and Samsung. Also this framework can be used in a PC with a touch device like Microsoft surface or an accelerometer device like the Wiimote.

Gesture recognition with touch/accelerated devices are represented by their patterns of the input data. The recognition is made by comparing the input pattern with the database pattern, checking if they match. In order to extract the pattern from the input data stream and comparing with the database pattern, this data must be prepared and analyzed. This work uses Hidden Markov Models in order to fulfill that need. In order to build a database, the framework needs to train and saves a set of gestures, which are also used by the mobile phone for recognition.

The proposed framework has the following steps:

- Segmentation: is used to determine when the gesture begins and when it ends;
- Filtering: is used in order to eliminate some parts of the data stream that do not contribute to the gesture;
- Quantitizer: is used to approximate the stream of input data into a smaller set of values;
- Model: is used to compute likelihood of analyzed gestures.
- Classifier: is used in order to identify the input gesture accordingly to the database.

All the steps from the mobile phone to the user feedback are illustrated in figure 1. It is possible to notice that the framework has two distinct modes: one for gesture training, i.e, build the database, and one for gesture recognition, i.e, comparing the input gesture with the database.

3.1 Segmentation

Segmentation is used mainly to automatic determinate the begin and end of the gesture. This identification in touch gestures is very easy since the begin of the gesture is when the user first touch the screen and it ends when the user release the finger/pen from the screen. For the accelerometer gestures the segmentation is more difficult since the data keeps coming in a frequency that normally

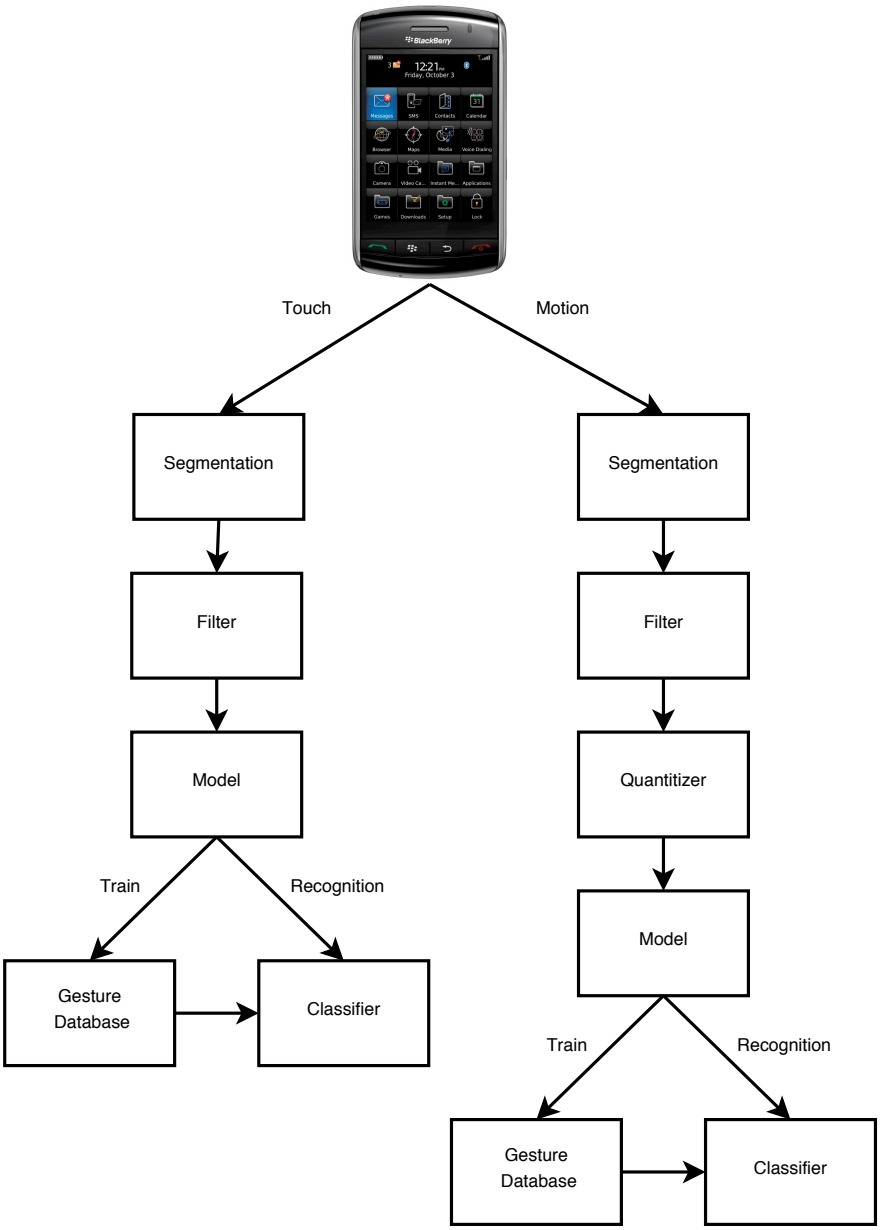


Figure 1: *System block diagram with major components of the framework.*

varies from 20 Hz to 80 Hz. This data that comes from the accelerometer is, normally, 3 numbers representing the acceleration in three axis (X, Y, Z), as figure 2 illustrates, that ranges from -3G to 3G (with G meaning the gravity).

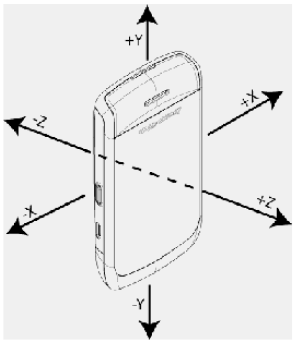


Figure 2: *The axis accelerometer*

There are some accelerometer gesture recognition systems, like [Schlömer et al. 2008], that does the segmentation by using a button based segmentation, i.e, the user needs to press a button in order to sign the beginning and the end of the gesture. This seems like an easy approach, since it avoids computations to determine the begin and the end of a gesture, like an automatic segmentation must have. But on the other hand, the interaction is worst than a no-button segmentation. In a mobile phones, sometimes, the pressing of buttons in a game can be hard since it was designed for number dialing. This comes as another reason why this work has implemented a no-button segmentation. This work does a similar approach as the works [Hofmann et al. 1998; Prekopcsák et al. 2008a].

In order to correct segmentate an accelerometer gesture, a definition of this kind of gesture is needed. This definition is made during the observation of the accelerated data and the movement of user during the recognition of different gestures. Normally, gestures begins with a fast acceleration, a continuous direction change during the gesture, and it ends with a stop of the movement. In this work, the authors have observed that normally, a good gesture needs a duration of more than 0.6 seconds and less than 2 seconds.

To correct segmentate the accelerometer gesture based on the definitions some preprocessing of the accelerated data is needed. This work uses a simple mechanism. It checks the size of a vector made by the sums of the derivative, which is the difference between the axis float and the last axis float, which is illustrated by equation 1.

$$D = \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2 + (z_k - z_{k-1})^2} \quad (1)$$

If the D value is bigger than 0.3, which was chosen during an extensively empirical study of gestures, the segmentation begins. And if the gesture is happening and this values drops to bellow 0.1 the gesture ends, i.e, it is assumed that the accelerometer device is in an idle position.

3.2 Filtering

This pass is used in order to eliminate some parts of the data stream that do not contribute to the gesture. This work uses two kinds of filters in order to eliminate noises and data that are very similar.

When a gesture is made, the data stream of the gesture may contain errors that if are sent to the HMM, some errors on the recognition may occur. In order to avoid such errors, a low pass filter is applied which is a very common filter used for noise remove.

When the gesture is made, there are a lot of data on the data stream that does not contribute to the overall characteristic of the gesture. In order to diminished the data passed to the HMM, this work uses an idle threshold filter. This uses the same equation 1, and if D value is less than 0.2, it is not included in the gesture data stream.

3.3 Quantitizer

This step is only used for accelerated gestures. Because the accelerometer continues sends its data to the processor, the amount of data may be to big to for handling into a single HMM. Also, since the amount of RAM memory of mobile phones are limited, the use of a quantitizer keeps less information in the gesture database. This work uses a k-mean algorithm which is a method of cluster analyzer. The algorithm aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. This is a similar approach as the work [Schlömer et al. 2008].

For this approach, a k value must be chosen. For this work we selected $k = 14$, since experiments from [Schlömer et al. 2008] shows that this values is optimal for the Wiimote, which can be extended to the phone accelerometers, since they have a similar architecture.

3.4 Hidden Markov Model

A hidden Markov model (HMM) is a popular statistical tool for gesture/pattern recognition. This work has based its implementation in an open source HMM implementation [Franois 2009] and in an open source HMM Wii gesture recognition [Poppinga 2009].

This work uses left-to-right HMM with 8 states for each gesture. The reason why this work choose this configuration is because it is a very efficient for accelerometer recognition, following the tests made in [Schlömer et al. 2008]. For the training process, the HMMs with the iterative Baum-Welch algorithm was used. And for recognition, the forward-backward algorithm was used. More information on this algorithms can be obtained in [Rabiner and Juang 1986; Rabiner 1990; Bilmes 2006].

3.5 Classifier

The classifier is used in order to identify gesture selecting the gesture with more likehood between the input gesture and the database gesture. This work uses a naive Bayes classifier also called simple Bayesian classifier [Friedman et al. 1997]. Naive Bayes is simple probabilistic classifier based on the so-called Bayesian theorem and it is a well known algorithm both in statistics and machine learning.

The accelerometer stream data input has some noise movements, i.e., movements that are not gestures. Because of that, a probability threshold of 0.5 is included, so that any gesture probability below that threshold is considered as a noise. The value of this threshold was determined according to empirical values.

4 Results evaluation

All tests of this work were made in a BlackBerry Storm 9530 [Kao and Sarigumba 2009] which has a 528 MHz Qualcomm processor with 128 MB of RAM, touch screen and accelerometer. In order to evaluate properly the gRmobile framework an application to train, recognize and save gestures were developed for the BlackBerry Storm. A screenshot of the Blackberry application can be seen on figure 3.

Figure 3: *An Screenshot of the application.*

Two types of tests were made in order to validate the architecture performance, one for evaluating the impact that the architecture can have on the mobile phone, and another for recognition, in order to evaluate the accuracy of the gRmobile. These tests are presented in the next two subsections.

4.1 Performance Evaluation

There are some available frameworks for accelerometer gesture recognition but most of them cannot be used in mobile phones since the processing power is very low when compared to a PC. gRmobile was designed to be used with constraint hardware of mobile phones.

The authors of this work have observed that the size of gesture database influenciates on the total time of the recognition. Another observation is that the time for touch gesture is lesser than motion gestures, since it has much less data to analyze. Table 1 shows the numerical results in average time with gesture different database sizes.

The results shows that with a database with ten or less gestures, the gRmobile can be used in real-time applications, such as games,

Table 1: *Numerical results from performance evaluation with different database sizes in average time in milliseconds.*

# of gesture in the database	time accelerometer (ms)	time touch (ms)
5	52	40
10	87	71
15	125	107
20	186	154

without major impact on the overall performance of the application. Since more than ten gestures becomes inpratical due to the fact that user must learn all different gestures, like [Pylvänäinen 2005] argue and the authors of this work agreed. These results shows that gRmobile provides a good rate for real-time gesture recognition.

Also this architecture were tested in a PC with an 3500+ Athlon64 processor with 2GB RAM memory with the Wiimote as the accelerometer device and the mouse simulating the touch device. This tests shows that resource consumption is insignificant in a PC even with a gesture database of twenty gestures.

4.2 Recognition Evaluation

In order to test the accuracy of the recognition, a dataset of gestures have been created. These work have defined ten different gestures for motion gestures, and ten gestures for touch gestures similar to the motion gestures. These gestures can be seen on figure 4.

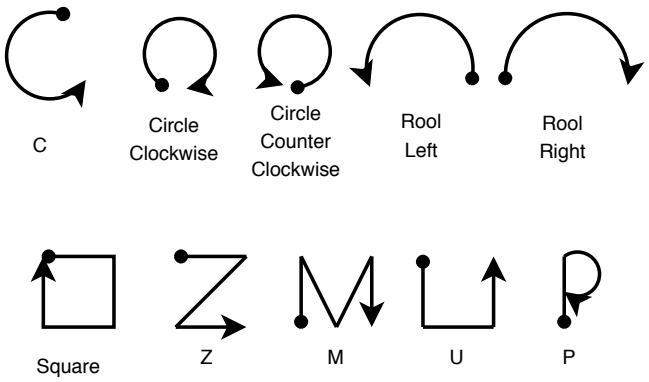


Figure 4: *The gesture database used in tests.*

All gestures were repeated ten times by a group of four different users, which provided a four hundred examples overall. The group consists of three men users (participants A, B and D) and one women user (participant C) with age ranging between 21 and 42. None of the participants was physically disabled. One participant have major experience with touch/accelerometer device (participant A), two have minor experience with such devices (participant B and C) and one have none experience with the touch and accelerometer mobile phone devices (participant D). The results can be seen on table 2.

These results show a high fidelity recognition rate of the gRmobile framework with an average recognition rate of 89% in motion gestures and 98 % for touch gestures.

Also the results shows that the specialist participant (User A) obtained 99.9 % of recognition in motion gestures and 100 % in touch gestures. The low experience participants (User B and User C) have an 88.5 % in motion gestures and 98 % in touch gestures. The no experience participant (User D) have 79 % in motion gestures and 96 % in touch gestures. These results show that touch gestures are easily to be performed and recognized. The tests also show that gestures can be used with both user with big expertise and no expertise on the subject.

Figure 5 shows the average recognition in % rate of motion gestures. This results shows that the P gesture has the lowest recognition rate, an average rate of 75%, and $RollLeft$ gesture has the highest recognition rate, an average rate of 97.5%.

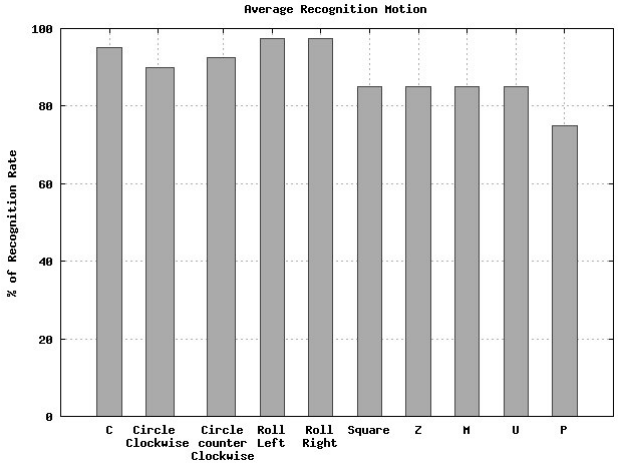


Figure 5: *Average Recognition Rate of the Motion Gestures.*

Figure 6 shows the average recognition in % rate of touch gestures. This results shows that the $Square$ gesture has the lowest recognition rate, with an average rate of 95% and $RollLeft$, $RollRight$ and U gestures have the highest recognition rate, with an average rate of 100%.

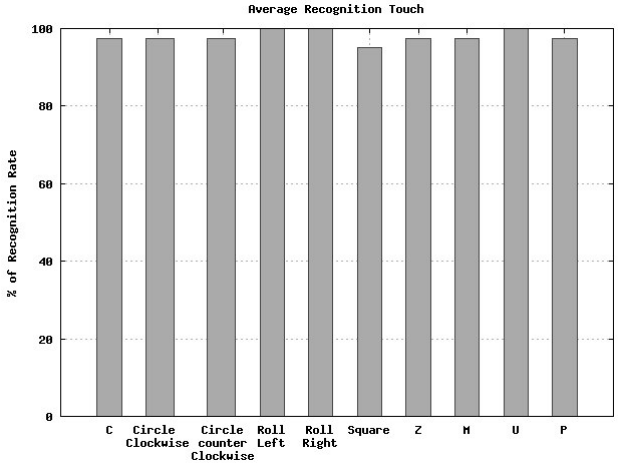


Figure 6: *Average Recognition Rate of the Touch Gestures.*

5 Conclusion

The mobile gaming market is a growing market, motivating game developers to have more focus on mobile development. Also, nowadays, mobile phones have much more processing power, allowing more complexity in mobile games.

Even with touch screens and accelerometers features becoming available in most devices, current mobile games almost do not explore these features. This work has presented a novel framework, the gRmobile, that can recognize touch and motion gestures in real-time using the mobile phone limited hardware.

The gRmobile was designed to help developer to better explore this new kind of input through gestures. This framework was built in Java in order to address most mobile phones (Blackberries, Nokias,

Table 2: *Recognition tests results in % of accuracy between all participants.*

Gesture	User A		User B		User C		User D	
	Motion	Touch	Motion	Touch	Motion	Touch	Motion	Touch
C	100	100	90	100	100	90	90	100
Circle Clockwise	100	100	100	100	80	100	80	90
Circle counter Clockwise	100	100	90	90	100	100	80	100
Roll Left	100	100	100	100	100	100	90	100
Roll Right	100	100	100	100	90	100	100	100
Square	100	100	80	100	90	100	70	80
Z	100	100	80	100	90	100	70	90
M	100	100	90	100	80	90	70	100
U	100	100	80	100	80	100	80	100
P	90	100	70	100	80	90	60	100

Sony Ericsons, Motorolas, Androids, LGs, Samsungs and others phones that have the necessary features and a java virtual machine).

There are others frameworks for gesture recognition on accelerometer and touch devices, but none of them provide the unique characteristics of gRmobile like: providing both touch and accelerometer recognition on the same platform; run in real time on the restricted mobile phone hardware (others works that implements accelerometer gesture recognition uses the CPU for processing the recognition); and it is developed in a way that can run in any system with JVM.

This work also presented many performance tests with the framework, showing that the solution can run in real-time on mobile phone devices. Also recognition test were made, showing that the gRmobile has a high recognition rate of 89% in motion gestures and 98 % for touch gestures.

In the future work topics, it is included to porting the framework to iPhone and Windows Mobile platforms. The authors also point as future work, the study of how to substitute the key press process for gestures in traditional mobile games and how this affects the gameplay.

References

ANDERSON, L., PURDY, D. J., AND VIANT, W. 2004. Variations on a fuzzy logic gesture recognition algorithm. In *ACE '04: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACM, New York, NY, USA, ACE, 280–283.

BAILADOR, G., ROGGEN, D., TRÖSTER, G., AND TRIVI NO, G. 2007. Real time gesture recognition using continuous time recurrent neural networks. In *BodyNets '07: Proceedings of the ICST 2nd international conference on Body area networks*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, BodyNets, 1–8.

BILMES, J. A. 2006. What hmms can do. *IEICE - Trans. Inf. Syst. E89-D*, 3, 869–891.

BUCKLAND, M. 2002. *AI Techniques for Game Programming (The Premier Press Game Development Series)*. Course Technology PTR.

CAPIN, T., PULLI, K., AND AKENINE-MÖLLER, T. 2008. The state of the art in mobile graphics research. *IEEE Comput. Graph. Appl. 28*, 4, 74–84.

CHEHIMI, F., AND COULTON, P. 2008. Motion controlled mobile 3d multiplayer gaming. In *ACE '08: Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, ACM, New York, NY, USA, ACE, 267–270.

CHEHIMI, F., COULTON, P., AND EDWARDS, R. 2008. Evolution of 3d mobile games development. *Personal Ubiquitous Comput. 12*, 1, 19–25.

CHOI, E. S., BANG, W. C., CHO, S. J., YANG, J., KIM, D. Y., AND KIM, S. R. 2005. Beatbox music phone: gesture-based interactive mobile phone using a tri-axis accelerometer. In *Industrial Technology, 2005. ICIT 2005. IEEE International Conference on*, ICIT, 97–102.

CRAMPTON, N., FOX, K., JOHNSTON, H., AND WHITEHEAD, A. 2007. Dance dance evolution: Accelerometer sensor networks as input to video games. In *IEEE HAVE 2007*, IEEE HAVE, 74–84.

DE SOUZA E SILVA, A. 2009. Hybrid reality and location-based gaming: Redefining mobility and game spaces in urban environments. *Simul. Gaming 40*, 3, 404–424.

FRANOIS, J.-M., 2009. Jahmm: Java implementation of hidden markov model (hmm) related algorithms. Avalible at: `http://code.google.com/p/jahmm/`.

FRIEDMAN, N., GEIGER, D., AND GOLDSZMIDT, M. 1997. Bayesian network classifiers. *Mach. Learn. 29*, 2-3, 131–163.

GARTNER, 2007. Gartner says worldwide mobile gaming revenue to grow 50 percent in 2007. Avalible at: `http://www.gartner.com/it/page.jsp?id=507467`.

GILBERTSON, P., COULTON, P., CHEHIMI, F., AND VAJK, T. 2008. Using "tilt" as an interface to control "no-button" 3-d mobile games. *Comput. Entertain. 6*, 3, 1–13.

HOFMANN, F. G., HEYER, P., AND HOMMEL, G. 1998. Velocity profile based recognition of dynamic gestures with discrete hidden markov models. In *Proceedings of the International Gesture Workshop on Gesture and Sign Language in Human-Computer Interaction*, Springer-Verlag, London, UK, International Gesture Workshop, 81–95.

HOGGAN, E., BREWSTER, S. A., AND JOHNSTON, J. 2008. Investigating the effectiveness of tactile feedback for mobile touchscreens. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, ACM, New York, NY, USA, CHI, 1573–1582.

JOSELLI, M., ZAMITH, M., VALENTE, L., CLUA, E. W. G., MONTENEGRO, A., CONCI, A., AND FEIJÓ, PAGLIOSA, P. 2008. An adaptative game loop architecture with automatic distribution of tasks between cpu and gpu. *Proceedings of the VII Brazilian Symposium on Computer Games and Digital Entertainment*, 115–120.

KAO, R., AND SARIGUMBA, D. 2009. *BlackBerry Storm For Dummies*. For Dummies.

KING, D., LYONS, W. B., FLANAGAN, C., AND LEWIS, E. 2004. Signal processing technique utilising fourier transform methods and artificial neural network pattern recognition for interpreting complex data from a multipoint optical fibre sensor system. In *WISICT '04: Proceedings of the winter international synposium on Information and communication technologies*, Trinity College Dublin, WISICT, 1–6.

KOIVISTO, E. M. I. 2006. Mobile games 2010. In *CyberGames '06: Proceedings of the 2006 international conference on Game research and development*, Murdoch University, Murdoch University, Australia, Australia, CyberGames, 1–2.

LOMBARDI, L., AND PORTA, M. 2007. Adding gestures to ordinary mouse use: a new input modality for improved human-computer interaction. In *ICIAP '07: Proceedings of the 14th International Conference on Image Analysis and Processing*, IEEE Computer Society, Washington, DC, USA, ICIAP, 461–466.

M1NDLAB, 2007. Alien revolt: Location-based massive-multiplayer online rpg. Avalible at: http://www.alienrevolt.com.

MENA-CHALCO, J., CARRER, H., ZANA, Y., AND CESAR JR., R. M. 2008. Identification of protein coding regions using the modified gabor-wavelet transform. *IEEE/ACM Trans. Comput. Biol. Bioinformatics 5*, 2, 198–207.

MLCH, J. 2009. Wiimote gesture recognition. In *Proceedings of the 15th Conference and Competition STUDENT EEICT 2009 Volume 4*, Faculty of Electrical Engineering and Communication BUT, Faculty of Electrical Engineering and Communication BUT, 344–349.

MOYLE, M., AND COCKBURN, A. 2002. Analysing mouse and pen flick gestures. In *In Proc. of the SIGCHI-NZ Symposium On Computer-Human Interaction*, SIGCHI, 266–267.

NARAYANASWAMY, S., HU, J., AND KASHI, R. 1999. User interface for a pcs smart phone. In *ICMCS '99: Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, IEEE Computer Society, Washington, DC, USA, ICMCS, 9777.

NOKIA, F. 2006. Turn limitation into strength: Design one-button games. Tech. rep., Nokia.

OLIVER, E. 2008. A survey of platforms for mobile networks research. *SIGMOBILE Mob. Comput. Commun. Rev. 12*, 4, 56–63.

PARK, A., AND JUNG, K. 2009. Flying cake: Augmented game on mobile devices. *Comput. Entertain. 7*, 1, 1–19.

POPPINGA, B., 2009. wiigee:a java-based gesture recognition library for the wii remote. Avalible at: http://www.wiigee.org.

PREKOPCSÁK, Z., HALÁCSY, P., AND GÁSPÁR-PAPANEK, C. 2008. Accelerometer based real-time gesture recognition. In *Proceedings of the 12th International Student Conference on Electrical Engineering*, International Student Conference on Electrical Engineering.

PREKOPCSÁK, Z., HALÁCSY, P., AND GÁSPÁR-PAPANEK, C. 2008. Design and development of an everyday hand gesture interface. In *MobileHCI '08: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, ACM, New York, NY, USA, MobileHCI, 479–480.

PYLVÄNÄINEN, T. 2005. Accelerometer based gesture recognition using continuous hmms. *Pattern Recognition and Image Analysis*, 639–646.

RABINER, L., AND JUANG, B. 1986. An introduction to hidden markov models. *ASSP Magazine, IEEE [see also IEEE Signal Processing Magazine] 3*, 1, 4–16.

RABINER, L. R. 1990. A tutorial on hidden markov models and selected applications in speech recognition. *Readings in speech recognition*, 267–296.

ROHS, M. 2007. Marker-Based Embodied Interaction for Handheld Augmented Reality Games. *Journal of Virtual Reality and Broadcasting 4*, 5 (Mar.). urn:nbn:de:0009-6-7939, ISSN 1860-2037.

SCHEIBLE, J., OJALA, T., AND COULTON, P. 2008. Mobitoss: a novel gesture based interface for creating and sharing mobile multimedia art on large public displays. In *MM '08: Proceeding of the 16th ACM international conference on Multimedia*, ACM, New York, NY, USA, MM, 957–960.

SCHLÖMER, T., POPPINGA, B., HENZE, N., AND BOLL, S. 2008. Gesture recognition with a wii controller. In *TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction*, ACM, New York, NY, USA, TEI, 11–14.

SOH, J. O. B., AND TAN, B. C. Y. 2008. Mobile gaming. *Commun. ACM 51*, 3, 35–39.

VALENTE, L., DE SOUZA, C. S., AND FEIJÓ, B. 2008. An exploratory study on non-visual mobile phone interfaces for games. In *IHC '08: Proceedings of the VIII Brazilian Symposium on Human Factors in Computing Systems*, Sociedade Brasileira de Computaç ao, Porto Alegre, Brazil, Brazil, SBC, 31–39.

VAUGHAN NICHOLS, S. J. 2007. New interfaces at the touch of a fingertip. *Computer 40*, 8, 12–15.

WEI, C., MARSDEN, G., AND GAIN, J. 2008. Novel interface for first person shooting games on pdas. In *OZCHI '08: Proceedings of the 20th Australasian Conference on Computer-Human Interaction*, ACM, New York, NY, USA, OZCHI, 113–121.

WESTEYN, T., BRASHEAR, H., ATRASH, A., AND STARNER, T. 2003. Georgia tech gesture toolkit: supporting experiments in gesture recognition. In *ICMI '03: Proceedings of the 5th international conference on Multimodal interfaces*, ACM, New York, NY, USA, ICMI, 85–92.

WOBBROCK, J. O., WILSON, A. D., AND LI, Y. 2007. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, UIST, 159–168.

ZYDA, M., THUKRAL, D., JAKATDAR, S., ENGELSMA, J., FERRANS, J., HANS, M., SHI, L., KITSON, F., AND VASUDEVAN, V. 2007. Educating the next generation of mobile game developers. *IEEE Computer Graphics and Applications 27*, 2, 96, 92–95.

ZYDA, M. J., THUKRAL, D., FERRANS, J. C., ENGELSMA, J., AND HANS, M. 2008. Enabling a voice modality in mobile games through voicexml. In *Sandbox '08: Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, ACM, New York, NY, USA, Sandbox, 143–147.

Hierarchical PNF Networks: A Temporal Model of Events for the Representation and Dramatization of Storytelling

Erick B. Passos Cesar T. Pozzer* Anselmo A. Montenegro
Flávio S. C. da Silva** Esteban G. Clua

UFF - MediaLab *UFSM - LaCA **USP - IME

Abstract

Storytelling is an important feature in games and also other types of (semi) automated entertainment systems such as machinima and digital-TV. The majority of the current research in storytelling use precedence-based directed acyclic graphs, or even linear sequences, to model the ordering of events in a story. This approach makes it easier to plan, recognize and perform these events in real-time, but it is also too simple to represent complex human actions, which form the basis of the most interesting stories in this niche. PNF-Networks and Interval Scripting are frameworks to represent, recognize and perform human action that was proposed in the context of computer-aided theatre. In this paper we describe two extensions to this framework that were designed and developed to enable its use in larger scale storytelling systems: Hierarchical PNF-Networks and a template-based definition. Hierarchical PNF-Networks present lower complexity propagation heuristic while the definition language enables high-level and abstract description of the temporal structure of the actions and events that compose an interactive story or game.

Keywords: storytelling, PNF networks, interval algebra

Authors' contact:
{epassos,anselmo,esteban}@ic.uff.br
*pozzer@inf.ufsm.br
**fcs@ime.usp.br

1. Introduction

A strong trend in game design is to include storytelling elements and narrative structures to enhance the player experience. However, being interactive applications, games put several challenges into this integration. These challenges are also present in other applications of interactive storytelling such as experiments on machinima [Riedl et al. 2008, Jahala et al. 2008] and interactive TV [Pozzer 2005], and can be divided into three main categories: story modeling, generation/planning, and dramatization. Story modeling tackles the problem of representing the events that make up a story, creating a solid framework for planning and dramatization tools. Planning algorithms have been used to create coherent story models and to keep it in this fashion, even in the presence of user generated events. Dramatization systems are designed to present the storyline and to interact with the user. Besides, the control of virtual actors or other agents such as camera systems still present opportunities to enrich the user experience.

Many storytelling models are constructed as directed acyclic graphs (plans are an example of such graphs), which can represent only incomplete temporal relations such as precedence or causality [Charles et al. 2003, Ciarlini et al. 2005, Barros and Musse 2005]. Since these temporal models are simple, the integration with planning and real-time dramatization systems becomes more straightforward. However, these models are a poor representation of complex human actions, which are the most common type of events in stories. This is due to the fact that these models are especially weak regarding temporal relations between events, being unable to directly represent certain types of parallelism such as mutual exclusion [Allen and Ferguson 1994]. We believe that a richer representation can lead to stories with higher complexity and consequently to a more interesting user experience.

PNF Networks were introduced by Pinhanez [Pinhanez et al. 1998] as a human action recognition framework, which uses the qualitative temporal predicates of James Allen's interval algebra [Allen 1983] to label the relations between the events that compose an action. Unlike precedence-based directed acyclic graphs, the constraints of a PNF network can represent all possible temporal relations that are present in the real world. The problem with interval algebra networks is that the propagation of the temporal constraints is NP-Hard. However, Pinhanez proposed a heuristic for this problem, called PNF-Propagation, which is linear on the number of constraints for networks where the knowledge about the temporal status of each event is discrete, restricted to three possible values and their combinations: P (past), N (now) and F (future). For instance, an event that is known to have already happened is labeled P, while an event that is known to be happening is labeled N. An event labeled PN is currently happening or already happened, and an event whose status is unknown must be labeled PNF.

This framework, augmented with real-time sensor output and past information, was used to represent and recognize complex human actions in the context of computer theatre. Pinhanez also proposed a script language [Pinhanez et al. 1997] that combines PNF-Propagation with user-defined sub-actions to control

interactive experiments in real-time. The use of these techniques to storytelling purposes was mentioned but not implemented by the original authors.

In this paper, we propose two extensions to Pinhanez's PNF-Networks that were developed to make them suitable as a model and dramatization tool for interactive storytelling. The main contributions of our work are: a template-based definition language that is used to compose high level stories by using abstract, reusable descriptions of actions; and a hierarchical version of PNF-Networks, that is augmented with an adaptation of the original restriction heuristic. The adapted heuristic aims to avoid unnecessary traversal of unaffected nodes in the network, leading to a lower complexity that enables the use of larger networks/stories. This Hierarchical PNF-Network was designed, implemented and integrated with a game engine to permit further research on interactive storytelling.

The rest of the paper is organized as follows: Section 2 compares the proposed system with related work in the field. Section 3 explains PNF-Networks and the Interval Script paradigm in more detail, while section 4 explains the new features we are proposing to the original approach. Section 5 brings an analysis of how these features are related to common storytelling concepts and constructs, which is another contribution of the present work. Finally, section 6 concludes the paper and shows future directions of our research.

2. Related Work

This section is organized in two parts: the first covers some important previous research in storytelling modeling and planning techniques. But since this work is strongly based on the PNF-Networks temporal model and algorithms, we also include here the origins and more recent work in this subject.

Chris Crawford created an interactive storytelling system, *Erasmatron* [Crawford 2004], in which the user generates stories by using *verbs* that define the actions of the characters. Each verb has an associated set of *roles*, which can be linked to the characters and objects in the particular setting. These *verbs* represent the same concept as the actions in our framework, while the *roles* are similar to the variables in our template definition language. Although Crawford´s approach is more align with character-based emergent storytelling and PNF-Networks fit better plot-based storytelling, his ideas can still benefit from PNF-Networks to detect what actions a particular character has already performed.

Plans have become one the most common approach for storytelling research partly due to their similarities to story models that emerged from narratology studies [Barros and Musse 2007]. Several previous researches included planning as part of their storytelling systems [Riedl and Young 2003, Charles et al. 2003, Ciarlini et al. 2005, Barros and Musse 2005]. We consider our work complementary to these, since we are not proposing new planning algorithms, but the integration of the Hierarchical PNF-Networks framework with current planning approaches as a formal model for the real-time execution and detection of the events that compose an interactive story.

PNF-Networks are a model and recognition framework for complex human action. It is been implemented and tested in several experiments and also with computer-aided theatre plays. We strongly recommend reading some of Claudio Pinhanez's published work [Pinhanez 1999, Pinhanez et al. 1998, Pinhanez et al. 1997] to better understand the extended framework we propose in this paper.

Meyer [Meyer 2002] further experimented with computer-aided theatre, introducing the use of XML as the definition language for the PNF-Networks. In this paper, we propose further extensions to Pinhanez's work, and also analyze some challenges and opportunities that derive from its application in storytelling research.

3. PNF Networks and Interval Scripts

PNF-Networks are a symbolic framework for modeling high-level events such as human actions - and an algorithm for detecting the occurrence of such events in real-time - that is rich in its representation power and features linear computational complexity, making it suitable to runtime interactive systems. Such as other models for representing high-level events in a story, PNF-Networks decompose actions into simpler sub-actions, but the detection of their occurrence is based on temporal reasoning about the state of the other actions in the network with a heuristic called PNF-Propagation.

Interval scripts are a proposal for the integration of PNF-Networks with user-scripted actions, which are activated by a real-time engine that uses the PNF-Propagation heuristic to determine what actions should be started or stopped. In this section we explain some terminology and the main concepts behind PNF-Networks: action representation by the means of networks of temporal constraints; the PNF-Propagation heuristic; and the interval script approach for complex action activation.

3.1 Terminology

Informally, an action is something performed by some being (character, animal, machine) that can have some consequences. An event can be understood as the perception of an action, its consequences or a natural phenomenon. In this paper, however, this subtle distinction is not so important to the mathematical formalism. More important is the perception that both

events and actions happen in intervals of time, which states that if enough information is available, one can distinguish if an event (or action) is currently happening (present), already happened (past) or has not happened yet (future). For now on, we may refer to an event or action with the term *interval*.

The other important terminology is related to the temporal relations between these intervals. Each event or actions will be represented by a node in a graph corresponding to its interval. PNF-Networks are graphs where the nodes are connected by binary directional relations. Each relation represents a constraint that affect the temporal state of the *restricted* (the destination node of the directed edge) interval based on the state of the *restrictor* (the origin of the edge).

3.1 Action composition

Most human actions can be described by a composition of simpler sub-actions. This concept is key to storytelling systems and, for instance, lets describe the events that can form a common drama in love stories: the knight characters declares this love for the princess, who is to marry a prince from another kingdom. The component sub-events can be: a) establishing shot of the castle; b) knight declares love; c) prince comes and kisses the princess; and d) princess wonders about what just happened. In order to understand this drama, the viewer must see the characters perform each of these sub-events in some organized fashion. Even this simple example will be enough to show how different models drift in representation power.

Since the occurrence of each one of these sub-actions must happen in definite intervals of time, one must define temporal relations between them to properly model and control either its runtime execution or automatic detection. A simple approach to model the temporal relations of these intervals is to consider that the sub-actions always occur in sequence, which would lead to the directed acyclic graph shown in Figure 1, where each edge represent a precedence relation. When used to control automated characters and objects such as a virtual camera, this model shows the order in which the events must be performed. If used to detection purposes, the system must perceive that the events happened in the declared order, to properly conclude that the described drama has actually been presented to the viewer, in case a Finite State Machine will easily suffice.

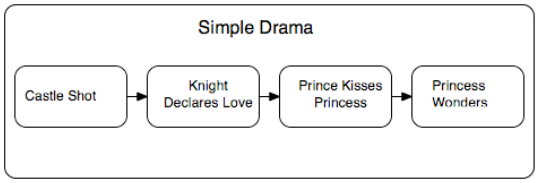


Figure 1: Sequential model for a common drama

The sequence in Figure 1 is enough when one just wants to use it as a high level representation for scripting purposes, but in many cases, some of the component sub-events can happen in parallel or in unknown order. For instance, this concept can be naively represented by the graph in Figure 2, where the two events of the knight declaring its love and the kiss between the prince and the princess are not tied to a specific order. However, this simple parallel model fails to capture an important temporal restriction of the real world: the impossibility of these two events to happen at the same time, which would otherwise lead to a inconsistent performance. The overall conclusion is that models for temporal relations between events that are based solely in precedence are not able to represent mutual exclusion situations properly [Allen and Ferguson 1994].

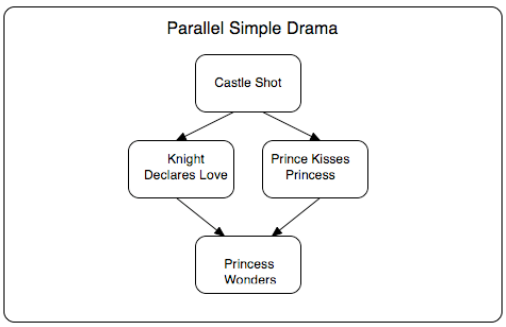


Figure 2: Naive parallel model of the drama event

Before coming back to the drama example, lets first introduce some concepts that are important to the understanding of PNF-Networks.

3.2 Temporal constraints: IA-Networks

To better represent high-level actions, one needs a richer model for temporal relations between the component sub-actions, which proper capture their complex nature. Instead of relying only in precedence and composition, the temporal relations defined by James Allen's Interval Algebra [Allen 1983] are a set of thirteen binary predicates, each representing a unique temporal relation between two intervals. Being an *algebra*, all possible knowledge about the temporal relation between any two events is proven to be a subset (empty included) of the predicates described by the *Interval Algebra*. These predicates are illustrated in Figure 3. One can notice that, except for the equal relation (the first on Figure 3), all other are pairs with an inverse version represented by an "*i*" at the beginning. For instance, the inverse of A <*Before*> B would be A <*iBefore*> B, which, concerning only the temporal relation, is the same as B <*Before*> A. However, the inverse relations are made necessary because the edges in an interval algebra graph are directed.

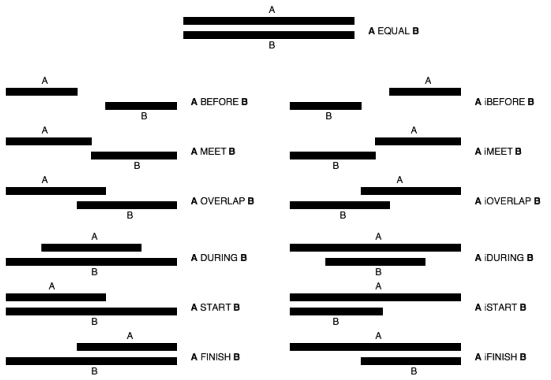


Figure 3: Allen's primitive temporal relations between two intervals

Two illustrate how these predicates better represent temporal relations between sub-actions, we can use them to define the most common relation in storytelling models: precedence. Saying that interval A precedes interval B implies that the first must end its execution before the second starts its own. But the actual start of interval B can either immediately follow the end of interval A or take some time to begin. This means that two predicates in Allen's algebra can be used to represent precedence: *Before* or *Meet*, which leads to three possible interpretations for the precedence relation:

- A <*Before*> B, where the start of interval B must take some time after the end of A to actually happen;
- A <*Meet*> B, where the start of interval B immediately follows the end of A;
- A <*Before*,*Meet*> B, where interval B either immediately follows or happens after some time A has finished, which is the usual meaning of precedence in more simple models.

These three possible representations for precedence are impossible to distinguish unless a more powerful set of predicates is used. In Interval Algebra, high level events (or long chains, sequences or other arbitrary sets of events) are represented by directed (possibly cyclic) graphs called IA-Networks, where the nodes are the sub-events and each directed edge is a subset of the thirteen predicates that compose Allen's Interval Algebra. These networks are useful when one wants a richer and finer grained representation of temporal relations. Figure 4 uses a possible IA-Network to the drama event of previous examples. It is important to notice that in IA-Networks the relation between the high-level event and the same types of predicates also represent its component sub-events. The "castle shot" camera sub-event marks the start of the "drama" event, so it is restricted by a <*iStart*> (inverse start) relation.

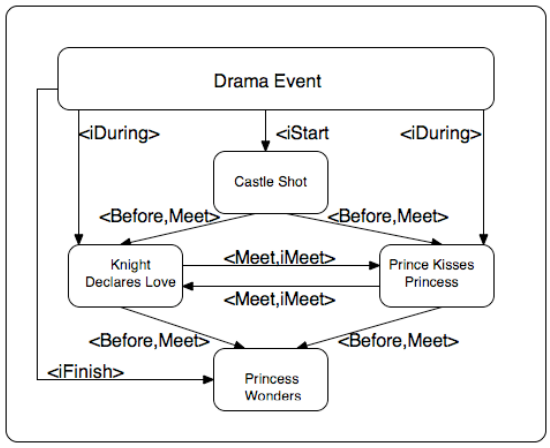


Figure 4 - Drama model represented by an IA-Network

3.3 PNF Propagation

The most important feature of an IA-Network is that it makes possible temporal reasoning, which means that one can use the knowledge about the temporal status of some nodes to infer about other nodes in the network. This is possible because each (directed) relation possibly restricts the status of the second node based on the temporal status of the first. For instance, if node A and B are related by the predicate *Meet*, and node/interval A is known to be happening now, one can conclude that node/interval B is yet to happen.

The problem with temporal reasoning in IA-Networks is that propagation algorithms for them are normally NP-Hard, which are not suitable for real-time systems, especially when used to represent complex stories (large IA-Networks). In order to solve this issue, Pinhanez [Pinhanez et al. 1998] proposed the simplification of IA-Networks by restricting the temporal knowledge about an interval to three discrete values and their combinations: P (past), N (now) and F (future). For instance, an event that is known to have already happened is labeled P, while an event that is known to be happening is labeled N. An event labeled PN is currently happening or already happened, and an event whose status is unknown must be labeled PNF.

At the same time, the temporal relations defined by Allen's interval algebra can be mapped to correspondent constraints also using discrete PNF values. A PNF constraint is a three-valued tuple, where each value is a combination of the values P, N and F. The first value in the tuple marks the possible status of the *restricted* interval given that the *restrictor* has in its current state the value P, the second value marks the possible status values for the presence of N in the origin, and finally, the third value restricts the destination in presence of F in the origin. The complete restriction is given by the sum set of the restrictions of the tuple. A conversion table between interval algebra relations and PNF constraints is given in Figure 5.

	P	N	F
E	< P , N , F >		
B	< PNF , F , F >		
iB	< P , P , PNF >		
M	< PN , F , F >		
iM	< P , P , NF >		
O	< PN , NF , F >		
iO	< P , PN , NF >		
S	< PN , N , F >		
iS	< P , PN , F >		
D	< PN , N , NF >		
iD	< P , PNF , F >		
F	< P , N , NF >		
iF	< P , NF , F >		

Figure 5 - Mapping of Allen's primitives into PNF constraints, which defines a PNF-Restriction function

For instance, the relation *Meet* would be represented by the tuple <PN,F,F>. The first value PN means that if the origin interval is possibly in the past (P) state, the destination can either be happening now or already happen. The second value, F, implies that if the origin has the state N, the destination has to be in the future. Similar constraint happens with the third value. The sum set of the values needs to be used because sometimes one cannot know the exact status of a given interval, which will lead to more than one possible restriction path. This mapping defines a function that returns the value of the *restricted* interval (X_i) based on the current value of the *restrictor* (X_j) and the binary constraint between them (C_{ij}):

X_i = PNF-Restrict(X_j, C_{ij});

Given this 3-valued discrete representation of the status and restrictions for intervals (or events that happen during this intervals), Pinhanez developed an arc-consistency heuristic [Pinhanez et al. 1998] to propagate changes in a PNF-Network (IA-Network with PNF mapped relations). This arc-consistency heuristic is linear on the number of constraints in the network and computes the restricted temporal status for each node (interval).

Formally, a set of PNF values for all the nodes in a PNF-Network is a component domain. The goal of a restriction algorithm is to find the minimal domain that satisfies all the constraints in the network, given the fixed PNF values of the sensor nodes. The arc-consistency heuristic is conservative and always finds a domain that satisfies the constraints, but its not guaranteed that this domain is minimal. Code 1 shows the pseudo-code for this heuristic.

```
Input:   A PNF-Network N with values x1, x2,
         ... , xn, and the binary constraints
         Pij;
         W, a component domain of PNF values
         for N.
Output:  AC-R(W), a component domain that
         satisfies the constraints

Algorithm:
queue = all variables xi where  Wi ≠ PNF;
W_AUX = W;
while queue is not empty
   xq = pop(queue);
   for each xi in W
      X = PNF-Restrict(W_AUXq,Pqi);
      if (W_AUXi ≠ W_AUXi ∩ X)
         W_AUXi = W_AUXi ∩ X;
         push(xi, queue);
return W_AUX;
```

Code 1 - Arc-consisntency heuristic [Pinhanez 1998]

PNF-Networks coupled with arc-consistency are a framework for representing and detecting complex actions, given that the temporal (PNF) state of some nodes in the network are determined by the output of real-time sensors. Each time a sensor output changes, the PNF-Propagation routine is executed, resulting in a new state for the nodes in the network. However, this framework alone is not useful in a storytelling scenario, where one needs mechanisms to automatically control virtual actors and other AI controlled objects.

3.4 Interval Scripts

Interval scripts are an extension to PNF-Networks that enable its use as an engine for controlling the execution of automated characters or other objects by using the results of real-time detection of user (and other automated) actions with the described PNF-Propagation approach. An interval script is a PNF-Network with three types of nodes:

- Sensor - node whose temporal status is given by direct detection such as determining if an object is touching other. Its status cannot be changed by PNF-Propagation;
- Passive node - represents an event that cannot be directly detected. Its temporal status is solely given by PNF-Propagation;
- Actions - contains callback functions that are executed every time its temporal status should change. Its PNF status is defined by user code instead of PNF-Propagation.

The difference from basic PNF-Networks is the presence of action nodes, which are placeholders for user-generated scripts. It is important to notice that the interval script engine will never change the temporal status of any user-generated script (action node). Instead, whenever a change in the network implies a temporal change in such action node, the engine executes the callback function that is correspondent to the expected change on the temporal status.

- Start - callback that should contain the code to execute when the action is started. It is not necessary that the temporal status of the action to be N (now) after the execution of this callback;
- Stop - must contain the code to be executed when the action should be stopped. Similarly,

does not guarantee that the status is P (past) after its execution;
- State - function that should return the actual current temporal status of the action node. This status is the combination of the values P, N and F, as any other node in a PNF-Network.

Deciding what "start" and "stop" functions are executed, given the current state of the nodes in the network, does the control of the activation of action nodes. The steps to be taken in the heuristic proposed to this task in [ref] are:

1. Determination of the current states of all the nodes of the network;
2. PNF-Propagation of constraints to find the restricted desired states;
3. Thinning of the solution: the result of the last step can possibly contain more than one solution for the network, so a heuristic is used. For each action node, its desired state is computed as the intersection of its current state and the result of the PNF-Propagation. If the result of this intersection is empty, the state computed by the PNF-Propagation is used;
4. Execution of the necessary callback functions based on the thinning process, for all action nodes.

Table 1 describes what callback function to execute, given current and desired temporal states, where the later is the result of the thinning process. The first column denotes the presence of the given state, meaning that the current state of the action only needs to contain the denoted value. With the thinning process, for each action node, the heuristic actually tries not to call either the "start" or the "stop" callbacks, unless its actual current state contradicts the result of the PNF-Propagation.

Table 1 - Conditions for the activation of callbacks

Current	Desired	Callback
F	N,PN	Start
N	P	Stop
F	P	Stop

4. PNF Extensions

In this section we explain our main contributions to the existing framework: the template-based definition language and the hierarchical PNF-Propagation algorithm, which reduces the complexity for large networks.

4.1 Template-based definition language

One possible benefit of PNF-Networks is its ability to represent events based on previously-defined reusable sub-events. However, the whole system must be integrated with custom real-time game/graphic engines that implement the virtual sensors and the actual user-generated scripts with the callback functions. One of the most important features of such integration is the ability to represent reusable sub-events without having to specify the exact objects and actors that compose them. We developed a template system that provides for an easy to use method for describing and composing PNF-Networks in XML. The code below shows a simple network of two nodes and the causality constraints between them.

```
<network>
   <node name="cause" type="sensor" >
      <scene object="box" target="player" type="collision" />
   </node>
   <node name="consequence" />
   <constraint origin="cause" destination="consequence"
                type="b,m" />
</network>
```

Code 2 - Simple PNF-Network in XML

This XML sample shows the basic language to describe a network of PNF-Constraints. The <*node*> tag is used to describe all the intervals, with a required *name* attribute to define a label for each one. Besides being an id for the parsing and template mechanisms, this attribute is used in real-time to implement a query system for the temporal state of any node. The *type* attribute is optional for *passive* nodes and obligatory for *actions,* and *sensors*.

The *sensor* node type needs a special <*scene*> tag to define the attributes that specify the kind of sensor and the real-time objects that are subject to its "sensing" mechanism. Actual sensors are implemented by the customized graphic engines and the role of this XML specification is to describe its *type*, which can be *collision*, *proximity*, *look* or *grab*; and other attributes that are needed in real-time. For a sensor node, the necessary attributes are the *object* that should host the sensing mechanism and its *target*, which represents the label that another object must be marked with to properly activate the sensor. For instance, in the example above the real-time engine should host a collision sensor on the object named "box" and detect the collisions with objects labeled "player". By definition, sensor nodes are marked with the temporal state PF (past or future) when are not activated; and N (now) when activated.

The most useful features of the template system are its ability to reuse previously defined actions and the use of variables to describe the objects that compose them. The following XML sample shows the definition of a template, also showing how variables and action nodes are defined.

```
<template name="pass-through">
   <node name="pass" />
   <node name="begin" type="sensor" >
      <scene host="$1" object="$3" type="collision" />
   </node>
   <node name="end" type="sensor" >
      <scene host="$2" object="$3" type="collision" />
```

```
  </node>
  <constraint origin="begin" destination="pass" type="s" />
  <constraint origin="end" destination="pass" type="f" />
</template>
```

Code 3 - Sample template with wild cards

A template is similar to any other PNF-Network, but it is necessary to start with the definition of a *passive* node, normally one whose interval bounds the sub-network. Later on, when reusing this template to compose higher level structures, this passive node is the one to be connected by the constraints defined in the higher level network. The other difference is the inclusion of variables (marked with the *$* symbol) that can be replaced whenever the template is re-used in another definition.

This particular example shows a simple template that detects if a given object (represented by the variable $3) passes through two locations (respectively $1 and $2) in a particular order. The constraints guarantee that the passive node "pass" will be in the N (now) state as soon as the object touches the first sensor and in the P (past) state only after a collision with the second sensor (location) ceases to exist. Any other ordering of the events will collapse the network, meaning that the described event actually did not happen. Code 4 shows how to compose a higher level network by reusing previously defined templates and action nodes, that can be used to control automated scripts in the graphic engine.

```
<network>
  <node name="pass-instance" template="pass-through">
    <param name="$1" value="trigger1" />
    <param name="$2" value="trigger2" />
    <param name="$3" value="player" />
  </node>
  <node name="consequence" type="action">
    <scene object="npc" />
  </node>
  <constraint origin="pass-instance"
              destination="consequence"  type="m" />
</network>
```

Code 4 - Reuse of a template in a higher level network

The above example defines a simple network that activates the script represented by the "consequence" node immediately after the end of the event represented by the "pass-instance" node, which is an instance of the previously defined "pass-through" event. The activation is guaranteed by the MEET (m) constraint between the two nodes. The sample also shows how the variables can be replaced by the actual values to be used by the real-time graphic engine to implement the sensors. The definition of *action* nodes, those that control the execution of user-created scripts, also needs the specification of a scene attribute that indicates the object that should contain the callback functions *start* and *stop*. In our implementation, we use the message passing API of the Unity3D Game Engine [Unity Technologies 2009] to perform this operation.

When a template is instantiated, it acts as a wrapper for an independent sub-network. The constraints applied over the instanced node will be connected to the wrapper, instead of the internal nodes themselves. This wrapper acts as a barrier that detects when internal or external PNF-Propagations should be passed or not. This will happen only in the case where the temporal state of this wrapper changes. Figure 6 shows the graph representation for the hierarchical network presented in Code 3.

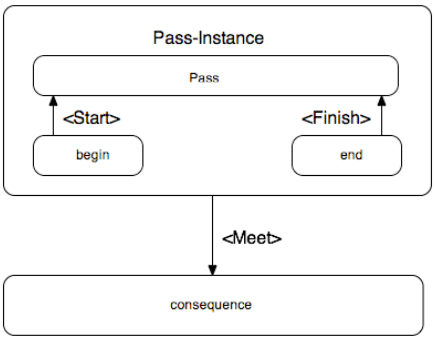


Figure 6 - hierarchical PNF-Network

4.2 Hierarchical PNF Propagation

The PNF-Propagation algorithm is linear on the number of constraints of the network; however, this can still be a limiting performance factor when considering the possibility of very large networks that represent complete narratives, especially with the control of low level virtual actors. Due to the structure of the most common narratives found in the literature of storytelling research, we decided to create the hierarchical PNF-Networks model, which can be though as a layered graph, with the highest level comprising a sequential or parallel plot, composed of nodes connected only by *meet* constraints. The lower levels of the graph can use more sophisticated PNF constructs, for instance to enable the control of multiple actors in a scene.

For instance, in the sample network of Figure 4, which represents an event in a plot, there is one *passive* node, the higher level "drama"; being all other nodes user-implemented *actions*. One can notice that all the "internal" nodes of that network are bound by the interval represented by the "drama" *passive* node. This pattern of using a passive node to bind all internal intervals is important to the definition of coherent hierarchical PNF-Networks.

This guarantees that internal nodes of two different sub-networks never overlap, and PNF constraints are necessary only between nodes in the same level/layer. The wrapper node encapsulates all the links of the internal sub-network (actually the first passive node) with the external one (the higher level network). Using this feature, the PNF-Propagation algorithm can be restricted to only the sub-networks that are to be affected by its execution. For each wrapper node that connects a sub-network to its container network, the

algorithm in Code 5 is executed every time a PNF-Propagation occurs at the container level, while Code 6 is executed after an internal PNF-Propagation is fired.

```
// pNode is the passive node of the internal network
// wNode is the wrapper node that connects with the
// external network
if (pNode.state != wNode.state)
{
   pNode.state = wNode.state;
   pNode.propagate();
}
```

Code 5 - blocking external PNF-Propagation

Whenever an external PNF-Propagation is fired, this algorithm will block its execution for the internal network, including any sub-network on levels below. This provides for a barrier that will prevent unnecessary traversal of sub-networks that are not to be affected anyway. The algorithm in Code 5 is the same; the only difference being the direction of the propagation, in this case to the external network.

```
// pNode is the passive node of the internal network
// wNode is the wrapper node that connects with the
// external network
if (pNode.state != wNode.state)
{
   wNode.state = pNode.state;
   wNode.propagate();
}
```

Code 6 - blocking internal PNF-Propagation

Simple structures like these shown here can be used to compose complex narratives and provide for a runtime engine that controls its execution. In the next section we will show how this extended version of the original PNF-Network framework relates to common storytelling concepts and constructs.

5. Integration with storytelling concepts

With this paper, we propose the use of the hierarchical PNF-Network framework to model and control some levels of an interactive storytelling system. In this section we describe how the PNF model relates to common concepts in storytelling research.

5.1 Planning and execution with plot-based storytelling

Several previous works have focused on the use of planning algorithms to generate narratives [Riedl and Young 2003, Charles et al. 2003, Ciarlini et al. 2005, Barros and Musse 2005]. When these algorithms are used with a plot-based approach, the output often is an ordered (partially or totally) set of events, composed of subjects, verbs and objects. There is still research to be done in this subject, but in this section we show a possible mapping of the output of a planner to a hierarchical PNF-Network.

The generated plot can be mapped to a high-level PNF-Network, with each verb corresponding to a previously defined action, which is by itself composed of others, lower level, sub-actions. The subjects and objects of the plot sentences are the parameters that shall replace the variables needed by these action templates to be complete. For instance, the following story can be directly mapped to the (high level) PNF-Network of Figure 7. The sentences are meant to represent a very condensed version of the movie Star Wars, episode IV: A new hope.

1. Imperial troopers kill Luke's uncle and aunt
2. Luke meets Obi wan
3. Yoda trains Luke in Jedi
4. Luke joins the rebels
5. Luke destroys the death star

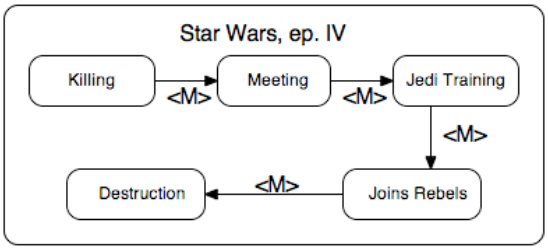


Figure 7 – Highest level of the Star Wars PNF-Network

In Figure 8, we chose to represent the last event of the plot, the destruction of the death star, because it highlights some features of the PNF-Network model. This event is composed of several sub-actions performed by the rebel and imperial spaceships, and ends up with the final destruction of the star-like weapon.

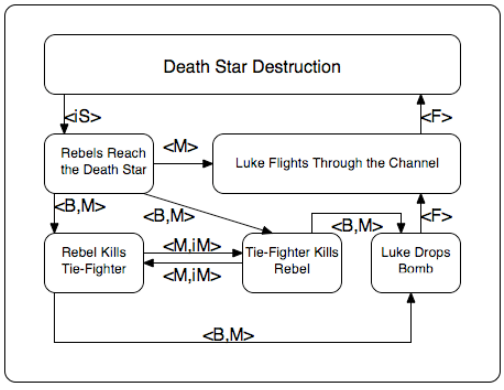


Figure 8 – Death star destruction PNF-Network

The actions that compose this mathematical model for the death star destruction can be realized in different ordering, but will always finish with Luke's precise bomb drop that explodes the imperial weapon. The "Death Star Destruction" node is the passive internal interval that bounds the execution of the others. The inverse start (iS) constraint to the "Rebels Reach the Death Star" action will guarantee that the later will be started as soon as a higher level PNF-Propagation changes the temporal state of the first to N

(now). The rest of the relations will bring parallel actions such as "Luke Flights Through the Channels" and a mutual exclusion between the two killing actions.

For now, our system is suitable for plot-based stories whose sentences are based in verbs that can be mapped to high-level PNF defined events. There is still work to be done to enable runtime planning in the case of user intervention. This is our main research interest at this moment.

5.2 Virtual cinematography

Many previous research, and this work is no exception, on modeling and planning for storytelling tend to describe the plots as high level events, leaving room for the use of different techniques to enhance the viewer experience. One important research area in interactive storytelling is the use of traditional cinematography concepts as automated agents.

We are experimenting with two approaches for the integration of cinematography agents with hierarchical PNF-Networks: camera actions and director agents. With camera actions we include action nodes in our PNF templates so that every time that template is used in a story it already comes integrated with the camera shots (as action nodes linked to pre-defined camera scripts). Figure 9 shows a PNF template for the sub-events of the drama that was analyzed in section 3, including camera actions that are tied to the sub-events with the constraint equal (E).

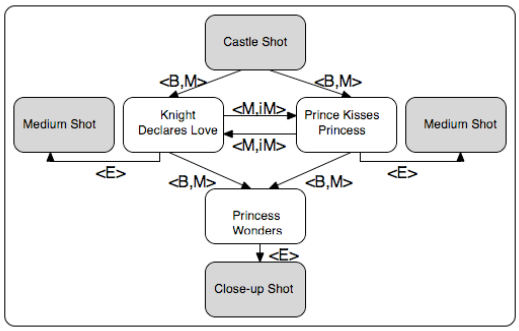


Figure 9 - Drama event with camera actions (gray nodes)

Another option, that is more flexible than the latter is to establish a communication protocol with an independent agent that chooses the camera shots to be used. The general idea is to send all *Start* and *Stop* messages to this director agent as well. The message to this director agent must include a reference to the PNF action that received it originally, so that useful information can be extracted from the mathematical model such as the name of the action (verb that possibly defines the type of shot to be used) and the parameters that link this action to the real-time characters and objects on the graphic engine. Figure 10 shows a schematic representation for this communication architecture.

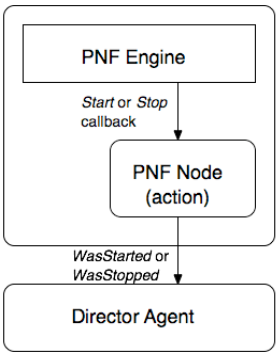


Figure 10 - Communication between the PNF engine and a director agent

5.3 Flashbacks and out-of-order execution

A narrative concept that helps the viewer to mind important events, while at the same time enhances the storyteller possibilities is the use of flashbacks [Bae and Young 2008], a replay of a part of the story already told/seen. The PNF-Networks original framework was not design with this possibility in mind, but the hierarchical feature of our model provides for partial independency of (lower level) events, meaning that the whole internal structure of a sub-network can be reset, leaving room for its (re) execution, possibly with a different approach from the director agent, highlighting the flashback timing with image filters or different shots.

This idea can be extended to out-order-execution if the first level of the network (plot) is controlled by another engine, which uses different decision rules to choose the ordering of the events. The plot events are now independent PNF-Networks that can be started in any desired order, but still benefitting from the features of the temporal model behind it at the lower levels.

5.4 Character-based emergent storytelling

Character based storytelling relies on the interaction of virtual actors that are agents directly driven by goals or assisted by some sort of planning. Creating coherent and interesting stories in real-time with this approach is still the objective of this research field, and one of the challenges is related to the way virtual actors perceive the environment.

PNF-Networks can be used with this approach to model and recognize the actions of all the virtual actors, providing for a flexible framework for the agents to perceive and react accordingly to its plan or drives.

These are some integration ideas made possible by Hierarchical PNF-Networks. We are still investigating if and how other key storytelling concepts can be used with this framework.

6. Conclusions and Future Work

With this paper we presented the Hierarchical PNF-Networks and its template-based definition language, two extensions over the original PNF-Network framework, a fine-grained representation for temporal relations between intervals that can be used to describe high level events such as complex human action. The two extensions made possible the use of PNF-Networks to represent large sets of interdependent events such as interactive. The XML schema used to describe the networks facilitates the description of a library of events and actions to be used by a storytelling system.

We are currently researching more extensions to Hierarchical PNF-Networks. For instance, we are trying to integrate planning algorithms to create and modify PNF-based stories in real-time. It is easy to adapt a current planning algorithm to only generate a PNF-Network that is solely composed of precedence relations such as *Meet*, but that approach would not make any use of the richer representation that these networks can bring to interactive storytelling models. Our goal is to create a system that can be used in multi-user scenarios such as emergent stories in massive games or user-influenced plot-based narratives for digital-TV.

References

ALLEN, J. F., 1983. Maintaining Knowledge about Temporal Intervals. 1983. *Communications of the ACM*, 26 (26), Pages 832-843.

ALLEN, J. F. AND FERGUSON, G., 1994. Actions and Events in Interval Temporal Logic. 1994. *Journal of Logic and Computation*, vol. 4 (5), Pages 531-579.

BAE, B., AND YOUNG, R. M., 2008. A Use of Flashback and Foreshadowing for Surprise Arousal in Narrative Using a Plan-Based Approach. *In: Proceedings of the 1st Joint International Conference on Interactive Digital Storytelling: Interactive Storytelling, 2008 Erfurt, Germany*. Pages 156-167.

BARROS, L. M. AND MUSSE, S. R., 2007. Planning algorithms for interactive storytelling. 2007. *Computers in Entertainment (CIE)*, vol. 5 (1), Article No. 4.

BARROS, L. M. AND MUSSE, S. R., 2005. Introducing narrative principles into planning-based interactive storytelling. *In: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology, 2005 Valencia, Spain*. Pages 35-42.

CAVAZZA, M. AND CHARLES, F. 2005. Dialogue generation in character-based interactive storytelling. *In: Proceedings of the AAAI First Annual Artificial Intelligence and Interactive Digital Entertainment Conference, 2005 Marina del Rey, CA, USA*

CHARLES, F., IBÁÑEZ, M. L., MEAD, S. J., BISQUERRA, A. F. AND CAVAZZA, M. 2003. Planning formalisms and authoring in interactive storytelling. *In: Proceedings of TIDSE'03: Technologies for Interactive Digital Storytelling and Entertainment, 2003 S. Göbel et al. eds. Fraunhofer IRB Verlag, Darmstadt, Germany.*

CIARLINI, A. E. M., POZZER, C. T., FURTADO, A. L. AND FEIJÓ, B. 2005. A logic-based tool for interactive generation and dramatization of stories. *In: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology, 2005 Valencia, Spain*. Pages 133-140

CRAWFORD, C. 2004. Chris Crawford on Interactive Storytelling. New Riders Games, Indianapolis, IN.

JAHALA, A., RAWLS, C. AND YOUNG, R. M. 2008. Longboard: A Sketch Based Intelligent Storyboarding Tool for Creating Machinima. *In: Proceedings of the 2008 Florida Artificial Intelligence Research Society Conference, 2008 Florida, USA*.

MEYER, T. A., 2002. Development of Computer-Actors within the Interval Script Paradigm. Unpublished Dissertation, Massey University. Available from:http://www.massey.ac.nz/~tameyer/research/computertheatre/docs/thesis.pdf [Accessed 10 June 2009].

PINHANEZ, C. S., 1999. *Representation and Recgnition of Action in Interactive Spaces*. PhD thesis, Massachusetts Institute of Technology.

PINHANEZ, C. S., MASE, K. AND BOBICK, A., 1998. Human action detection using PNF propagation of temporal constraints. *In: Proceedings of the 1998 IEEE Computer Society Conference in Computer Vision and Pattern Recognition, June 23-25 1998 Santa Barbara, CA, USA*. Pages 898-904.

PINHANEZ, C. S., MASE, K. AND BOBICK, A., 1997. Interval scripts: a design paradigm for story-based interactive systems. *In: Proceedings of the SIGCHI conference on Human factors in computing systems, 1997 Atlanta, Georgia, United States*. Pages 287-294

POZZER, C. T., 2005. *Um Sistema para Geração, Interação e Visualização 3D de Histórias para TV Interativa*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro.

RIEDL, M. O., ROWE, J. P. AND ELSON, D. K., 2008. Towards Intelligent Support of Authoring Machinima Media Content: Story and Visualization. *In: Proceedings of the 2nd international conference on intelligent technologies for interactive entertainment, 2008 Cancun, Mexico*.

RIEDL, M. O. AND YOUNG, R. M., 2003. Character-focused narrative generation for execution in virtual worlds. *In: Proceedings of ICVS 2003: International Conference on Virtual Storytelling, 2003 Toulouse, France*. Pages 47-56

UNITY TECHNOLOGIES. 2009. *Unity3D Game Engine* [software, game development tool] Available from: www.unity3d.com [Accessed 01 January 2009].

OpenMoCap: An Open Source Software for Optical Motion Capture

David Lunardi Flam
Thatyene Louise Alves de Souza Ramos
Daniel Pacheco de Queiroz
Arnaldo de Albuquerque Araújo
Universidade Federal de Minas Gerais

João Victor Boechat Gomide
Universidade FUMEC

Abstract

Nowadays motion capture is a valuable technique for virtual character animation in digital movies and games due to the high degree of realism that can be achieved. Unfortunately, most of the systems currently available to perform that task are expensive and proprietary. In this work, an open source application for optical motion capture is developed based on digital image analysis techniques. The steps of initialization, tracking, reconstruction and output are all accomplished by the built OpenMoCap software. The defined architecture is designed for real time motion recording and it is flexible, allowing the addition of new optimized modules for specific parts of the capture pipeline, taking advantage of the existing ones. Experiments with two cameras with infrared LEDs and reflexive markers were carried out and the created methodology was assessed. Although not having the same robustness and precision of the compared commercial solution, this work can do simple animations and it serves as an incentive for research in the area.

Keywords:: Motion Capture, Software Development, Stereo Vision, Digital Image Processing, Computer Graphics

Author's Contact:

{david,pacheco,thatyene,arnaldo}@dcc.ufmg.br
jvbg@hotmail.com

1 Introduction

Analysis and motion capture (MoCap) of human movement have been growing consistently over the last decade. Many researchers from different areas are now working with it because of the wide range of possible applications. Following a classification scheme presented in [Moeslund et al. 2006] it is possible to separate those applications into three groups based on their main objective: surveillance, control and analysis.

Surveillance applications focus on examining motion of a person as a sole object or as a group of objects in case of people agglomeration. Determining the flux of individuals in a mall in order to discover some pattern to optimize shops locations is one example. Another one is a model that describes movement behavior in a prison for security reasons.

Control applications interpret motion from an actor and transform it into a sequence of operations. Movies like King Kong (2005) and Polar Express (2004) use actors movements to operate characters. Other instances are animated characters from 3D computer games, like FX Fighter, Fifa and NBA Live series.

Analysis applications generally dedicate on studying a person as a set of objects. Some examples include improving athletes techniques, studying clinical cases and defining elder people body behavior.

Motion capture is a powerful technique for animating virtual characters and controlling them. It creates smooth movements, giving sensation of real ones. In addition, if used correctly, it can speed up animation, when compared to traditional methods like key-frame animation.

Currently, in Brazil, there are just two organizations that have robust motion capture systems for character animation: Rede Globo and RPM Produtora Digital. Every piece of hardware and software from those systems is imported and expensive, costing tens of thousand of dollars. Usually, game companies in Brazil rent them or buy imported MoCap data.

To our knowledge there are only two national solutions called DVIDEOW [Figueroa et al. 2003] and BraTrack [Pinto et al. 2008]. The first one is widely used for gait analysis, but it is post-processed. In other words, it does not support real time previewing and it can't be used to control operations. Besides that, its source code is closed and apparently it's not actively being developed anymore (homepage of the project [Figueroa 2009] is down at the present time). BraTrack, in its turn, can track objects in real time, but it is not freely available.

In order to acquire expertise and reduce costs, we present the development of an open source optical motion capture software for real time uses in this work. It is a standalone solution, that is, every task needed to acquire MoCap data is implemented by the built application. Further, it is modular and flexible, allowing new modules to be easily integrated and optimized, taking advantage of the existing processing chain. Conclusively, it has a simple graphical interface and it is ready to grow by receiving new contributions.

This paper is organized as follows: related work is introduced in the next section, including some history and commercial and academic approaches for optical motion capture. Our methodology is described in depth in Section 3. The software architecture, chosen programming language and libraries are presented in Section 4. Designed experiments and their results are shown in Section 5. Finally, some conclusions and future work are drawn in Section 6.

2 Related Work

The first process considered to be MoCap was done in 1872. Eadweard Muybridge took several pictures of a horse while galloping with many cameras to settle a bet. The question he answered with that experiment was if a horse would take all his feet of the ground while galloping. Indeed, the horse does take all his feet of the ground, see Figure 1. After that, many other analog processes appeared [Menache 2000], but they were all 2D.

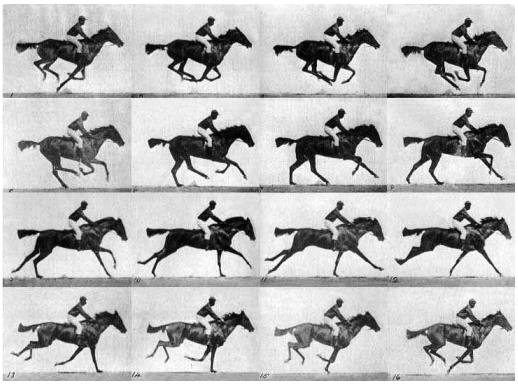


Figure 1: *Muybridge Horse Pictures*

Almost a hundred years later, with the advent of computers, digital 3D motion capture began with a commercial called Brilliance in 1985, during Superbowl. In order to produce it, several VAX 11 machines were borrowed across USA for two weeks to render 30 seconds of video. In 1995, warriors characters from the pc game Fx

Fighter were animated using digital MoCap. Until the present days, several other processes were developed and technologies created to record motion, most of them are detailed in [Kitagawa and Windsor 2008].

Now, we briefly describe three commercial systems that represent the current state-of-the-art in professional optical motion capture. They all cost more than 50 thousand dollars [Inition 2008] and are tightly integrated with proprietary hardware and software. The first one, from Vicon [Vicon Motion Systems 2008], uses passive markers. It is composed by eight T160 cameras, capable of capturing movement in real time at 120Hz with 16 megapixels. Further, it can capture at faster frame rates if camera resolution is reduced.

The second one is called IMPULSE, uses active markers and is made by [PhaseSpace Inc. 2008]. It has almost the same capabilities of Vicon's solution, recording motion at 120Hz with 16 megapixels. The last one is a system that doesn't use markers, called STAGE, made by [Stage 2009]. It is also capable of capturing data at 120Hz, but its estimated precision is equivalent to a 2 megapixels marker system.

While not as robust and precise as commercial systems presented earlier, many papers were published regarding optical motion capture systems. We review and discuss here some of the most recent and relevant.

[Figueroa et al. 2003] report the construction of a brazilian motion capture software for gait analysis, athletes enhancement and other application in the biomedical area. It focuses on tracking markers using a user configurable chain of algorithms based on mathematical morphology, pattern recognition and a Kalman filter. The software also does 3D reconstruction using calibrated cameras and DLT (Direct Linear Transform), but authors don't show experiments for that. Finally, it is post-processed, video sequences must first be recorded for each camera present in the system to obtain results.

[Uchinoumi et al. 2004] perform motion capture without using markers, using just silhouettes of actors iluminated by four cameras, each connected to a computer in a distributed system. Information is processed by four computers that send data to a server through a local network. The server then is responsible for combining every silhouette processed by each client so that 3D reconstruction can be carried out. Authors say that is possible to capture an object at almost 11 frames per second.

[Castro et al. 2006] describe the development of a motion capture system based on passive markers, focused on gait analysis, called SOMCAD3D. It is also post-processed as [Figueroa et al. 2003]. DLT is likewise used for camera calibration and 3D reconstruction. Tracking is done by curve interpolation. In addition, in this work, system precision and accuracy is compared with some other systems specially built for gait analysis.

[Raskar et al. 2007] present a high performance motion capture system with few restrictions regarding the recording environment. It does not use cameras, but instead photo-sensors acting as active markers. They are capable of determining their own position and orientation in space through binary patterns emitted by LEDs placed around the capture volume. Since it has no cameras, expensive hardware is not required and can record movement at rates of 480Hz. Although being composed of reduced cost hardware, their system's assembly and configuration require off-the-shelf electronic and optic components. Finally, it is not possible to capture faces due to the size of the markers.

[Pinto et al. 2008] present the first commercial low-cost marker-based optical tracking system developed in South America. The system is composed by two off-the-shelf USB cameras with custom-made electronic boards with infrared LEDs and reflexive markers. Authors claim that it is possible to capture movement at 60Hz.

Academic systems concentrate on proposing new techniques whereas commercial ones usually advance some already know, mature processing chains as industrial secrets. It does not mean that the reproduction of the papers described here is simple or even possible, principally because some important implementation details are omitted and none of them make their source code easily available.

OpenMoCap is the first step of a bigger project to build a complete and robust optical motion capture system. The main application of the developed solution is the generation of realistic data in real time to animate and control virtual characters. It is the first work that we have knowledge in Brazil of an open source motion capture system created to fulfill that purpose.

3 Methodology

Motion capture can be done using cameras and special selected points. The whole process is complex but can be divided in basically four steps. The first one, *initialization*, regularly is done only in the beginning of the process and relates special given points from a scene with points from a previous defined structure. The second task is called *tracking*, that is, monitor the position of those special points over a period of time. The third one is *reconstruction* or *pose estimation*. Finally, the last one is *output* and consists in outputting data in some special format.

Specifically, when working with 3D models, we must find the correspondence between those special points from each present camera image in the system and apply a triangulation algorithm to obtain their respective 3D coordinates. Obtaining those special points is possible using markers or other local features and heuristics, relative regions positions and skin texture and color. However, the simplest way to retrieve those special defined points is by using markers. Markers are special objects attached to a suit wore by an actor. Also, it is possible to place them directly over the actors body.

Markers can be considered active or passive, depending on their nature. A clear distinction between those classes is that active markers react to external impulses whereas passive markers do not. Furthermore, active markers necessarily have some kind of embedded processing and communication through different kind of sensors and hardware. On the other side, passive markers just have some special properties, like reflecting infrared light. Summarizing, passive markers only help segmenting a region of interest while active ones provide more information about themselves, like their own centroid position (special point) and their identity (semantic relation with the chosen model).

3.1 Initialization

3.1.1 POI Extraction

In order to execute this first step, we must extract some special image points called Points Of Interest (POIs). Each marker in the scene corresponds to a POI in the image. The extraction of those points is accomplished by applying a binary thresholding algorithm. The threshold parameter can be set to values between 0 and 255 (possible intensity values of a pixel in a gray scale image). At first, this procedure is enough to separate POIs from the background.

However, our goal is to separate uniquely detected POIs and also eliminate noise (high intensity parts of the image not related to markers). To perform this, a component connected algorithm with 6-adjacency is applied [Umbaugh 2005]. In practice, the algorithm suffered some modifications to increase its efficiency and reduce noise.

Thresholding is executed in parallel with pixel labelling, so that the image does not have to be looped through twice. As soon as each image element is analyzed, it is marked as being background or object, allowing the connected component algorithm to continue its traditional process. Than, the calculation of the connected components area takes place immediately, enabling the removal of the ones that are probably not markers. This removal procedure is based on user defined limits of minimum and maximum areas. Remaining objects have their centroid calculated. Figure 2 shows the process.

Figure 2: *POIs Detection*

3.1.2 Semantic Selection

The next step of initialization is to associate semantics to detected POIs. This is necessary to uniquely define each object with a simple name, like human body parts. OpenMoCap has a skeleton importer compatible with structures written in Biovision Hierarchy (BVH) format [Menache 2000]. An alternative way of semantic input is a simple text file, containing in each line a name. Finally, semantic attribution is carried out manually, in a simple manner, by the graphic interface, shown in Figure 3.

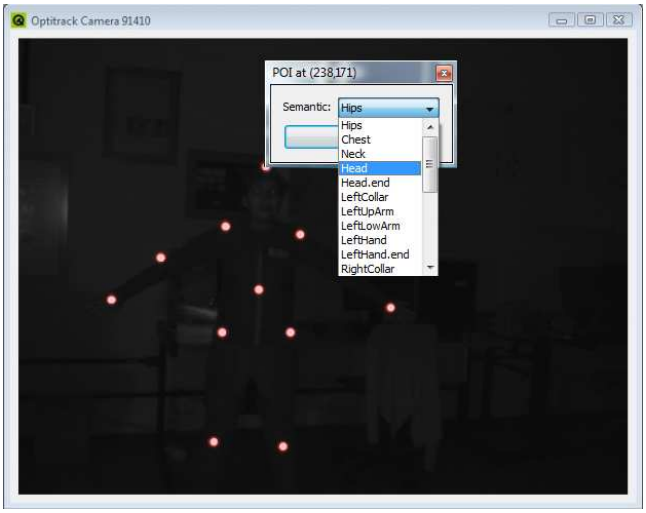


Figure 3: *POIs Semantic Selection*

3.1.3 Camera Parameters Estimation

The last step of initialization is the camera parameters estimation. Inspired on [de la Fraga and Vite Silva 2008], our approach to camera parameters estimation is based on modeling this task as an optimization problem, solved by differential evolution. The basic idea is to select some camera parameters as variables and define a cost function over them, using semantic information and POI localization, available from previously described steps. Final result is an estimated projection matrix for each camera, that minimizes that cost function.

Currently, OpenMoCap uses only two identical high quality cameras. Therefore, we ignore lens distortions and chromatic aberrations, simplifying the problem. Hence, the chosen parameters of each camera to be variables of the optimization problem are: focal length f_c, rotation vector $R_c = [r_c^x, r_c^y, r_c^z]$ and the translation vector $T_c = [t_c^x, t_c^y, t_c^z]$.

Variables have their lower and upper limits defined by the user, through the camera parameters estimation interface shown in Figure 4. The first set of parameters controls the differential evolution algorithm, the number of individuals, the number of generations, the differential variation and the recombination constant can be modified.

The second set of parameters effectively defines the lower and upper limits. The Translation Range Unit field defines the maximum translation unit for each coordinate axis, from the first camera to the second. The Max Rotation field determines the maximum rotation

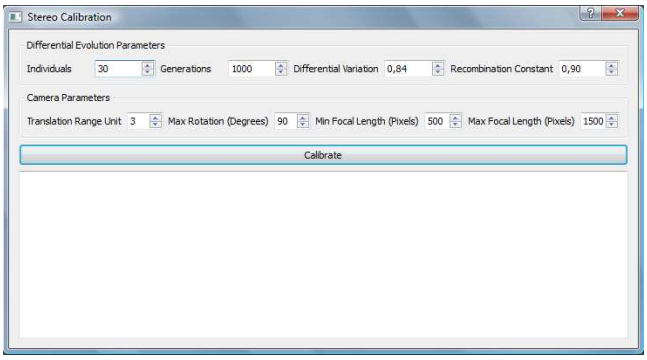


Figure 4: *Camera Parameters Estimation Interface*

in degrees (minimum rotation is always 0). Finally, the Min Focal Length and Max Focal Length fields specify the minimum and maximum focal length, respectively, in pixels.

After the definition of the variables and their limits, it is necessary to define the cost function. This function must measure the quality of an estimated solution. Specifically, its inputs are an individual to be evaluated and a set of ordered pairs. Each pair is composed by two POIs with equal semantic and their own centroid coordinates.

Begining with the variables from the target individual of evaluation, two projections matrices are constructed. They represent the estimated orientation and position of the two cameras. Through these matrices and a pair of correspondent POIs, and using triangulation solved by SVD [Hartley and Sturm 1997], we can project a 3D point. Afterwards, this 3D point is reprojected into the image planes. The sum of euclidean distances between those reprojected points and the real centroids of the ordered pair, is called reprojection error.

The total cost of an individual (cost of a possible solution) is obtained by the sum of reprojection errors of all POIs ordered pairs. The smaller the cost the better is the solution. Therefore, our goal is to find the matrices that minimize this global reprojection error.

After this initialization phase, the software is in a valid state, with all data necessary to begin motion capture.

3.2 Tracking

Tracking is the next phase, it is responsible for maintaining POIs' semantics through time. It tries to be as robust as possible regarding observed movement trajectories. This helps the motion capture software to continue in its correct initialization state. Lastly, it is one of the continuous phases of the capture workflow, carried out in every frame.

Tracking starts as soon as a POI receives a semantic, not only when the user requests to begin a motion capture recording section. This is possible because processing is done locally, i.e. only information from the last frame of the same camera is used to define the semantic of extracted POIs in a new frame.

The alpha-beta estimator [Yoo and Kim 2003] is used to appraise POIs' next positions. Figure 5 illustrates how tracking is accomplished. Images with timestamps represent occurred POIs' movements in the scene. While the ones without timestamps represent results obtained by the implemented tracker.

When a new frame is available, the estimated localization of a POI is matched with one of the new extracted centroids. This matching refreshes POI's position with the centroid that has the smaller euclidean distance from the estimated place. Nevertheless, the user can set a maximum allowed distance between these two points, exhibited in Figure 5 by the dotted circles. If no new detected centroid is inside that search region, the new POI's position will be the one predicted by the estimator.

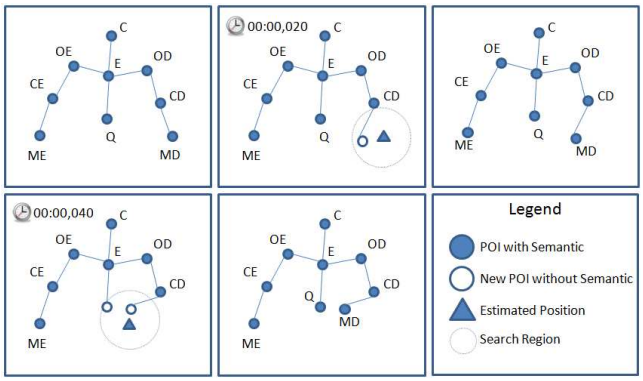


Figure 5: *Sample Tracking Diagram*

3.3 Reconstruction

3D reconstruction is carried out by linear-eigen triangulation [Hartley and Sturm 1997]. This process is responsible for calculating tridimensional POIs coordinates, through their projections on the image planes and their respectives camera matrices. Like the previously phase, this one is continuous, executed frame by frame. But, on the contrary of tracking, it is not local, it is global. In other words, information from all images acquired by the cameras at the same time must be gathered to achieve the desired result.

As described earlier in the section of camera parameters estimation, our method does not use tridimensional points with known metric coordinates to estimate camera projection matrices, it is not like a traditional calibration algorithm. Therefore, the final result obtained by triangulation is defined up to a scale factor. Despite this, relative distances between points from the captured structure are maintained. This is often enough to some simple animations and characters control.

3.4 Output

Transforming acquired data by triangulation into usable information is the last necessary phase to complete the motion capture workflow. Basically it runs only once, but it is continuously refreshed after each performed triangulation. In our approach, an output file is generated only after an user requests to stop a motion capture recording section.

There are several types of standard MoCap files [Kitagawa and Windsor 2008]. Fundamentally, they can be classified in two categories: hierarchical and translational. Hierarchical ones contain a defined structure with bones, joints and their relations. Their motion part consists of describing movement as relative rotations, beginning with the skeleton center of mass position.

The other way to represent MoCap data is by describing motion as global translations, without having to define a skeleton. This format is much simpler than the hierarchical one. Since our reconstruction phase produces just point clouds with associated semantic, our software exports data in a translational standard called TRC, developed by [Motion Analysis Corporation 2009]. Figure 6 shows a fragment of a sample file generated by OpenMoCap.

PathFileType 4 (X/Y/Z) output.trc

DataRate	CameraRate	NumFrames	NumMarkers	Units	OrigDataRate	OrigDataStartFrame	OrigNumFrames
25.00	25.00	100	10	mm	25.00	0	100

Frame#	Time	A			A.end			B		
		X1	Y1	Z1	X2	Y2	Z2	X3	Y3	Z3
1	0.00	-0.963505	0.567625	5.21397	-1.15364	-0.216878	4.87087	-0.270564	-0.7271	4.46509
2	0.04	-0.966132	0.573169	5.21397	-1.15364	-0.216878	4.87087	-0.270564	-0.7271	4.46509
3	0.08	-0.965992	0.578647	5.21397	-1.15364	-0.216878	4.87087	-0.270564	-0.7271	4.46509
4	0.12	-0.963653	0.582631	5.21397	-1.15364	-0.216878	4.87087	-0.270564	-0.7271	4.46509
5	0.16	-0.966262	0.588164	5.21397	-1.15364	-0.216878	4.87087	-0.270564	-0.7271	4.46509
6	0.20	-0.968691	0.593558	5.21397	-1.15364	-0.216878	4.87087	-0.270564	-0.7271	4.46509
7	0.24	-0.968542	0.599021	5.21397	-1.15364	-0.216878	4.87087	-0.270564	-0.7271	4.46509
8	0.28	-0.966199	0.602946	5.21397	-1.15364	-0.216878	4.87087	-0.270564	-0.7271	4.46509
9	0.32	-0.97114	0.604555	5.21397	-1.15364	-0.216878	4.87087	-0.270564	-0.7271	4.46509

Figure 6: *TRC Sample File*

The output file is basically divided in two parts, header and data. In the first, we have information about the generated file, the amount of data acquired in a capture recording section per second, the amount of images obtained per second by the cameras, the total number of processed frames, the total number of markers and the used measurement unit. Specifically, this last parameter is set to millimeter, although it has no meaning, since 3D points are recovered without defined scale. The last information in this first part is the title of the captured data columns and POIs' semantics.

The second part of the file is usually composed by many lines of POIs' 3D coordinates, organized according to the header previously described. Each line has the index of the frame which the coordinates were calculated and respective instant of time. Finally, although it is a simple format, it is natively imported (without plugin or scripts) by the 3ds Max software [Autodesk 2009].

4 OpenMoCap

OpenMoCap was built trying to employ most of the good software construction practices described in [Kernighan and Pike 1999] and [McConnell 2004]. Therefore, many architecture and implementation decisions were made to create a high quality, flexible and extensible code.

Figure 7 illustrates the defined architecture by a modules diagram. The separation of modules by threads was done to take advantage of the tendency of modern processors to have more cores. Further, tasks executed in real time such as POI detection, tracking, triangulation and visualization could be made parallel.

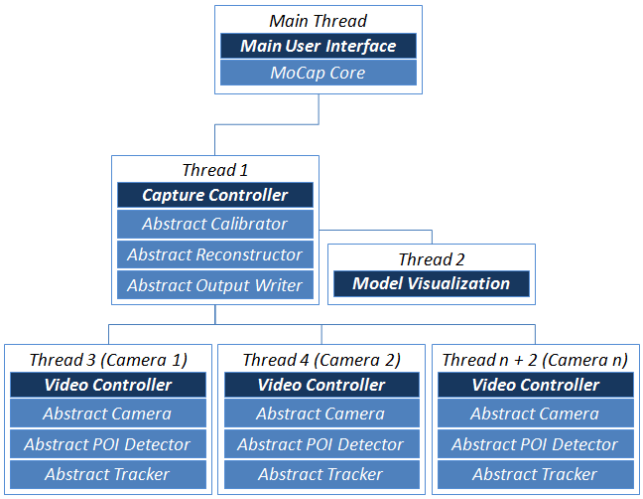


Figure 7: *OpenMoCap Architecture*

The main flow of execution is composed by the application core and the main user interface. MoCap core is responsible for initializing correctly every other flow of execution and their associated modules. Furthermore, it acts as a central information repository keeping track of what is the configuration of the connected cameras, which algorithms are available to perform specific parts of the motion capture workflow and what kind of object will be captured (available POI semantics).

A screenshot of the main user interface is shown in Figure 8. This graphical interface is responsible for showing captured data and for receiving and processing user requests, calling specific functions from the existing controllers. Basically, there are four components that interact with users within the software: the menu, the status bar, camera windows and visualization window.

The menu allows the start and the end of a motion capture recording section. It also informs total capture time and the algorithms being used. The status bar shows the resolution and the frame rate of the cameras being used.

The camera windows display acquired images for each video input device connected to the computer and allow the semantic selection

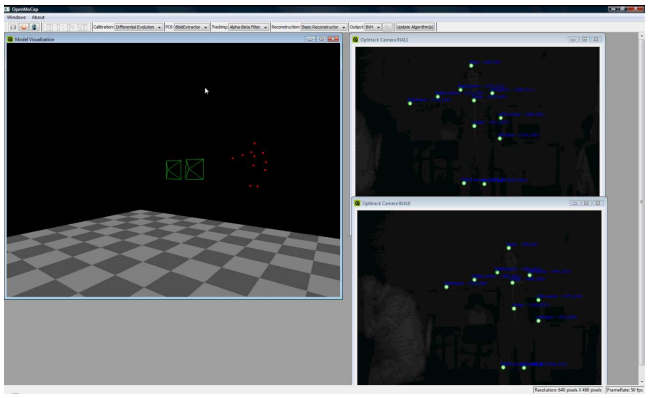


Figure 8: *OpenMoCap Screenshot*

for the detected POIs. The visualization window displays a preview in real time of the motion that is being recorded through a dedicated thread. The Multiple Document Interface (MDI) model was adopted to provide a common and flexible space for all those windows.

Each camera window is processed exclusively by a thread, coordinated by a video controller module. Each Video Controller has an instance of a camera, a POI detection algorithm and a tracking algorithm that are executed sequentially. In Figure 7 this is represented by the modules Abstract Camera, Abstract POI Detector and Abstract Tracker. They define a common and generic interface for the cameras and the 2D processing algorithms cited earlier.

This concept of abstract modules is very important to ensure the software extensibility, one of the goals of this work. If a new camera is to be used and it is not compatible with the current implementation, just some specific methods need to be implemented (like obtaining frames and changing resolution) following the pattern of the abstract camera module to take advantage of the existing processing chain. Another beneficial example is the possibility to substitute the POI detection algorithm with one based on body parts recognition instead of regions intensity. In other words, transform the system into one without markers.

The remaining flow of execution in Figure 7 that has not been described yet is the one composed by the capture controller module. It is the leading contributor for the correct software operation. Effectively, it executes a motion capture recording section, managing and processing data produced by the video controllers, feeding and updating the visualization module. Based on the same principle of abstract modules, the capture controller has an instance of a camera parameters estimation algorithm (Abstract Calibrator), an instance of a reconstruction algorithm (Abstract Reconstructor) and an instance of an output algorithm (Abstract Output Writer).

Until now we have discussed higher level architecture decisions, related chiefly with the concepts of abstraction, generalization, separation of interests and incremental development. In addition to it, some lower level decisions were also made to guarantee profit from existing tools and to obtain high performance.

Given the nature of the built application, an efficient and mature computer vision library was specially usefull for its construction, OpenCV. [Intel Corporation 2009] made it to demonstrate its processors performance. It is written in C++ and it's maintained nowadays by [Willow Garage 2009].

Unfortunately, the existing modules that support cameras in OpenCV are not robust for more than two of those devices and were not developed with object-oriented concepts in mind. Since we want our software to grow and solidly support multiple cameras, videoInput library [Watson 2009] was integrated. Theoretically, it supports up to 20 cameras and is compatible with every video input device that provide a DirectShow interface [Microsoft Corporation 2009].

Regrettably, it was verified that the DirectShow interface provided by the Optitrack FLEX:V100 cameras (used in our experiments) was only able to support one device at a time. For this reason we implemented two new methods in a concrete class for these types of cameras using Optitrack SDK. The software worked flawlessly after this tiny modification, demonstrating in practice the power of the designed architecture.

The graphical interface was entirely built using Qt library [Trolltech 2009]. It is simple, has an open source license and is portable to many operating systems. In addition it provides a easy way to use OpenGL for fast 2D or 3D graphics.

Currently, OpenMoCap only runs in Windows [Microsoft Corporation 2009] because included camera implementations are only supported in this operating system. Besides that, threads were implemented using Win32 API, because Qt threads are slow for our kind of use. Future versions of this software may support other operating systems just by creating concrete classes for cameras and threads, once they were all designed as abstract modules and the rest of the code is portable.

5 Experiments

In this section we present the designed experiments to assess our software and the created motion capture workflow. Furthermore, we discuss their results qualitatively and quantitatively, whenever possible.

In order to obtain quantitative results, we compared our solution with a commercial optical motion capture system made by NaturalPoint [OptiTrack 2008]. Tracking Tools 2.0 can output data in a raw point cloud format and also can display information about tracked 2D POIs, making it a great choice for our analysis. Therefore, we considered its produced data as ground truth.

All experiments were carried out with the same computer, configured with a Intel Core 2 Quad Q6600 processor, 4GB of RAM and a Geforce 8800 GTX. Further, we used the same cameras (OptiTrack FLEX:V100) for both programs. Those devices have a 480p resolution and are capable of obtaining frames at 100Hz. They also have IR LEDs attached and a IR filter that helps POI detection. Finally, identical passive reflexive markers were employed.

5.1 Precision and Stability of 2D Centroids

The main goal of this first experiment is to verify our POI detection algorithm in a static situation. We compare our software results directly with the ones obtained from Tracking Tools, which uses special hardware inside the camera to perform that task. We configured both approaches with the same parameters values shown in Table 1.

Table 1: *POIs Detection Configuration*

Parameter	*Value*
Resolution	640 x 480 *pixels*
Processing Speed	25 frames per second
Intensity Threshold	230
Minimum Area	0,005% of the Image
Maximum Area	0,400% of the Image

A marker was fixed in front of one camera supported by a tripod to make the scene as static as possible. Figure 9 exhibits two graphs showing noise in detected POI coordinates during 50 frames by both approaches.

We applied the normality test of Shapiro-Wilk [Boslaugh and Watters 2008] to a sample of 1000 frames, trying to characterize the generated noise. The result was negative, meaning it is not a gaussian noise. Therefore we analyzed this larger sample using order statistics displayed in Table 2.

The values presented in Table 2 imply that the difference between OpenMoCap and Tracking Tools is very small when detecting POI

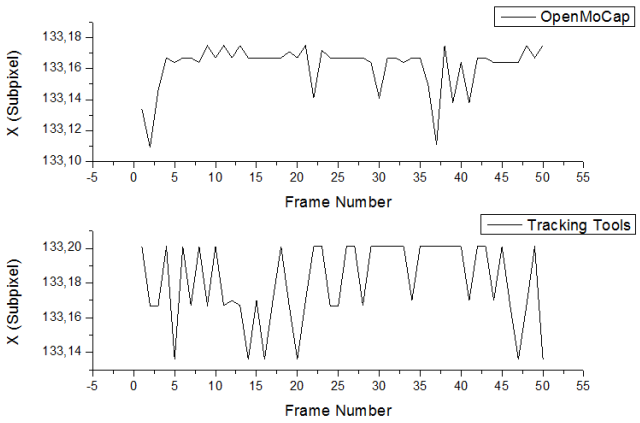


Figure 9: *X Coordinates from a Static POI Obtained by OpenMoCap and Tracking Tools*

Table 2: *Comparative Order Statistics for 2D Centroid Detection*

in subpixels	*OpenMoCap*	*Tracking Tools*
5th Percentile of X	133,164	133,136
Median of X	133,167	133,167
95th Percentile of X	133,201	133,201
5th Percentile of Y	352,826	352,826
Median of Y	352,838	352,844
95th Percentile of Y	352,906	352,856

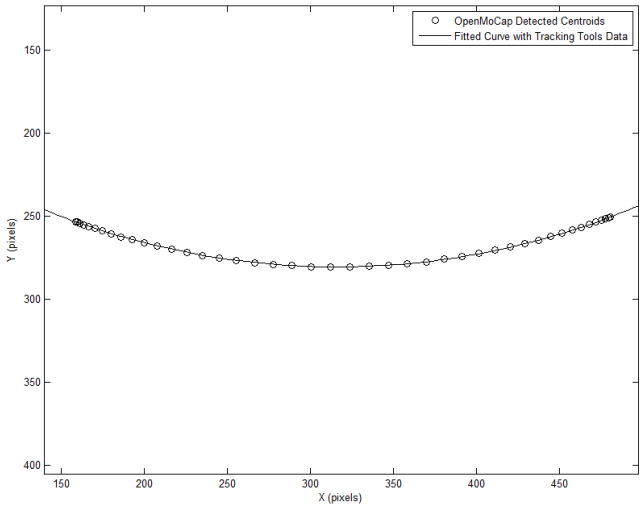


Figure 10: *Tracking Tools Fitted Trajectory and OpenMoCap Sample Trajectory*

Table 3: *Displacement Error*

Error	*Values (pixels)*
Minimum e Maximum	0,004 e 0,254
Median	0,087
85th Percentile	0,125

centroids, from the order of hundredths of pixels. Finally, the calculated percentiles suggest that OpenMoCap has less noise than Tracking Tools in X coordinate, but more noise in Y coordinate.

5.2 Tracking

We evaluated our tracking module by comparing trajectories. The chosen movement for this analysis was from a simple pendulum because its ideal path can easily be described by an arc of a circumference.

OpenMocap and Tracking Tools were configured just like the first experiment, as shown in Table 1. The only difference was the capture speed, in this experiment 50Hz. In addition, alpha and beta values from our tracker were set to 1,0 and 0,8, respectively.

Due to the difficulty of performing the pendulum movement exactly in a plane parallel to the camera's image one, several periods were captured alternating both approaches. Since we consider measures from Tracking Tools our ground truth, we fitted a curve to its results. Figure 10 shows the fitted trajectory and one random period acquired by OpenMoCap.

Two important conclusions can be taken based on this graph. The first one is qualitative, regarding the quality of the motion captured by OpenMoCap. The obtained trajectory is exactly the one expected from a simple pendulum. Slow speed on extremities and faster ones while getting closer to the center. Besides that, the movement is also symmetrical.

The second observation is quantitative and it is related to the existing displacement between detected OpenMoCap centroids and the places that they should appear (points belonging to the fitted curve). Table 3 summarizes this displacement error. Considering that there is friction and the measures could not be obtained simultaneously, those values let us believe that our tracker perfoms like the commercial solution.

5.3 Camera Parameters Estimation

Empirically, it was verified that the implemented algorithm in this work for estimation of camera parameters only works well in situations where the two used cameras are almost in the same plane. In other words, when there is only a small rotation between them. This happens because the cost function needs many points correspondence to be discriminant. Therefore, when using just a small number of them, the obtained solution is a local minimum. Future work includes recording points correspondence for a few seconds from a scene in order to obtain more projections relations. Finally, the number of estimated camera parameters could be increased, aiming to achieve a better representation of a real camera.

5.4 3D Structure

A direct comparison between 3D coordinates from a structure recovered with OpenMoCap and with Tracking Tools is not possible. The main reason for this is that in our approach we do not have a real scale factor while the commercial approach has.

Despite such difficulty, this work proposes a way to evaluate the quality of the reconstructed structure by comparing distances. This is valid because a point can uniquely be defined in a 3D space through the intersection of four spheres, with trilateration [Doukhnitch et al. 2008].

Figure 11 shows different views of the chosen scene containing a seven marker structure. Two markers on the top of the largest box, two over the black cube and three over a "L" shaped object.

The scene was kept static and was captured by the two programs. The only difference in the starting conditions was that the commercial solution used a third camera (Tracking Tools does not allow less that three cameras for 3D capture) and was calibrated. Our software used just two cameras and the seven points correspondence available in the scene. Figure 12 compares qualitatively the structure obtained by both applications.

Tables 4 and 5 show the calculated euclidean distance between points in the structure recovered by Tracking Tools (in meters) and OpenMoCap.

Considering that both structures are similar and the trilateration principles cited earlier, there must be a multiplicative factor that approximates those distances. Table 6 exhibits the normalized ratios. The lower the dispersion of this multiplicative factor, the better is the quality of the reconstructed structure.

The median of these samples is 0,658. Taking into account only the first and the third quartiles of them, 0,524 and 0,702, respectively,

Figure 11: *Scene with 3D Marker Structure*

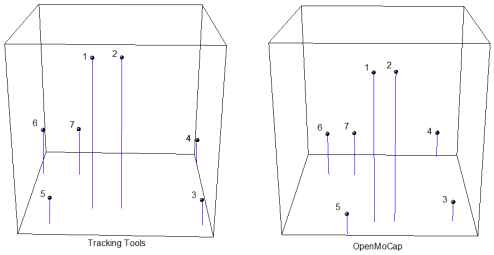


Figure 12: *3D Structure Comparison*

Table 4: *Euclidean Distance Between Points - Tracking Tools*

	1	2	3	4	5	6	7
1		0.075	0.561	0.590	0.498	0.448	0.421
2			0.530	0.557	0.521	0.482	0.440
3				0.313	0.378	0.518	0.435
4					0.510	0.474	0.370
5						0.281	0.283
6							0.106

Table 5: *Euclidean Distance Between Points - OpenMoCap*

	1	2	3	4	5	6	7
1		0,380	2,876	4,162	2,571	3,064	2,883
2			2,708	4,028	2,677	3,192	2,947
3				3,142	1,986	3,297	2,875
4					4,069	2,668	2,176
5						2,685	2,606
6							0,576

Table 6: *Multiplicative Factor Between Distances from OpenMoCap and Tracking Tools*

	1	2	3	4	5	6	7
1		0,504	0,510	0,702	0,514	0,681	0,682
2			0,508	0,719	0,512	0,659	0,668
3				1,000	0,524	0,634	0,658
4					0,794	0,561	0,585
5						0,951	0,916
6							0,540

the maximum difference from the median is 0,134. Therefore the maximum error in this piece of data is around 20%, using Tracking Tools as ground truth. This should be reasonable given our software starting conditions.

5.5 Output and 3D Movement

Even with all limitations of this work, it was possible to capture simple movements from a person with markers in real time at 50Hz. The figure in the first page of this paper shows a key frame of the produced video. The complete video sequence can be downloaded at [OpenMoCap 2009].

5.6 Processing Time

A question that must be answered in this work is if the implemented software architecture is able to record movement in real time. In other words, which the maximum attainable frame rate with the built workflow is. Figure 13 is an area plot that shows the contribution of each step involved in the processing chain to the total processing time for a series of 175 frames. Those frames were specially chosen because of the highest processing peaks observed while recording movement.

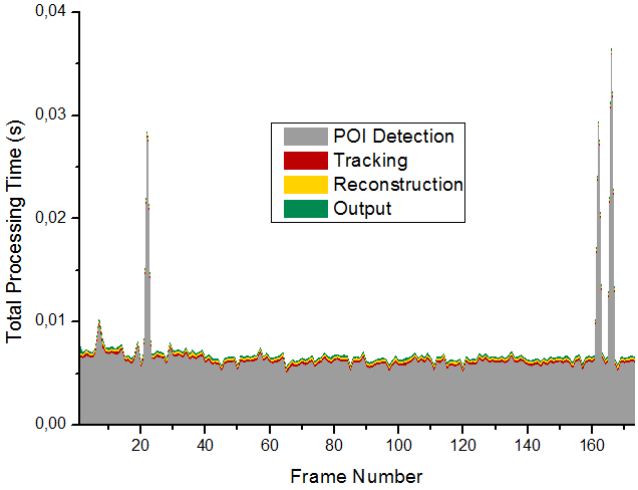


Figure 13: *Area Plot of Total Processing Time in OpenMoCap*

An immediate conclusion of the chart shown in Figure 13 is that almost all processing time is consumed by POI detection. The other steps correspond to minimal portions of the total effort required to record movement. This is why many commercial systems often implement POI detection in hardware.

Another important observation about Figure 13 is regarding processing peaks. The main reason for their occurrence is shared system resources. Processing peaks effectively determine how fast it is possible to record movement, depending on the final use of data. If it is acceptable to lose data from a frame and interpolate trajectories to fix that situation, our software can record movement at 50Hz. On the contrary, if it is not possible to use interpolation, our software is able to record movement at 25Hz.

There is still one contribution of this work that needs to be assessed, the multithreaded architecture. In order to evaluate it, a scene with 20 markers was observed by a variable number of cameras. Figure 14 is a boxplot of the number of cameras used by respective times spent by our software while detecting POIs (most time consuming task performed by our software).

Boxes on the graph represent the intervals between the first and the third quartiles of the samples, i.e. where 50% of obtained measures are located. The larger the number of cameras connected to our software the bigger the boxes are, meaning more data dispersion and less reliability. This is expected and it happens because the operating system shares hardware resources.

Finally, the last interesting conclusion about Figure 14 is related to processing times averages, symbolized by the small squares. Until the fourth camera, only a increase of two milliseconds is perceived on each average. This is much less than the time required to execute POI detection by just one camera, which is approximately six milliseconds. Therefore our multithreaded architecture is validated.

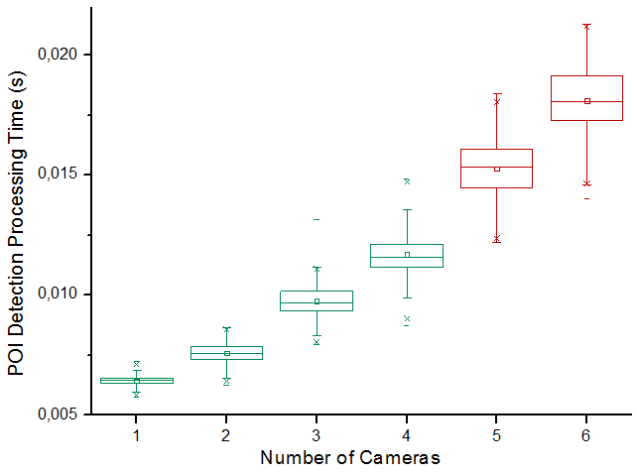


Figure 14: *Boxplot of POI Detection Processing Time by Number of Cameras*

6 Conclusion

In this work, we constructed a real time optical motion capture system from the beginning. The developed software for acquisition and interpretation of data has an open source code. It is a standalone solution, as OpenMoCap includes all the components to process the signal, and its architecture is flexible and extensible as it is possible to modify and add specific modules to improve the workflow. In this way, extensions to allow markless motion capture or to change the camera set can be easily implemented.

This work is part of a project to develop an efficient motion capture system. With this open source system we are making an effort to construct a robust and cheap solution to supply the need for mocap data in character animation in Brazil and in other places where it is not widely used for economical and absence of expertise reasons.

The experimental results show that although it is not a very precise and robust system yet, simple animations can be done with OpenMoCap. Several improvement ideas came up along the development of this software. Unfortunatelly, they haven't been implemented yet due to the lack of time and resources available. As a suggestion to continue this project some of these improvements are listed bellow.

- Define new experiments to better characterize our software.
- Extend OpenMoCap to use more and generic cameras.
- Amend our camera parameters estimation process in order to obtain real scale factor and better precision, allowing facial motion capture.
- Improve our software tracker.
- Perform automatic model initialization.
- Develop POI detection using GPU or distribute processing across a local network with cheap computer nodes.
- Produce output in a hierarchical format, like BVH *Biovision Hierarchy*.

Acknowledgements

This work was supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG).

References

AUTODESK, 2009. Autodesk - 2d and 3d design and engineering software for architecture, manufacturing, and digital entertainment. This is an eletronic document available at: http://www.autodesk.com. Last visited on: May 10, 2009.

BOSLAUGH, S., AND WATTERS, D. P. A. 2008. *Statistics in a nutshell*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

CASTRO, J., MEDINA-CARNICER, R., AND GALISTEO, A. M. 2006. Design and evaluation of a new three-dimensional motion capture system based on video. *Gait and Posture 24*, 1, 126 – 129. http://dx.doi.org/10.1016/j.gaitpost.2005.08.001.

DE LA FRAGA, L., AND VITE SILVA, I. 2008. Direct 3d metric reconstruction from two views using differential evolution. *IEEE Congress on Evolutionary Computation, 2008 (IEEE World Congress on Computational Intelligence)*. (June), 3266–3273. http://dx.doi.org/10.1109/CEC.2008.4631240.

DOUKHNITCH, E., SALAMAH, M., AND OZEN, E. 2008. An efficient approach for trilateration in 3d positioning. *Computer Communications 31*, 17, 4124–4129. http://dx.doi.org/10.1016/j.comcom.2008.08.019.

FIGUEROA, P. J., LEITE, N. J., AND BARROS, R. M. L. 2003. A flexible software for tracking of markers used in human motion analysis. *Computer Methods and Programs in Biomedicine 72*, 2, 155 – 165. http://dx.doi.org/10.1016/S0169-2607(02)00122-0.

FIGUEROA, P. J., 2009. Dvideow. This is an eletronic document available at: http://www.ic.unicamp.br/ pascual/dvideow.html. Last visited on: July 18, 2009.

HARTLEY, R. I., AND STURM, P. 1997. Triangulation. *Computer Vision and Image Understanding 68*, 2, 146–157. http://dx.doi.org/10.1006/cviu.1997.0547.

INITION, 2008. Motion Capture / Tracking from Inition. This is an eletronic document available at: http://www.inition.co.uk/inition/products.php?CatID_=11. Last visited on: November 26, 2008.

INTEL CORPORATION, 2009. Laptop, notebook, desktop, server and embedded processor technology - intel. This is an eletronic document available at: http://www.intel.com. Last visited on: May 26, 2009.

KERNIGHAN, B. W., AND PIKE, R. 1999. *The practice of programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

KITAGAWA, M., AND WINDSOR, B. 2008. *MoCap for Artists: Workflow and Techniques for Motion Capture*. Focal Press, Burlington, MA, USA.

MCCONNELL, S. 2004. *Code Complete, Second Edition*. Microsoft Press, Redmond, WA, USA.

MENACHE, A. 2000. *Understanding Motion Capture for Computer Animation and Video Games*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

MICROSOFT CORPORATION, 2009. Microsoft corporation. This is an eletronic document available at: http://www.microsoft.com. Last visited on: May 26, 2009.

MOESLUND, T. B., HILTON, A., AND KRÜGER, V. 2006. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding 104*, 2, 90–126. http://dx.doi.org/10.1016/j.cviu.2006.08.002.

MOTION ANALYSIS CORPORATION, 2009. The industry leader for 3d passive optical motion capture. This is an eletronic document available at: http://www.motionanalysis.com/. Last visited on: February 5, 2009.

OPENMOCAP, 2009. Sample capture video. This is an eletronic document available at: http://files.getdropbox.com/u/226543/mocap.avi. Last visited on: July 24, 2009.

OPTITRACK, 2008. Optical motion capture and tracking :: Optitrack. This is an eletronic document available at: http://www.naturalpoint.com/optitrack/. Last visited on: December 2, 2008.

PHASESPACE INC., 2008. PhaseSpace Inc — Optical Motion Capture . Este um documento eletrnico disponvel em: http://www.phasespace.com. Last visited on: November 13, 2008.

PINTO, F., BUAES, A., FRANCIO, D., BINOTTO, A., AND SANTOS, P. 2008. Bratrack: a low-cost marker-based optical stereo tracking system. In *SIGGRAPH '08: ACM SIGGRAPH 2008 posters*, ACM, New York, NY, USA, 1–1.

RASKAR, R., NII, H., DEDECKER, B., HASHIMOTO, Y., SUMMET, J., MOORE, D., ZHAO, Y., WESTHUES, J., DIETZ, P., BARNWELL, J., NAYAR, S., INAMI, M., BEKAERT, P., NOLAND, M., BRANZOI, V., AND BRUNS, E. 2007. Prakash: lighting aware motion capture using photosensing markers and multiplexed illuminators. *ACM Transactions on Graphics 26*, 3, 36. http://doi.acm.org/10.1145/1276377.1276422.

STAGE, 2009. Organic motion: Solutions. This is an eletronic document available at: http://www.organicmotion.com/solutions/stage. Last visited on: February 2, 2009.

TROLLTECH, 2009. Qt Software - Code Less, Create More, Deploy Everywhere. This is an eletronic document available at: http://trolltech.com. Last visited on: January 4, 2009.

UCHINOUMI, M., TAN, J. K., AND ISHIKAWA, S. 2004. A simple-structured real-time motion capture system employing silhouette images. In *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics*, 3094–3098. http://dx.doi.org/10.1109/ICSMC.2004.1400814.

UMBAUGH, S. E. 2005. *Computer Imaging: Digital Image Analysis and Processing.* CRC Press, Boca Raton, FL, USA.

VICON MOTION SYSTEMS, 2008. Motion Capture Systems from Vicon. This is an eletronic document available at: http://www.vicon.com. Last visited on: November 13, 2008.

WATSON, T., 2009. videoInput Library. This is an eletronic document available at: http://muonics.net/school/spring05/videoInput/. Last visited on: January 4, 2009.

WILLOW GARAGE, 2009. OpenCV - Wiki. This is an eletronic document available at: http://pr.willowgarage.com/wiki/OpenCV. Last visited on: January 4, 2009.

YOO, J.-C., AND KIM, Y.-S. 2003. Alpha-beta-tracking index ([alpha]-[beta]-[lambda]) tracking filter. *Signal Processing 83*, 1, 169 – 180. http://dx.doi.org/10.1016/S0165-1684(02)00388-2.

Procedural Generation of Hair

Vinícius Jurinic Cassol
Fernando Pinho Marson
Soraia Raupp Musse

Graduate Programme in Computer Science
PUCRS - Av. Ipiranga, 6681, Porto Alegre, RS, Brazil

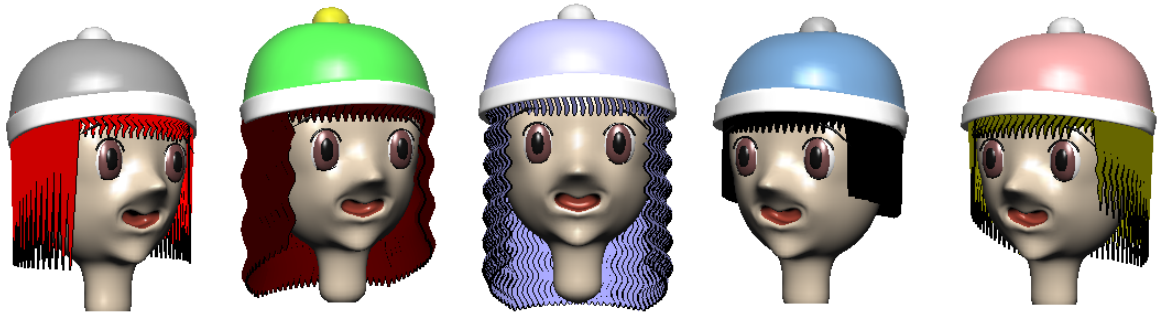


Figure 1: *Different types of hair generated through variation of parameters.*

Abstract

This paper presents a hair generation technique for cartoon and Anime characters. The main goal of this work is to provide a procedural generation technique, robust enough in order to create different types of hairs. We propose a parametric model that can produce diversity of obtained results through parameters variation. The model is organized in three steps: firstly, the strands generation, followed by positioning them in the head of the characters. Finally, the third step is responsible for providing cartoon rendering, including common characteristics in cartoon shading, as the black edges and solid colors. Visual inspection of obtained results indicate that generated hairs are coherent with expected in terms of visual aspects.

Keywords:: Hair Generation, Cartoon Hair, Procedural Modeling.

Author's Contact:

vinicius.cassol@acad.pucrs.br
{fernando.marson, soraia.musse}@pucrs.br

1 Introduction

Hair modeling and rendering in cartoon characters can be applied in animation movies and computer games. Hair effects can be used to distinguish among different cartoon characters, especially in Anime [Shin et al. 2006] characters. When characters faces are similar, they can be distinguished as a function of their hair geometry and color. Indeed, hair can be considered as an identifier or a factor who compose the characters identity. We can observe some of these factors in human beings, e.g. hair color, length and type: straight or curly. Such parameters compose the individual identity and, in our computational model are described in hair generation and rendering process.

In terms of appearance, computer graphics techniques have been used to provide different types of hair representation. Indeed, such techniques can provide photorealistic and non-photorealistic rendering. The photorealistic rendering techniques represent characteristics from the real world, i.e. they imitate real effects. On the other hand, the non-photorealistic rendering techniques are used to represent other visual languages, e.g. cartoon appearance.

Our work presents a procedural hair generation technique focused on cartoon and Anime characters. In order to procedural generate hairs, we propose a parametric model which can produce visually different results.

1.1 Procedural Generation

Procedural techniques can be defined as segments of code or algorithms that specify some model feature or a computer generated effect [Ebert et al. 2002]. Such techniques have been used since the beginning of Computer Graphics, in order to generate geometrical models, create textures and animation of objects and characters.

The main characteristic of procedural methods is the abstraction. Instead storing all details of a particular shape or animation, the main concept of procedural technique aims to provide details in-game (or during the simulation). Consequently, the algorithms should be robust enough to allow coherent re-creation, for instance each time a specific model should be re-created, e.g. a tree, it should be made through a computational procedure, and not loading files. Yet, procedural methods should be able to create different trees, while keeping the expected global picture of a tree.

The description of a robust and coherent procedural model allows the generation of shapes/animations, without required extra files. Indeed, procedural methods should generate different characteristics of obtained results. That is why it uses parameters which are associated with generated characteristics. Consequently, generated distinct objects or animations can share common features. For instance, one can mention an algorithm to codify the general concept of chair [Morkel and Bangay 2006]. There are many combinations of different features that can be found in chairs: height, material, type of seat and backrest, among others. However, instead of storing pre-computed models, the procedural methods aims to generate them. A strong point is the flexibility to generate diversity into obtained results. The model proposed in this paper aims to generate cartoon hair models and appearance through a procedural method.

The paper is organized as follows: in Section 2, we present an overview of published work in hair modeling area, while in Section 3, we present our procedural generation model. The prototype and obtained results are presented and discussed in Section 4, and finally, in Section 5, we conclude the paper with the possibilities for future work.

2 Related Work

Procedural methods of hair generation can produce characters for different types of applications. According to the application, the required level of realism can vary from photorealistic to non-photorealistic images. Both of representations can be used in different applications for computer games and animation movies.

Indeed, realistic hair rendering is still considered a challenge, since each hair, in each strand, should be individually modeled. Please, notice that a common hair volume has about 100.000 strands [Ward 2005]. The individual representation of any strand is necessary when we need to represent the perfection of the hair shape and motion to look like a human hair. In last years, many methods have been developed with the main goal of producing realistic hair models [Volino and Magnenat-Thalmann 2004], [Bertails et al. 2006], [Ward et al. 2007b].

Focused on photorealistic rendering, Magnenat-Thalmann et al. [Magnenat-Thalmann et al. 2000] presents a survey paper in hair generation methods where authors analyze the main presented challenges. Two are important requirements: firstly, the shape modeling considering the perfection of the human hair, and secondly, the computational time of proposed methods. Recently, another survey was produced by Ward [Ward et al. 2007a] and again presents the state-of-the-art in hair generation in terms of style generation, simulation and quality of rendering.

On the other hand, non-photorealistic rendering process do not represent the object or character as it is in real world. This kind of representation provides stylized representation, usually applicable in cartoon characters. Due to the visual aspect of cartoons, it is not necessary to model and generate each hair strand. Considering this perspective, it is possible to model specific shapes and rendering often presented in cartoons characters. One example of hair modeling and cartoon rendering is presented by Shin [Shin et al. 2006], who develop a model based on particle system, using GPU and billboards in order to produce rendering highlights. To any generated strand, the silhouette is highlighted as expected in cartoons.

Sugisaki [Sugisaki et al. 2004] presents a hair animation technique based on Physical Simulation. The process begins with the extraction of hair motion from a existing cel cartoon animation. With this data, hair motion database is built, which is physics-based. In this method, an human intervention is required in order to produce visually good results. Yet, this technique provides hair motion through deformation functions applied in strands angles. Another work from the same authors [Sugisaki et al. 2006] describes a new model to animate hair motion where existing cel are used. The sequences are extracted from a database and parameters are applied in order to control the hair stiffness. Phisically-based equations are used to produce hair shape and motion.

Noble [Noble and Tang 2004] shows an approach to generate and animate cartoons hair. Such technique uses NURBS surfaces [Piegl and Tiller 1997] to compose a geometric mesh which produces different hair shapes and motion. An animated NURBS volume define a primary shape and motion. By this way, along the surface are generated clumps of hair geometry. With the technique developed by Noble, artists have a flexible tool to provide hair generation and animation in real-time.

Yet, considering non-photorealistic rendering techniques, different hair characteristics can be simulated. Moreover, it is defined by the type of character where we will apply the hair. We can use a cartoon rendering to produce these characteristics in a draw cel character, e.g. Decaudin [Decaudin 1996] can be considered as a cartoon rendering forerunner. In such technique the author consider some typical cartoon features as to use uniform colors in the draw fill and always use the same edge black thickness in the character silhouette. In this way, other points can be considered in the rendering process like shadows and lights.

3 Procedural Hair Generation

In this section, we describe the method for hair generation, the proposed steps as well as the overall architecture. The procedural generation of hair consists in three main steps:

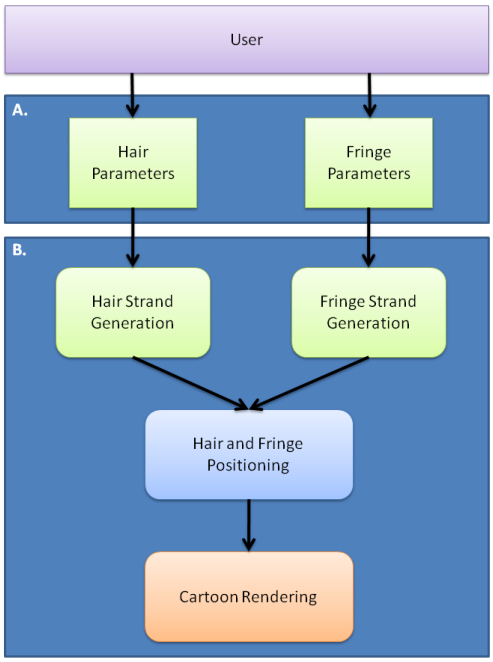


Figure 2: *The pipeline of hair generation process.*

- Parametrization and generation of strands: The user must set the desired features for the hair and the fringe. These settings are made through a graphical interface. This phase ends with the generation of each strand based on parameters selection.
- Strands positioning in the head: After the generation, the strands and fringe must be correctly positioned and aligned with the character head.
- Cartoon Rendering: In this step, we apply cartoon rendering characteristics in the generated geometry.

The pipeline of proposed model is presented in the Figure 2. Box A represents the user inputs that are used to define the hair features and box B shows the main steps involved in the hair generation process. In the next Section, the steps will be more detailed.

3.1 Strands Generation

As it was presented before, our technique is based on procedural generation. By this way, the model need some parameters definition to produce different hair models. To allow the user to easy and fast parameters definition and variation, we developed a prototype wit and a graphic interface. In this prototype the parameters can be defined or changed and the user can see and validate the results in real time. This prototype will be detailed in the Section 4.

Our model can generate three different styles of hair: straight, wavy and curly. The modeling of each wick of hair is given initially by setting a guideline. This guideline will be used as a basis to generate the final shape of strand. The amount of vertex in the guideline allows to vary the obtained result in a wavy or curly hair. The other parameters are the desired style of hair, the width, the initial thickness and the length of each match. All these parameters are defined in the interface.

Every vertex in the guideline is used to create other four points: two points to specify the width in the x axis and two points for the thickness in the z axis. In Figure 3, we can see the guideline vertex (red) and the four auxiliaries points (blue). The distance between the guideline points (in y axis) is given by a step times the length. This step is obtained by

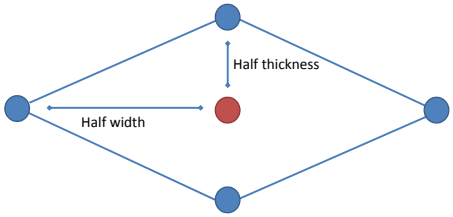


Figure 3: *Generation of auxiliary points (blue points) based on a guideline vertex (red point).*

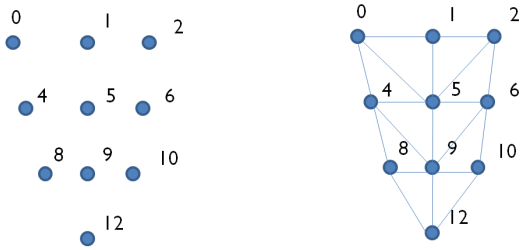


Figure 4: *Vertex distribution in the strand (left) and the strand triangularization (right).*

$$step = (1/number\ of\ vertex). \quad (1)$$

Based on the number of vertex in the guideline, we can calculate how many vertex and faces will compose the surface strand's:

$$surface\ vertex = (number\ of\ vertex\ .\ 4) + 1, \quad (2)$$

$$surface\ faces = ((number\ of\ vertex\ -\ 1).8) + 4. \quad (3)$$

After calculating the number of surfaces vertex, they are organized in a triangle shape where the larger size is given by the user and the others are linearly distributed, as illustrated in Figure 4 (left). Based on the vertex distribution, we can do the strand triangularization, generating the mesh illustrated in the Figure 4 (right). In this example, the vertex number 3, 7 and 11 located at the back face of strand generate other triangles; e.g. 0, 1, 3 and 4, 5, 7.

In the straight and wavy hair style, the entire width and thickness defined by the user, are considered from the beginning of the hair (from the top of head). For each point at the guideline from the beginning to the end of the hair, the width is reduced, causing an effect into the strand as seen in Figure 5. If the process is interrupted by the user before the end, we can create an additional effect: the strand seems to be cut. The distinction between straight and wavy hair is obtained applying a variation wavy factor, positive and negative alternately, in the x coordinate, that is given by

$$wavy\ factor = (1 - step_n). \quad (4)$$

A peculiar feature of the wavy hair is the location where the wavy effect should start. By this way, we can define the point of the strand where the frizz should begin. With this possibility is not necessary to produce always the same strand wavy.

For the curly hair style, the process is a little bit different. Firstly, it is defined the angle(θ) that the curly hair should have to provide the curly effect, e.g. 20 degrees. The y value is calculate according Equation 1, but the value of x and z are given, respectively by

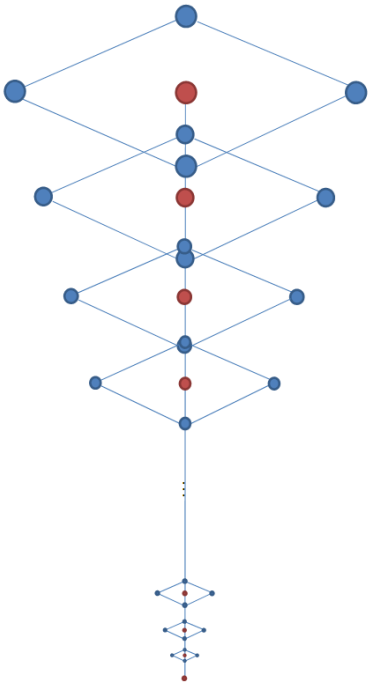


Figure 5: *Generation of a straight strand considering the position of the auxiliary points (blue points) with the points of the guideline (red points).*

$$x = sin(\theta) \times (\frac{width}{4}), \quad (5)$$

$$z = cos(\theta) \times (\frac{width}{4}). \quad (6)$$

After each step, it is added 20 degrees into the angle. Different values can generate distinct results. The resulting points from this process are used to building a polygonal mesh. The Figure 6 shows the three styles of strands generated by our technique.

The next Section presents the next step in the pipeline to place artificial hair in characters.

3.2 Positioning of Strands in the Head

After the generation of each strand, it is necessary to place them correctly positioned and aligned with the head of the character. This

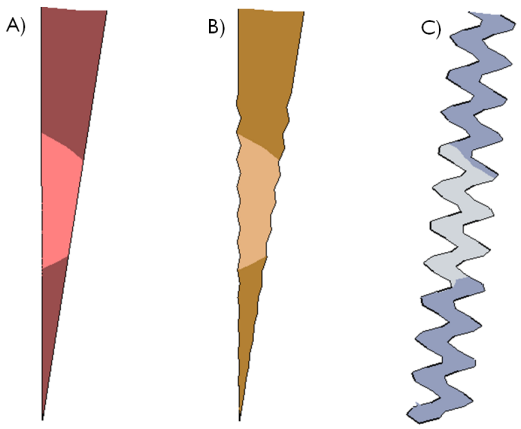


Figure 6: *Styles of hair generated with our technique: a)straight strand, b)wavy strand with the definition of a point to start the frizz, c)curly hair.*

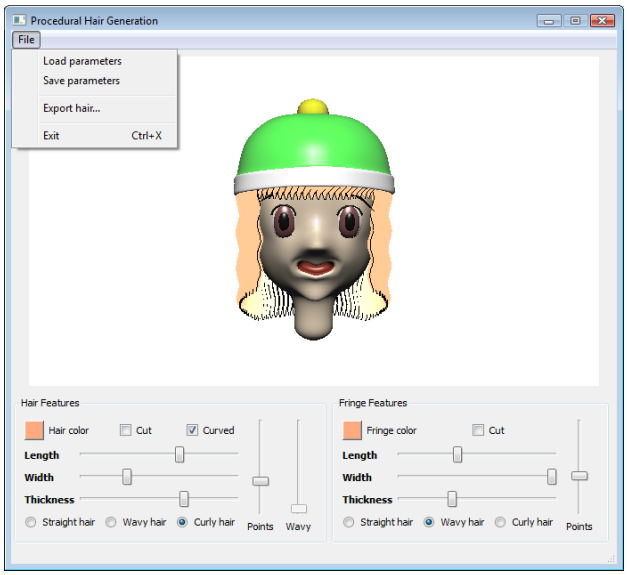


Figure 7: *The interface of the prototype.*

process is composed by four steps. At first, the initial point of the guideline is translated to a specific y value in the head were strand should start. After, it is necessary to calculate the angle between the normal vector of the strand ($\vec{u}$) and the normal vector of each triangle that will receive the strand ($\vec{v}$). The strands are generated at position and orientation.

3.3 Cartoon Rendering in This Work

With a cartoon rendering technique, it is possible to apply a cartoon style in objects or characters, like an animated movie or represent comics draws. There are some peculiar features about cartoon. In our results, we can identify some features of this technique as used in the Decaudin model [Decaudin 1996]:

- Uniform colors in the draw fill: In the traditional cartoons draws we can see the use of solid colors in the draw fill, as we used to produce our hair. Otherwise, in a sophisticated rendering process, it is possible to use different levels of color intensity. This levels can provide some different results in the character rendering.
- Use the same edge black thickness to define the character silhouette: This is a important point to allow to identify different characters in the same scene. By this way, we can avoid the overlap among the characters or objects rendered. This point is verified in the strands generation process. In this process, we identify and stored the external edge to allow to render this edges in black color.

4 Prototype and Results

Based on the proposed model, a prototype was developed to validate the technique and evaluate obtained results. It was coded using C++. The QT framework was used to create the interface. A head was modeled in order to test the correct positioning of strands. Figure 7 presents the interface of the prototype.

The needed parameters are separated into two different sets. First one allows to control just the fringe (frontal hair). The other set, controls the rest of hair. To the both sets, the user can choose the color, the length, the width, the thickness of hair or the fringe. The style of hair (straight, wavy or curly) also can be applied individually. For curly or wavy hair, it is possible to modify the number of sampled points. This feature allows to create more defined strands. Just for the wavy hair style, is feasible specify where the frizz point should start, as we can see in the Figure 8. Also it is possible to create a cut effect, i.e. the strand seems to be cut. Yet, in the Figure 7, we can see the last parameter: the curved hair. With this parameter it is possible creates short strands on the sides and longer at

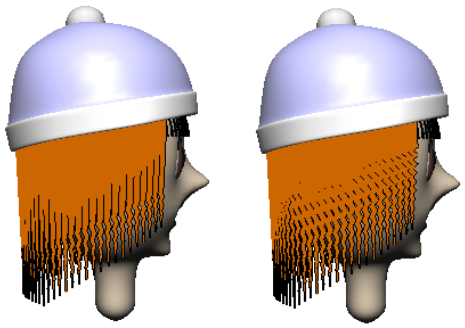


Figure 8: *Different results obtained by the variation of the wavy start point.*

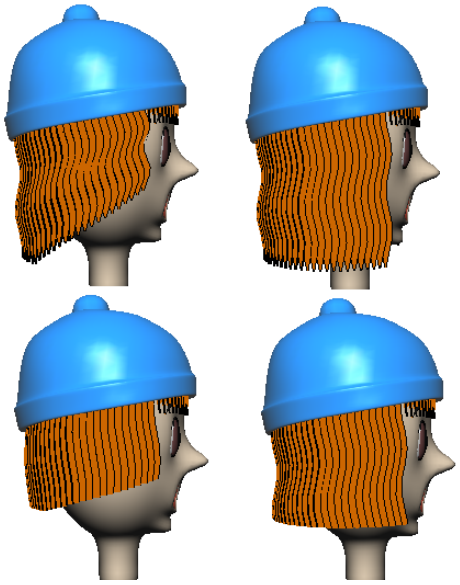


Figure 9: *Different results by using curved and cut parameters.*

the nape of neck. The Figure 9 shows results obtained by applying curved and cut parameters.

The prototype allows to save the used parameters into a configuration file in order to use them in a next time. Also it is possible to export the generated polygonal mesh to the OBJ[1] format, thus allowing its use in games and other graphical applications.

With the parameters variation, we can present different results in the Figure 10. An interesting feature of our model is the real time hair generation. This is possible because our model has low computer cost. The time for hair generation is basically the time for interface manipulation, since the model is generated in interactive frame-rates.

5 Conclusion and Future Work

This paper presents a procedural technique to generate different kinds of hair that can be applied to cartoon characters. The main feature of our method is the capability to simulate, analyze and validate visually the results in real-time. This is provided by a parametric model that can be changed through the specification of parameters provided by the user. Another important feature is the simplicity of our method. It can be easily reproduced and modified to add new hair styles.

As a future work we want to improve our model by handling other parameters generating a varied range of results. Currently, the strands are not generated from the top of hair. This feature will

[1] A geometry definition file format first developed by Wavefront Technologies

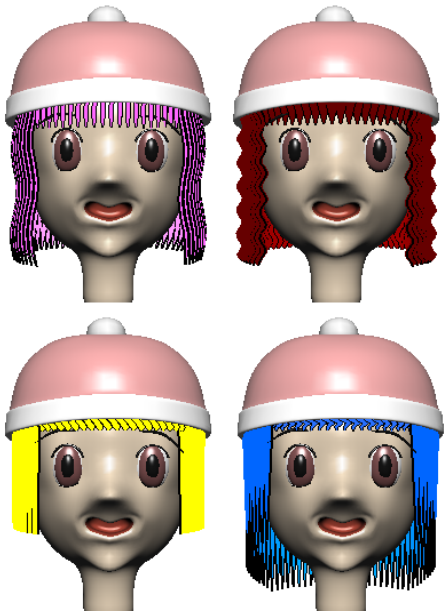


Figure 10: *Results produced by our model.*

be implemented soon. Other question is the possibility of using different head models. To do this, it is necessary to create a tool that allow to specify the faces where the hair will grow up. In this sense, a metalanguage can be used to import and export geometry.

References

BERTAILS, F., AUDOLY, B., CANI, M.-P., QUERLEUX, B., LEROY, F., AND LÉVÊQUE, J.-L. 2006. Super-helices for predicting the dynamics of natural hair. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM, New York, NY, USA, 1180–1187.

DECAUDIN, P. 1996. Cartoon looking rendering of 3D scenes. Research Report 2919, INRIA, June.

EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

MAGNENAT-THALMANN, N., HADAP, S., AND KALRA, P. 2000. State of the art in hair simulation. In *International Workshop on Human Modeling and Animation*, Computer Graphics Society, 3–9.

MORKEL, C., AND BANGAY, S. 2006. Procedural modeling facilities for hierarchical object generation. In *Afrigaph '06: Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, ACM, New York, NY, USA, 145–154.

NOBLE, P., AND TANG, W. 2004. Modelling and animating cartoon hair with nurbs surfaces. In *CGI '04: Proceedings of the Computer Graphics International*, IEEE Computer Society, Washington, DC, USA, 60–67.

PIEGL, L., AND TILLER, W. 1997. *The NURBS book (2nd ed.)*. Springer-Verlag New York, Inc., New York, NY, USA.

SHIN, J., HALLER, M., AND MUKUNDAN, R. 2006. A stylized cartoon hair renderer. In *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, ACM, New York, NY, USA, 64.

SUGISAKI, E., YU, Y., ANJYO, K., AND MORISHIMA, S. 2004. Cartoon hair animation based on physical simulation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Posters*, ACM, New York, NY, USA, 27.

SUGISAKI, E., KAZAMA, Y., MORISHIMA, S., TANAKA, N., AND SATO, A. 2006. Anime hair motion design from animation database. In *CyberGames '06: Proceedings of the 2006 international conference on Game research and development*, Murdoch University, Murdoch University, Australia, Australia, 33–40.

VOLINO, P., AND MAGNENAT-THALMANN, N. 2004. Animating complex hairstyles in real-time. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 41–48.

WARD, K., BERTAILS, F., KIM, T.-Y., MARSCHNER, S. R., AND CANI, M.-P. 2007. A survey on hair modeling: Styling, simulation, and rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 2, 213–234. Member-Lin, Ming C.

WARD, K., GALOPPO, N., AND LIN, M. 2007. Interactive virtual hair salon. *Presence: Teleoper. Virtual Environ. 16*, 3, 237–251.

WARD, K. A. 2005. *Modeling hair using levels of detail*. PhD thesis, Chapel Hill, NC, USA. Adviser-Lin, Ming C.

Prototyping games for training and education in Second Life: Time2Play and TREG

K. Vega A. Pereira G. Carvalho A. Raposo H. Fuks

Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Department of Informatics, Brazil

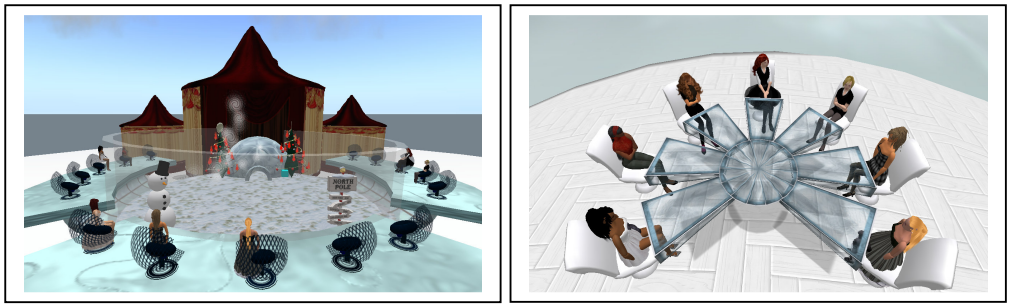


Figure 1: Educational Games in Second Life: Time2Play (left) and TREG (right).

Abstract

The purpose of this paper is to report the experience in prototyping 2 games for education and training in Second Life, Time2Play and TREG. Starting from a prototyping process, it was adapted for getting better results in the development of the games. Based on our experience, Second Life provides a sound platform for the step-by-step prototyping evolution.

Keywords: prototyping, prototyping process, game development, Second Life.

Authors' contact:
{kvega, andreiapereira, guga, abraposo, hugo} @inf.puc-rio.br

1. Introduction

Virtual environments have been successfully used in several contexts like educational, social, gaming and commercial. A virtual world is an environment simulated by a computer provided for some specific goals. Second Life is a 3D virtual world opened to the public since 2003 where users have the possibility of creating their part of that world [Linden Labs 2009]. Gartner reports that 80% of the active users will have a "Second Life" by the end of the 2011 [Pettey 2007]. This offers new challenges and opportunities for educators.

Educational simulations seem to be the new paradigm of knowledge transfer and social training. Moreover, current studies indicate that students are becoming more pragmatic, visual and computer-savvy [Aldrich 2005].

The Time2Play and TREG games were created in Second Life (SL) using its building blocks and scripting possibilities. In this paper we will discuss our experience in prototyping the Time2Play and TREG games using Second Life as the development platform.

This work is organized in eight sections. Section 2 introduces Second Life and its possibilities for creating educational games. Section 3 presents some related work of serious games created in Second Life. Section 4 shows overviews of Time2Play and TREG: a storytelling and training game designed for Second Life. Section 5 discusses the prototype process followed by its development. Section 6 and 7 describe the prototyping methods used and our experience prototyping Time2Play and TREG. Section 8 concludes the work.

2. Second Life as a development platform

Virtual worlds are an interactive multi-user environments simulated by a computer. They are also called Massive-Multiplayer Online Role-Playing Games (MMORPGs) and have these common features [Book 2009]:

1. Shared Space: Many users are simultaneously sharing the same world;
2. Graphical User Interface: They have visual environments from a 2D style to a more immersive 3D world;
3. Immediacy: Interaction with the world takes place in real time;
4. Interactivity: Virtual worlds use to allow users to make changes to it like alter, develop, build, or submit customized content;

5. Persistence: the environment continuous existing and being developed internally even if there are no users interacting in it; and

6. Socialization/Community: the world allows and encourages the formation of in-world social groups like teams, guilds, clubs, cliques, housemates, neighborhoods, etc.

The online 3D virtual world Second Life was launched by Linden Lab in 2003 [2009]. Users, represented by avatars, do real life activities - they interact, play, build and do business purchasing and selling the virtual currency Linden Dollars - building most of the existing content in Second Life. Their raw materials are prims that are the basic building 3D geometric blocks for creating objects. These objects needs scripts for "getting alive" in order to interact with other objects, avatars and Non-Player Characters. Scripting is done using the environment's event oriented language: Linden Scripting Language (LSL) which is familiar for C programmers.

Below we present some of SL building and scripting features that are relevant for this work.

- Appearance and motion. Avatars can change the body and clothes using configured ones or they can configure their own body and clothes using the appearance editor. The face and body motion is made by animations and gestures.
- Textures. It is possible to use textures applied to objects and clothes, obtained out-world, shared among avatars or as in-world snapshots.
- Freebies. There are elements that are shared with the SL community.
- Permissions. These are configured for a collaborative development process. The SL land owner or administrator sets permissions for a group or an avatar to build in their lands. An avatar gives permissions to specific avatars for moving, copying or editing their in-world objects. And an object or script owner gives the permission to the next owner to copy, modify or resell it.
- Teleporting. An avatar is teleported to specific places by other avatars invitations, Landmarks or scripts.
- Voice chat. Avatars could communicate and coordinate enabling this feature.
- Machinima. A filming technique which was used for making videos with avatars, NPCs and objects in virtual worlds.

All that features can be used for the creation of these games in Second Life. But, it was necessary to consider some Second Life constrains in order to develop them.

- The hardware requirements must be the minimal considered by Linden Labs [2009]. It is recommended having a powerful video card. This will facilitate the load in the Client of textures and sculpted objects like the NPCs and some delaying in scripts like listen running scripts or rezzing objects.
- Second Life allows permissions configuration for building and collaborative editing. However there are some constrains for developing in-world like there is no version control system, the LSL editor in-world doesn't have a debugger and compiler and Second Life doesn't have a database repository in-world.

3. Related Work

Oblinger [2004] calls this new students' generation the Net Generation (NetGen). NetGeners tend to be experiential learners, community-oriented and their learning preferences include teamwork and technology use. Games and simulations are a potential learning environment to create educational engagement for them. Virtual worlds can be an effective environment for educational games [Cunha at el. 2008]. Second Life gives the possibility to create different educational content like classes, discussion panels and games [Klunge and Riley].

Kidz Connect is a program created by ZoomLab that connects young people in different countries via media art, performance and collaborative creation in virtual worlds like Second Life (Figure 2). Guided by artists and educators from theatre and digital arts, students learned skills like playback theatre, digital storytelling, and 3D modeling. Students from two different countries write, create and perform a live show. In Second Life, the students met and collaborated to build a hybrid virtual city combining aspects of both countries and in that common space, they created a performance that occurred both live and online simultaneously [Kidz Connect 2009].

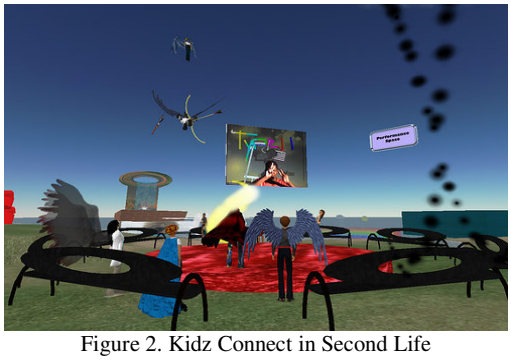


Figure 2. Kidz Connect in Second Life

Training by playing is a great way to improve people skills and have fun at the same time. Nowadays, companies and universities are researching and investing in Second Life for training in several fields.

Idaho State University created a large scale training game to simulate top exercises in health care and emergency preparedness [Ramloll et al. 2006]. Figure 3 shows a hospital scenario where a pandemic flu training exercise was simulated [Idaho 2009].

Our research found games developed in Second Life for computer science. The Ohio University developed two multi-player games in Second Life for software engineering education [Ye 2007]. The first one based on Groupthink Software Specification Exercise developed at the M.I.T. [Ernst 2006a]. Groupthink aims to teach in a game show style how to write effective specifications. This game is available to use in-world [Ernst 2006b]. The other game was developed at the UC Irvine based on the SimSE game where students manage a simulated software project [Navarro et al. 2007].

Figure 3. Hospital Scenario in Play2Train

IBM developed a series of games called Open Encounters of z Virtual Kind for challenging skills in IBM technologies and Open Source like IBM System z mainframe, Service Oriented Architecture (SOA), the Cell/B.E. processor, Grid computing, Linux, Java, and a host of other technologies [IBM 2009].

4. Overview of Time2Play and TREG

In this section we give an overview of the two games that were prototyped using Second Life as the development platform. This choice was supported because of the immersive and collaborative features inherent to Second Life that are needed for the games Time2Play and TREG.

4.1 Time2Play

Time2Play enables the creation of stories and the re-creation of well-known stories in a 3D environment. Each learner has an avatar for enacting her part in the story, role playing that way with other learns, allowing the socialization of knowledge.

After logging in Second Life, the avatar is teleported to the auditorium shown in Figure 4. It is a theatre that enables avatars to watch and participate in the story being enacted. Sitting avatars may at any moment grab clothes or objects, entering this way into the performance of the play, moving from lurkers to players bringing new life into the stage.

The backstage auditorium comprises 3 rooms loaded with different features and functions for supporting learners' performances: main room, dressing room and animations room. In the Main Room, there is a panel that provides scenarios based on themes such as a beach, a forest, a snowy park, among others. These scenarios play environmental sounds based on the theme they represent. There are also two panels that provide objects and special effects to complement the scenarios, offering other possibilities for the players. The Dressing Room has panels that provide clothes, hair, makeup and accessories that allow the avatar's characterization according to the stories that will be performed. There is also the Characters panel that enables learners to transform their avatars into non-human characters by acquiring an alternative form such as a robot, cat, witch, etc. Finally, there is the Animations Room that provides different animations that engage the avatar in a sequence of movements such as swimming, running, or dancing, that could be activated in their performances.

Figure 4. Time2Play auditorium.

Learners divided their work into two activities: story creation and story enactment. They coordinated these activities using the voice chat featured available within SL. Firstly, they came up with the story idea, and then, they selected and modified the scenarios using the panels and, finally, chose their characters' appearances and costumes. During story enactment, each learner played a character in the story, that had no rules or actions pre-defined by the game.

4.2 TREG

The Training in Requirements Engineering Game (TREG) is a 3D online game which aims to teach requirements engineering techniques using simulations based on collaboration. The expected audience is a stakeholder involved in requirements elicitation (students, customers, users or software suppliers) that wants to be trained in this topic. In this phase of the project, we are focusing on training in the workshop

technique, a collaborative way for gathering and analyzing requirements.

There were used the building capabilities in Second Life to create the main building, metaphor rooms, simulation rooms, NPCs and heads-up-display (HUD). Scripts were used to move the avatars by teleporting between the metaphor rooms, to program the HUD with the game states, to handle the NPCs' functionalities in the simulation rooms, to communicate objects, and to play videos. The main building includes a reception area, teleporters that transports the trainees to the metaphor rooms and a NPC, a guide in-world. Figure 5 shows Miss Workshop, a specific NPC implemented for guiding the trainee in the playing of the game.

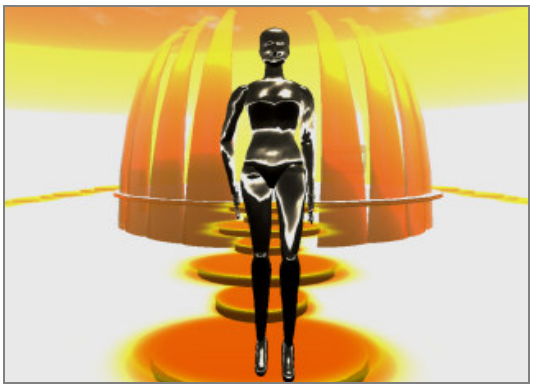


Figure 5. Miss Workshop, the NPC implemented for guiding

The HUD created for the TREG game controls the scores of the game: main score, investment, time, mission and technique. One can score more points, loose investment and time and move to a next level. This information is registered for following the trainee's participation in the game.

The NPC is an object that resembles an avatar but it is controlled by scripts. They are normally used in fictional simulations or role-playing games for interacting with avatars in pre-articulated situations where human beings are dispensable or not available [Bartle 2004]. TREG makes use of NPCs for representing simulations, guiding and shooting machinima videos. The "Making Workshops" Room in Figure 6 shows a workshop session populated by NPCs.

Figure 6. The Cooking Metaphor and the "Making Workshops" Room

Machinima is a technique that relies on the use of 3D game engines to generate a recorded performance in virtual worlds and uses in-world film techniques where characters and events can be controlled by humans, scripts or artificial intelligence [Nitsche, 2005]. Machinima was used to film some problematic workshop situations. There, the trainee will watch a machinima showing the consequences of choosing the "Right People", for example, learning this way a new workshop technique based on a collaboration pattern, if it applies.

5. Prototyping Process

An incremental and iterative prototyping process was used for the creation of these games. Second Life facilitates the prototyping process as it shows the elements in-world as-is which makes it possible to have an early vision of the game and figure out the necessity of additional features and functions.

Figure 7 shows the Prototyping Process of the book Effective Prototyping for Software Makers [Arnowitz et al. 2007]. An iteration consists of the four phases of this process. They will iterate until the software validate all the requirements established.

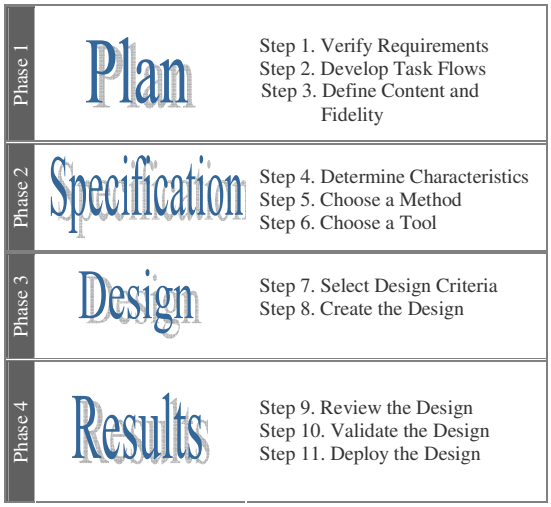


Figure 7. The Effective Prototyping Process [Arnowitz et al. 2007].

The following steps describe how the process had been customized for implemented the game in Second Life.

Step 1. Verify Requirements

The software requirements were discovered from assumptions taking into account that the audience to each game is orientated and their goals. They were gathered, inventoried and prioritized.

Step 2. Develop Task Flows

The task flows depicts the steps that the learner has to follow to complete an activity. In this step, it was defined these tasks and the narration of the scenarios.

A list of the tasks was used for identified the learners' actions in Time2Play. In TREG, branching Stories were used as the simulation games genre for mapping the scenarios of the gameplay. The Branching Stories Graph gives an overview of the game and guides the interaction of the different scenarios given to the trainee. A scenario template was used for getting a fine-detailed specification of the gameplay.

Step 3. Define Content and Fidelity

It was decided to use Second Life as the main prototyping tool for a high fidelity representation. The level of detailed depended on the iteration milestone, from a high level visualization perspective of the game to a coded behavior of the game.

Step 4. Determine Characteristics

There were identified the characteristics proposed by Arnowitz [2007] to determine the prototype method to applied in each iteration. These characteristics are audience, stage, speed, longevity, expression, style and medium. The next sections shows the characteristics found in the games. As it was a prototype driven development, a prototype was created in all the iterations.

Step 5. Choose a Method

A prototyping method was chosen depending on the iteration characteristics. Arnowitz [2007] proposes the following prototyping methods: card sorting, wireframe, storyboard, paper, digital, blank model, video, Wizard-of-Oz and code. Sections 6.1 and 7.1 expose the chosen methods in the games.

Step 6. Choose a Tool

Although it was decided since the beginning of the projects that Second Life would be the prototyping tool, it was necessary the use of other tools for clarifying concepts or specifying functionalities such as office or CASE ones.

Step 7. Select Design Criteria

Despite Second Life promotes a freedom content creation, there are some building and scripting constrains that must be taken into account. In addition, the prototype characteristics had to be considered for each iteration. A design guideline was used to minimize user's memory load. Thus, the game objects don't get overlooked or trivialized. In Time2Play, panels have the same measures and navigation functionality. In TREG, a specific NPC for guiding the trainee is located in the rooms.

Step 8. Create the Design

This step applied all the design rationale into the prototype. In the games was important to prioritize the elements to develop. In TREG a top-down strategy was applied whereas Time2Play applied the opposite strategy, bottom-up. In addition, Time2Play, as it was a collaborative development, required to preset the environment.

Step 9. Review the Design

The prototypes of the games were reviewed by an internal audience of researchers involved in the projects. A teacher was required as the subject matter expert in Time2Play and an expert in software development process for TREG.

Step 10. Validate the Design

After the design revision, it became necessary to validate the prototypes with external stakeholders. Time2Play used usability tests for validating the user experience and ensuring its usability. Section 5.2 specifies the validation steps.

Step 11. Deploy the Design

The development environments for each project were specific sandboxes located at the land in Second Life where the prototypes were developed. Then, these prototypes were deployed to the production environment taking into account the characteristics and requirements of each game.

6. Prototyping Time2Play

Time2Play prototype characteristics were pre-defined for choosing the prototyping method for each iteration. These characteristics were based on the Effective Prototyping Process [Arnowitz et al. 2007].

- Audience: Internal when the prototype was showed to the internal team and external when the audience was a subject matter expert or the children.
- Speed: As it was a reusable strategy, the speed of prototyping was rapid. No more than 2 weeks each prototype.
- Longevity: Long. The prototype was persistent and all its elements continued existing from the beginning of the prototyping process.
- Expression: Conceptual in a card sorting prototype for determining the concept of the project. And it was experiential using Second Life in all the other prototypes.
- Style: It was interactive as the audience could actively explore it in Second Life. Card sorting prototype used a narrative style for getting the conceptual design.
- Medium: It was digital. But in the Card Sorting prototype a physical medium was used.

6.1 Time2Play Prototyping Methods

Time2Play makes use of a reutilization strategy that joins a storytelling game, a 3D auditorium and different tools for creating a storytelling environment. This strategy takes into account the idea of a 2D

environment called Legal in which children learn by reading and writing stories [Pereira and Lopes]. Time2Play combines an auditorium and other elements developed in Second Life for offering the theater environment and tools like clothes or animations for enacting the stories.

- **Iteration 1. -** ***From Idea to Card Sorting.***

The idea of Time2Play comes from Legal. A Card Sorting prototype was used by the internal team to determine the conceptual overview of the game. Cards with the required features were sorted to define the high level functionality of the scenario panel, the clothes panels and the on-demand auditorium.

- **Iteration 2 -** ***From Card Sorting to Low-coded Prototype***

Card sorting prototype was used to get an overview of the conceptual model of the game. Then, a low-coded prototype was developed in Second Life. Several freebies were used to implement the scenarios and to offer clothes for enacting the play. It also reused the on-demand auditorium and transformed it to looks like a theater.

- **Iteration 3 -** ***From Low-coded to High-coded Prototype***

After the review session of the low-coded prototype, new functionalities were required: characters, hair and skins, special effects, animations and objects. All these features were added to the storytelling environment. A teacher was invited as a subject matter expert to validate the prototype.

- **Iteration 4 -** ***From High-coded Prototype to Product Version***

Finally usability tests validated the software requirements. Section 6.2 describes the process used in these tests.

6.2 Prototyping Time2Play Experience

Being a storytelling based game, Time2Play need resources for empowering their users to act as players in a play. Therefore, they need costumes, make-up, scenarios, story elements and special effects for enacting their roles.

In SL, costumes, make-up and story elements maybe bought, built or collected as freebies, the latter being the case in Time2Play. Scenarios were modeled and assembled using the appropriated story elements combined with animations and scripts. Special effects had to be programmed.

As the prototyping of the game was done in-world, all the changes applied to the visible objects were noticed by the members. The scripts were coded in a collaborative programming way. Due to the lack of an in-world version control system, developers were assigned different objects and tasks. The coordination was done using the local chat. They also used Instant Messages for asynchronous collaboration when they were not logged in SL.

Prototyping proved itself very useful in the interaction with the young users. They had all sort of desires that we wanted to accommodate in the game. In each prototyping cycle, a few of their whims were included. These cycles were even more important for the developers themselves. They used this interplay in order to add new features and functionalities for offering new enacting possibilities. For example, while performing Snow White and the Seven Dwarfs, when the fairy tells the prince to kiss Snow White, the kids loved to trigger the bubbly-butterfly-fairy-hearts-stars special effect, carefully scripted for their contentment.

After the third iteration in the prototyping process, 3 'proof of concept' sessions took place with 8 learners from age 7 to 12, divided into groups of 2 or 3. The usability test that was used [Dumas and Redish] consists in installing the prototype and evaluating its overall quality. It was observed the learner's navigation, understanding and interaction with the environment. Each learner used a pre-created avatar having navigation limitations on account of their age (younger than 18). Four phases of the usability test were executed: Profiling Questionnaire to figure out learners' profile, Training in Second Life and Time2Play, Main Task which is a challenge for collaborative building and enacting a story using Time2Play and Final Interview for giving learners the chance to express their views on the game and the technology. During the "Validate the Design" step in the last iteration (Section 5), the demand for additional characters, imagined or requested, brought evidence of the need to create non-player characters for this game.

7. Prototyping TREG

This section shows how the prototyping methods changed throughout the iterations in the TREG project. Also it explains our experience in the prototyping process.

TREG prototype characteristics were pre-defined for choosing the prototyping method for each iteration. These characteristics were based on the Effective Prototyping Process [Arnowitz et al. 2007].

- Audience: Internal. The researchers participated in the revisions of the prototypes.
- Speed: Rapid at the early iterations for obtaining a faster feedback and diligent at the last iterations for having more code time to get a high-fidelity and best quality prototype.
- Longevity: Short as the first prototypes were used for clarifying the project main idea. The last iterations took place having in mind a long longevity life as some of the objects were supposed to continue existing in the project.

- Expression: TREG focused on an experiential expression for testing the look and feel of the game. However, during the early iterations, a conceptual expression was needed for understanding the game unfolding using slides.
- Style: A narrative style was used in the early iterations of the process. The scenarios were specified in a template and shown using slides. Also there was a low-coded prototype in Second Life for narrating the branching stories.
- Medium: A digital medium was used for getting a true game representation.

7.1 TREG Prototyping Methods

TREG prototyping used a trial and error strategy, taking all together 5 iterations for refining the requirements and prototyping the workshop technique. There were several revision meetings during each iteration where the researchers compromised with a look and feel of the game. Second Life was used as the main tool for prototyping. Figure 9 shows the 5 iterations and the improvements of the lobby using the prototyping process.

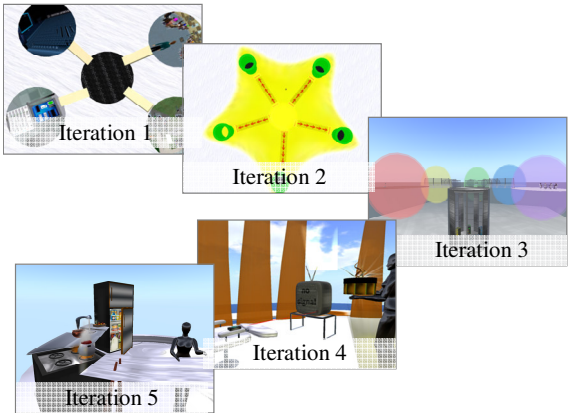


Figure 8.- Iterations improvements with the Effective Prototyping Process.

- **Iteration 1 -** ***From Idea to Quick Wireframe.***

At the very beginning of the prototyping process, the main idea was the creation of a training environment in Requirements Engineering. In the first prototype iteration, an idea of the game was conceived. A low-fidelity model presenting the first visualizations: the main reception, monoplayer games, a maze game, classrooms/discussion groups' rooms and an auditorium. It was instrumental to clarify that the goal was the creation of a classical environment for training.

- **Iteration 2 -** ***From Quick Wireframe to Wireframe***

Requirements process, techniques, management, analysis and validation were initially selected as tentative activities areas to be part of the game. A quick wireframe of a main lobby leading to 5 teleporting pods for going to these activities areas was prototype. Based on this high level structure wireframe, it was decided to narrow the prototyping to only one area, namely, the workshop technique.

- **Iteration 3 -** ***From Wireframe to Low-coded Prototype***

A new main lobby more aligned to the expected audience was prototype. Teleporting scripts were reused. In this iteration fleshed out the need of a simulation game genre, namely, branching stories. Storyboards prototyping method was not used. Detailed specifications were created using scenario templates.

- **Iteration 4 -** ***From Low-coded to High-coded Prototype***

A new high-fidelity and non-realistic main lobby was refined, including a NPC for guiding and giving the basic game resources for the trainee. A new communication feature was scripted in the NPCs and a metaphor relating cooking to making workshops was implemented.

As the simulations were defined and specified using branching stories and scenarios, the inclusion of state machine diagrams was an attractive option. Despite of the fact that they don't represent the look and feel of the game, these diagrams gave a new perspective of the game and improved the implementation in LSL.

- **Iteration 5 -** ***From High-coded prototype to Product Version.***

Finally, an advanced version for training in workshops was delivered. The main lobby, the HUD, the cooking metaphor and NPCs carried on in the following iterations for prototyping the other requirements techniques.

7.2 Prototyping TREG Experience

In order to keep the trainees immersed for enhancing their skills through a playful activity, real life metaphors are used for joining a common real life task together with a requirements engineering situation. For example, when the trainee is being trained in the workshop technique, a kitchen metaphor is proposed. The trainee enacts a *chef* role-play that must find the ingredients for the "making workshops" recipe. Figure 6 shows part of the environment implemented for this specific technique.

In order to prototype the game, first, the content was conceived taking into account requirements engineering concepts originated from Gottesdiener's book: "Requirements by Collaboration: Workshops for Defining Needs" [Gottesdiener 2002]. There she suggests 14 ingredients for accomplishing a successful requirements workshop. All these ingredients were reorganized using a structure based on the workshop process framework phases: plan, do check and act.

When the trainee chooses one of the ingredients for starting the training session, a situation related to the chosen ingredient is shown to the trainee revealing multiple paths to follow. Most paths lead to "making workshops" that don't really work. Then, collaboration patterns are offered to the trainee for choosing paths that do work. Figure 9 shows the TREG Design Process Box for the training process in the workshop technique. It takes the ingredients as input and the collaboration patterns as output.

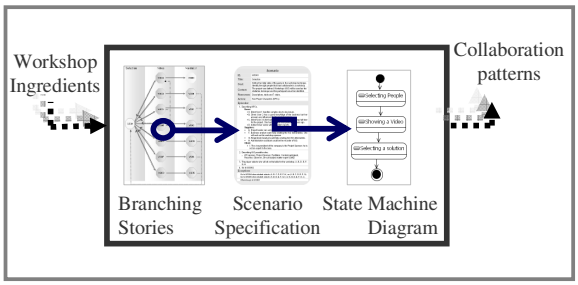


Figure 9. The TREG Design Process Box

Then, three implementation strategies were applied to design TREG. The game makes use of the branching stories genre for simulating situations that will immerse the trainee in a workshop scenario that must be planned, conducted and finished. Each node is described using the scenario template for fine tuning the situations specified in the branching stories graph. State machines use these diagrams, i.e., the specifications and transitions annotated in these templates, for modeling the system behavior.

The game makes use of the branching stories strategy for simulating different situations that will help the trainee to understand how a workshop must be planned, conducted and finished. Each state will be described using a Scenario template [Leite et al. 2000]. State machines diagrams model the game elements behavior.

Branching stories is one of the computer-based simulations genres defined by Aldrich [2005]. A graph with all the possible scenarios is created using the branching stories strategy for connecting and following the sequence of scenarios. The first trials were done using Storyboards. Unfortunately, this technique didn't. It consumed too much time for making the images to illustrate the game, instead of generating the content and sequences needed for the unfolding of the game. Branching stories, on the other hand, proved itself as a practical way to do the job.

Scenario is a description technique useful for depicting situations in an environment suitable for elicitation and specification of software requirements [Leite et al. 1997]. Each node of the branching stories graph will have a description based on a Scenario Template [Leite et al. 2000]. Moreover, exceptions in the Scenario Template lead to paths that clarify misunderstandings that are normally associated to trainees' choices.

State Machine Diagrams are used for modeling the dynamic perspective of the system. These diagrams define the states and transitions for each game object. Given that LSL is a state based language, from the early stages of the prototyping process, the relevance of these diagrams for implementing the system was clear.

All the objects and scripts were created from scratch for getting a more realistic and personalized setting. In the prototyping process, objects are built and shown as-is in-world in order to choose from the set of available objects those that will remain and those that will not be used in the next prototype. For example, 3 versions of the main building were discarded and a NPC walking capability was ruled out.

In the game, NPCs perform 3 specific functions: simulation, filming and guiding. They are used to simulate a specific workshop situation and interact with the trainee. They were filmed using the Machinima technique to create videos with several scenarios showed in-world to the trainee. Finally, some NPCs are presented to the trainees as in-world guides.

8. Conclusion

Prototyping helped to design and evaluate some aspects of the game in Second Life, and provided feedback for improving the quality of the game. The Second Life platform accommodates different development strategies as shown in Subsections 5.1 and 6.1. The prototyping process was customized for their development.

Time2Play and TREG are educational games with different audience and objectives. Time2Play is a storytelling game created for children. TREG is a game for training in software requirements engineering. Thus, they were developed in different ways:

- While Time2Play makes use of a reutilization strategy that joins a storytelling game, a 3D auditorium and different tools for creating a storytelling environment, the strategy adopted for TREG was a trial and error one where the requirements were refined.
- Time2Play was prototyped in a collaborative way; hence, some SL features had to be customized for this purpose.
- In Time2Play, the users participated in the "Validate the Design" step of the last iteration of the prototyping process. While they were playing the game, they asked for new features and some were included in the next iterations.
- Learners in Time2play had the permissions for building in the land and rezzing any objects during their performances or creating

another element for the scenario. TREG, on the other hand, was built and scripted from scratch. It demanded more time and investment for getting better resolution and definition of the objects, particles and scripts.

- TREG design was divided into two phases: Content creation and the combination of various implementation techniques. The content is based on the book: "Requirements by Collaboration: Workshops for Defining Needs" [Gottesdiener 2002]. Prototyping was instrumental in finding out the 3 techniques used in the implementation: branching stories for the unfolding of the game, scenarios for defining the game specification and state machine diagrams for modeling behavior.

There are some hardware and software specificities that were considered in these developments. They are related to bandwidth and rendering capabilities on the client side for preserving the quality of the play experience.

References

LINDEN LABS, 2009. Second Life website [online]. Available from: http://secondlife.com [Accessed 20 July 2009].

C. PETTEY, 2007. Gartner Says 80 Percent of Active Internet Users Will Have A "Second Life" in the Virtual World by the End of 2011, *Gartner Press*.

ALDRICH, C., 2005. Learning by doing; *Pfeiffer*, USA, 2005.

B. BOOK, 2009. What is a Virtual World. [online] Available from: http://www.virtualworldsreview.com. [Accessed 20 July 2009].

D. OBLINGER, 2004. The next Generation of Educational Engagement, *Journal of Interactive Media in Education,* SSN:1365-893X, May 21st, 2004.

M. CUNHA, A. RAPOSO, H. FUKS, 2008. Educational Technology for Collaborative Virtual Environments, *In Proceedings. of 12th International Conference on CSCW in Design,* China, April 16-18, 2008.

S. KLUGE, L. RILEY, 2008. Teaching in Virtual Worlds: Opportunities and Challenges, *Issues in Informing Science and Information Technology*, Informing Science Institute, 2008.

KIDZ CONNECT, 2009. Kidz Connect: Connecting cultures through creative collaboration [online]. Available from: http://www.kidzconnect.org/». [Accessed 20 July 2009].

RAMLOLL, R., BEEDASY, J., HUDNALL STAMM, B., PILAND, N., CUNNINGHAM, B., KIRKWOOD, A., MASSAD, P., SPEARMAN, R., PATEL, A., TIVIS, R. AND C. KELCHNER, 2006. Distance Learning and Simulation Technologies to Support Bioterrorism Preparedness Education, *In Proceedings of the ISCA 21st International Conference*, ISBN: 1-880843-58-7, pp 235-241.

IDAHO STATE UNIVERSITY, 2009. Virtual training at Bingham Memorial Hospital [online], Available from: http://play2train.us, [Accessed 20 July 2009].

EN YE, CHANG LIU, J. POLACK-WAHL, 2007. Enhancing Software Engineering Education Using Teaching Aids in 3-D Online Virtual Worlds", *37th ASEE/IEEE Frontiers in Education Conference T1E-8*, Milwaukee, WI, October 2007.

M. ERNST, 2006a. The Groupthink Specification Exercise, In *Software Engineering Education in the Modern Age: Challenges and Possibilities*, Lecture Notes in Computer Science vol. 4309, Dec. 2006, pp. 89-107.

ERNST, 2006B Groupthink exercise at Ohio University in Second Life. [in-world]. Available from: http://slurl.com/secondlife/OHIO%20Outreach/144/156/32 [Accessed 20 July 2009].

EMILY OH NAVARRO, ANDRE VAN DER HOEK, 2007. Comprehensive Evaluation of an Educational Software Engineering Simulation Environment, *20th Conference on Software Engineering Education & Training (CSEET'07).* Pp.195-202.

IBM, 2009. Open Encounters of z Virtual Kind skills challenge [online]. Available from: http://www-304.ibm.com/jct01005c/university/students/contests/SecondLife/index.html [Accessed 20 July 2009].

R. BARTLE, 2004. Designing Virtual Worlds. New Riders Games.

M. NITSCHE, 2005. Film live: And Excursion into Machinima", In: Developing Interactive Narrative Content: sagas_sagasnet_reader, *Brunhild Bushoff, Munich*, 210-243, 2005.

Arnowitz, J., M. Arent and N. Berger, 2007. Effective Prototyping for Software Makers, Morgan Kaufmann, Elsevier, Inc.

PEREIRA, A. R. ; LOPES, R. D. 2005 . Legal: Ambiente de Autoria para Educação Infantil apoiada em Meios Eletrônicos Interativos.. *In: Simpósio Brasileiro de Informática na Educação*, Juiz de Fora.

DUMAS, J., REDISH, J., 1999 A practical guide to usability testing *Intellect Books*.

E. GOTTESDIENER, 2002 Requirements by Collaboration: Workshops for Defining Needs, Addison Wesley, 2002.

LEITE, G. HADAD, J. DOORN, G. KAPLAN, 2000.A Scenario Construction Process. *In Proceeding: Requirements Engineering,* 2000; pags 38 - 61.

J. LEITE, G. ROSSI, F. BALAGUER, V. MAIORANA, G. KAPLAN, G. HADAD, A. OLIVEROS, 1997; Enhancing a Requirements Baseline with Scenarios. *In Proceedings: Requirements Engineering, Springer*; pags 84-198.

Support Vector Machines for Cinematography Real-Time Camera Control in Storytelling Environments

Edirlei E. S. de Lima[1] Cesar T. Pozzer[1] Eduardo C. D. Favera[1] Marcos C. d'Ornellas[1]
Angelo E. M. Ciarlini[2] Bruno Feijó[3] Antonio L. Furtado[3]

[1]UFSM, Depto. de Eletrônica e Computação, Brasil
[2]UNIRIO, Depto. de Informática Aplicada, Brasil
[3]PUC-Rio, Depto. de Informática, Brasil

Figure 1: Camera shots selected by the director using support vector machines.

Abstract

This paper proposes an intelligent cinematography director for camera control in plot-based storytelling systems. The role of the director is to select in real-time the camera shots that best fit for the scenes and present the content in an interesting and coherent manner. Director's knowledge is represented with a collection of support vector machines (SVM) trained to solve cinematography problems of shot selection. With this work we introduce the use of support vector machines, applied as an artificial intelligence method, in a storytelling director. This approach also can be extended and applied in games and other digital entertainment applications.

Keywords: Storytelling, Cinematography, Artificial Intelligence, Support Vector Machine.

Authors' contact:
edirlei@msn.com
{pozzer,ornellas,favera}@inf.ufsm.br
angelo.ciarlini@uniriotec.br
{bfeijo,furtado}@inf.puc-rio.br

1. Introduction

Current advances in graphic technologies are paving the way to realistic digital entertainment applications. However, with this evolution new challenges have emerged. One area that deserves emphasis and has been the target of several researches in last the years is the application of cinematography in games and storytelling applications.

Cinematography is defined as the art of film-making. It consists of techniques and principles that control how a film should be produced and filmed. Most of the principles of cinematography are about how a camera should be used in order to accomplish tasks such as engaging the interest of the viewer, enhancing and clarifying the narrative, and presenting the content in an interesting and coherent manner. Viewers are used to a general storytelling pattern. Therefore, when they watch a single movie, they unconditionally try to impose a pattern of his/her own.

In this paper, we focus on the application of cinematography concepts to storytelling applications. Interactive storytelling is a new medium of digital entertainment where authors, audience, and virtual agents engage in a collaborative experience. It can be seen as a convergence of games and filmmaking. Storytelling systems can be divided in two different models. The first model corresponds to the character-based approach [Cavazza et al. 2002; Mateas and Stern 2000; Young 2000] where the storyline usually results from the real-time interaction among virtual autonomous agents that usually incorporates a deliberative behavior. The main advantage of a character-based model is the ability of anytime user intervention. As a result of such strong intervention, there is no way to estimate what decisions or actions will be made by the virtual actors. The director does not have then the same control over the process as it usually occurs in real filmmaking. The other model corresponds to the plot-based approach [Grasbon and Braun 2001; Spierling et al. 2002], where characters incorporate a reactive behavior, which follows rigid rules specified by a plot. The plot is usually built in a stage that comes before dramatization. This approach ensures that actors can follow a predefined script of actions that are known beforehand. The script may be

built automatically from a plot or with the help of the author.

To apply cinematography concepts in storytelling applications there are two most common approaches. The first approach is the use of film idioms, which represents the most usual way to present a specific type of scene. Idioms are used in works such as Charles et al. [2002]. The second approach is the division of the system in different modules or agents that represent the various roles people play in a movie set, such as in Hawkins [2004]. In other works, such as Courty et al. [2003] both approaches are used. However, these works have only superficially incorporated cinematography rules.

This paper proposes a cinematography director for plot-based storytelling systems. The director uses a collection of Support Vector Machines (SVM) trained with cinematography knowledge to select, in real-time, the best shots for the dramatization of scenes. The rest of the paper is organized as follows: section 2 compares our approach with previous research. Section 3 presents the principles and concepts of cinematography. Section 4 presents our system architecture. Section 5 brings a detailed look at the director implementation. In section 6, we analyze the performance and accuracy results to demonstrate the efficiency of our approach. Finally, in section 7 we present the concluding remarks.

2. Related Works

Many works have already been done with the objective of applying concepts of cinematography in games. The basic principle of camera positioning employing cinematography knowledge in form of idioms was first explored by Christianson et al. [1996]. These idioms encapsulate the combined knowledge of several personal roles in a traditional filming set and are widely used in research involving camera systems. However, film idioms are only able to solve the problem of direct manipulation of the virtual camera. In other works, such as Hawkins [2004], the system is divided in different modules or agents, the most common approach consider three elements: director, editor and cinematographer. This approach is known to be a better solution since some cinematography rules do not only involve the camera manipulation.

In research involving cinematography applied to storytelling systems, there is a clear distinction between the techniques that can be applied to character-based and plot-based approaches. Plot-based applications give access to all the actions before camera planning, allowing the system to have a greater control of the scenes based upon pure cinematography knowledge. Character-based applications do not allow the same level of control over the scenes, making camera planning more complicated, since all information is sent in real-time to the camera system.

The first camera system in character-based storytelling applications was developed by He et al. [1996]. They organized film idioms as nodes of hierarchical trees. Each idiom operates as a state machine and defines the scene shots to be used. Halper et al. [2001] has proposed a camera control based upon constraint specifications; however high constraint satisfaction implies in poor frame coherence. Charles et al. [2002] have explored architectural and organizational concepts to achieve satisfactory camera planning when we have different context timeline stories that can be alternated with the flow of the time. In plot-based applications, Courty et al. [2003] introduces a scheme for integrating storytelling and camera systems.

Current approaches only reach superficial implementation, and do not provide a good dramatization quality to become comparable with a real movie. In this work, we try to contribute towards this goal by proposing a novel approach for the architecture and implementation of a cinematography virtual director.

3. Principles of Cinematography

The term cinematography was created in the film industry a long time ago to describe the process of creating images on film. With the advancement of industry and the emergence of new technologies in digital video with high definition formats, the tern are expanded. Now it is understood as a generic term covering all aspects of camera work, including the creative aspects involved with making aesthetically pleasing images and the technical aspects involved with using cameras, lights, and other equipment [Newman 2008].

Although a film can be considered a linear sequence of frames, it is often helpful to think of a film as having a structure. At the highest level a film is a sequence of scenes. Each scene is composed of a number of shots, a shot being a continuous view filmed by one camera without interruption. The transition from one shot to the next is known as a cut.

The size of the image on the film is determined by the distance of the camera from the subject. The closer is the camera, the larger is the image. This distance defines a shot type. Supposing that the subject is a character, an example of shot type is the medium shot, which depicts characters from the thighs to above the head. Another example is the close-up, which depicts them from the chest to above the head [Mascelli 1998].

The type of camera angle strongly influences the way a scene is perceived by the viewers. It also defines how viewers may become part of the action. When a choice is made to the objective angle, the viewer sees

the event on screen as if an unseen observer [Mascelli 1998]. A subjective camera angle makes the viewer a part of the scene.

Another important aspect of filmmaking corresponds to the camera movements. They affect the aesthetic and psychological properties of a scene. An example of camera movement is the tracking, when the camera moves alongside a character while filming, giving to the viewers the feeling that they are walking alongside the character [Mascelli 1998]. Movements should be executed in such a way that the viewer does not get disoriented.

Cinematography is a complex process, and many rules demand human interpretation of the scenes to be correctly applied. However, cinematographers have defined some heuristics for selecting good shots [Arijon 1976]. Some examples are:

- **Create a line of action:** this line should connect the two major points in one scene (most of the times, the two actors that interact in the scene);
- **Parallel editing:** Scenes should alternate between different contexts, locations and times;
- **Show only peak moments of the story:** Repetitive movements should be eliminated;
- **Don't cross the line:** Once a scene is taken by a side of the interest line, the camera should in principle keep itself in that side, not making unexpected movement shots. The camera can switch sides, but only upon an establishing shot, that shows that transition;
- **Let the actor lead:** The actor should initiate all movement, and the camera should come to rest a little before the actor;
- **Break movement:** A scene illustrating a movement must be broken in two shots at least.

4. System Architecture

Our storytelling system architecture is organized in four modules. The Scriptwriter is responsible for controlling the plot and the story flow; the Scenographer is responsible for creating and arranging the sceneries; the Director defines how scenes will be filmed; and the Cameraman is responsible for positioning the cameras. Figure 2 shows a diagram of the architecture.

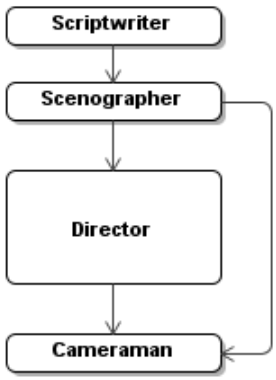


Figure 2: System architecture

The main component of our architecture and the focus of this work is the Director. It concentrates the cinematography knowledge and decides, in real-time, the best way to present scenes. The knowledge is represented by means of several support vector machines trained to solve cinematography problems involving camera shot selection. Support vector machines are used as an effective method for general purpose pattern recognition; they are based on statistical learning theory and are specialized for small sample sets [Tyagi 2008]. A similar approach is used by Passos et al. [2008] to select camera shots in a race car game using a neural network classifier. Support vector machines have better generalization than neural networks and guarantee local and global optimal solutions similar to those obtained by neural networks [Gunn 1998]. In recent years, support vector machines have been found to be remarkably effective in many real-world applications such as in systems for detecting microcalcifications in medical images [El-Naqa et al. 2002], automatic hierarchical document categorization [Cai et al. 2004], spam categorization [Drucker 1999], among others.

In our system, the modules are agents that communicate with each other by means of message exchange and can be summarized as follows:

1. The Scriptwriter reads the information about the current scene from the story plot and sends it to the Scenographer;
2. The Scenographer prepares the actors and scenario for the scene dramatization and also places objects and involved actors in the scene. The information about the scenario is sent to both the Cameraman and the Director;
3. The Cameraman, following cinematography rules, places a set of cameras in the scene for all possible shots for the current scene;
4. The Director extracts from the scene all important data and applies them to a support vector machine to select the best shot for the scene. This information is then sent to the Cameraman;

5. The Cameraman activates the shot selected by the Director and, if necessary, executes a camera movement or zooming operation.

5. The Director

In a film production, the director creatively translates the written word into specific images. He visualizes the script by giving to abstract concepts a concrete form. The director establishes a point of view on the action that helps to determine the selection of shots, camera placements and movements. The director is responsible for the dramatic structure and directional flow of the film.

In our system, the role of the director is to choose which shot should be used at each time to highlight the scene emotion and to present the content in an interesting and coherent manner. To perform this task, the director uses a collection of support vector machines trained to classify the best shots for the dramatization scenes.

The process consists of two steps. First, the training process, which is done before the story dramatization, consists in simulating some common scenes and defining the solution for the shot selection. The features of these scenes, actors and environment are used to teach the support vector machine how to proceed in this situation in order to detect similar situations in the future. The second step is the prediction process that is done in real-time during the dramatization by using the knowledge acquired through the training process to predict (classify) an unknown situation. Subsequent sections detail all this process.

The input of our support vector machines are the important features from the environment, scene, and involved actors. The output is the selected shot that best matches with the input features, as shown in Figure 3.

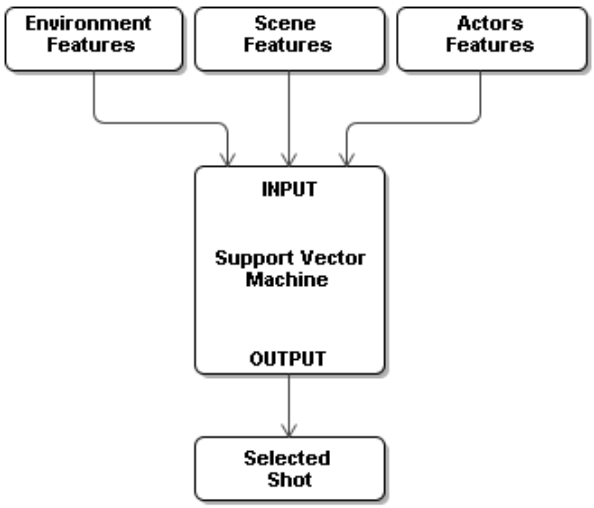


Figure 3: Support vector machine input and output

5.1 Support Vector Machine

The support vector machine, proposed by Vapnik [1995], is a powerful methodology for solving machine-learning problems. It consists of a supervised learning method that tries to find the biggest margin to separate different classes of data. Kernel functions are employed to efficiently map input data, which may not be linearly separable, to a high dimensional feature space where linear methods can then be applied.

The original idea of SVM is to use a linear separating hyperplane to separate the training data set into two classes. Figure 4 shows an optimal hyperplane separating the blue class from the green class.

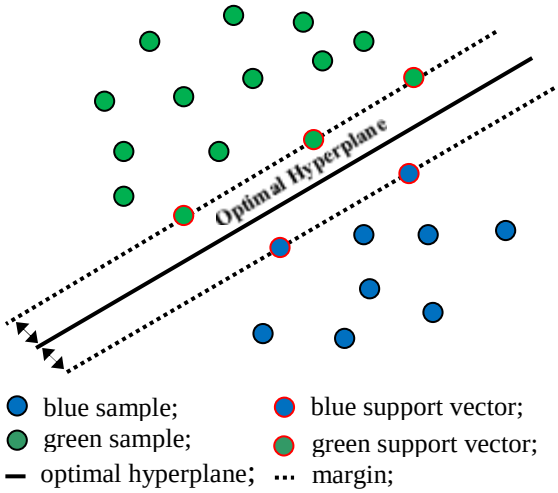


Figure 4: Optimal hyperplane separating two classes

Suppose the training data $(x_1, y_1), \cdots, (x_l, y_l)$, where each sample $x_i \in R^n$ belongs to a class $y_i \in \{-1, +1\}$.

The boundary hyperplane can be as follows:

$$\omega \cdot x + b = 0$$

and the separate margins as:

$$\omega \cdot x + b = +1$$
$$\omega \cdot x + b = -1$$

where,

- ω is a weight vector;
- b is a bias;
- x is a point in the space R^n.

This set of vectors is separated by the optimal hyperplane if and only if it is separated without error and the distance between the closest vector and the hyperplane is maximal. The separating hyperplane can be described in the following form:

$$\begin{cases} \omega \cdot x_i + v \geq +1, & if \quad y_i = +1 \\ \omega \cdot x_i + v \leq -1, & if \quad y_i = -1 \end{cases}$$

or equivalently:

$$y_i(\omega \cdot x_i + b) \geq 1, \qquad i = 1, \ldots, l$$

The optimal hyperplane is the one that satisfies the conditions and minimizes the function: $\frac{1}{2}\|\omega\|^2$.

Vapnik [1995] has shown that, to perform this minimization, we must maximize the following function with respect to the variable α_i:

$$W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

subject to $0 \leq \alpha_i$, $i = 1, \ldots, l$ and $\sum_{i=1}^{l} \alpha_i y_i$

Those $x_i s$ with $0 < \alpha_i$ are termed Support Vectors. The support vectors are located on the separating margins and are usually a small subset of the training data set, denoted by X_{SVM}.

For an unknown vector x_i, its classification corresponds to finding:

$$f(x) = sign\left(\sum_{x_i \in X_{SVM}} \alpha_i y_i (x \cdot x_i) + b \right)$$

where

$$\omega = \sum_{x_i \in X_{SVM}} \alpha_i y_i x_i$$

and the sum is over those nonzero SVs with $0 < \alpha_i$. In other words, this process corresponds to finding which side of the hyperplane the unknown vector belongs.

However, in most cases, the classification is not so simple, and often more complex structures are needed in order to make an optimal separation. For example, in Figure 5, the separation requires a curve that is more complex than a simple line.

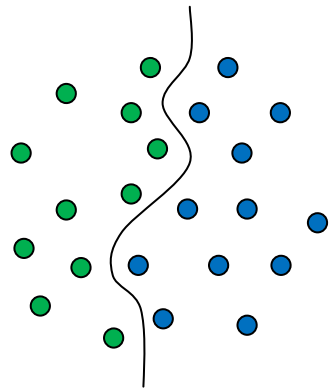


Figure 5: Non-linearly separable classes

To construct the optimal hyperplane in the case when the data is linearly non-separable, SVM uses two methods. First, it allows training errors. Second, it non-linearly transforms the original input space into a higher dimensional feature space by a function $\varphi(x)$. In this higher space, it is possible that the features may be linearly separated [Wang and Zhong 2003]. Then the problem can be described as:

$$\min \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{l} \xi_i \quad (2)$$

subject to

$$y_1(\omega \cdot \phi(x_i) + b) \geq 1 - \xi_i, \ \xi_i \geq 0, \ i = 1, \ldots, l, \ C > 0$$

A penalty term $C\sum_{i=1}^{l} \xi_i$ in the objective function takes the training errors into account. If the data are linear separable, problem (2) goes back to (1) as all ξ_i will be zero. We can equivalently maximize $W(\alpha)$ but the constraint is now $0 \leq \alpha_i \leq C$ instead of $0 \leq \alpha_i$:

$$W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \left(\phi(x_i) \cdot \phi(x_j) \right)$$

subject to

$$0 \leq \alpha_i \leq C, \ i = 1, \ldots, l \text{ and } \sum_{i=1}^{l} \alpha_i y_i = 0$$

The inner products in the high-dimensional space can be replaced by some special kernel functions. Some popular kernels are radial basis function kernel and polynomial kernel.

For example, to linearly separate the classes showed in Figure 5, the classes need to be mapped and rearranged using a kernel function in a high-dimensional space. After the mapping, classes become linearly separable and the optimal hyperplane can be created (Figure 6).

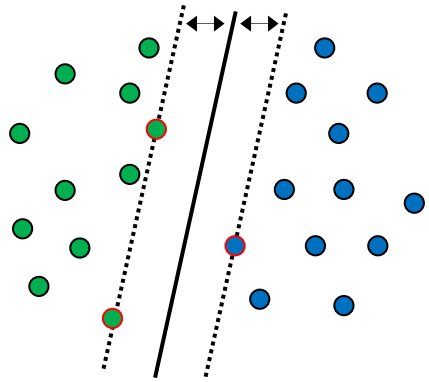


Figure 6: Classes mapped and rearranged to become linearly separable.

Support vector machines were originally created for binary pattern classification. For our problem, a multi-class pattern recognition is necessary because in most part of the scenes we have more than two possible shots for the same scene. To solve this problem, we use the "one-against-one" approach [Knerr et al. 1990] in which classifiers are constructed and each one trains data from two different classes, creating a combination of binary SVMs. The first use of this strategy on SVM was by Friedman [1996]. In classification we use a voting strategy to decide the class of the input pattern.

5.2 Training Process

Before using support vector machines to select the shots in our dramatization, they have to be trained to acquire the necessary knowledge to create the optimal hyperplane separating the shots, so that it can be used to predict the best shot for new scenes.

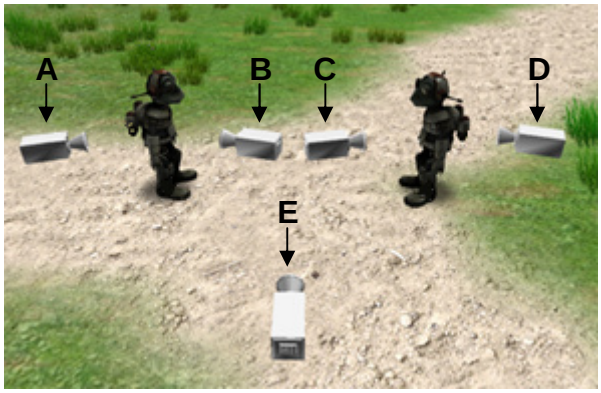
In order to train the support vector machines, we simulate some common situations that happen in real films. Based on cinematography rules and principles, we perform the selection of best shots for these scenes and store them in a database together with features from the simulated scenes. The training database is composed of several samples of simulated scenes, each one with the features and the selected shot for the simulated scene. This training database is created once and is used in all future dramatizations.

The used features are:

- Normalized values of the position (X, Y and Z) (relative to the center of scene) of the actors involved in the scene. These values influence the camera shot in action scenes when the position of the actor can change during the dramatization.
- The current emotional state of the actors involved in the scene (happiness, sadness, angriness or scariness). The emotional state in most cases influences the selected shot to highlight the emotional actor state.
- The acting or talking actor. This feature is the most important because the actor must be visible in the shot.

Numerical values are associated with the abstract types. The emotional state happiness is, for example, represented by the value 1, sadness by the value 2. All features are then normalized (between -1 and 1).

The classes are the possible shots (camera angles) for the scene. These shots are defined in our system by the Cameraman module, which, for each scene, creates a line of action and positions the cameras in an appropriated location, improving the scene visualization by following standard cinematography rules and patterns proposed by Arijon [1976]. For example, in a dialog scene between two actors (Figure 7) there are 5 possible shots (classes); camera A and camera D highlight the viewer's attention to one actor while keeping the other actor visible in the scene; camera C and camera B highlight the attention only to one actor and emphasizes his emotional state; and camera E shows both actors. For this scene, we can extract 9 features: the position X, Y and Z of the two actors (6 features), the emotional state of the two actors (2 features), and the active talking actor (1 feature).



Camera A

Camera C


Camera D


Camera B


Camera E

Figure 7: Possible camera shots for a dialog scene.

For each type of scene we have a different support vector machine; the number of features (inputs) and classes (outputs) depends on the type of scene and number of involved actors. Figure 8 illustrates this combination of support vector machines. The director has N support vector machines and each one with different inputs and outputs.

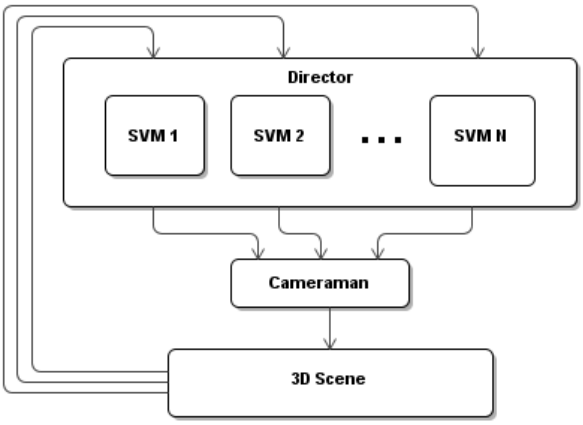


Figure 8: Director Architecture.

5.3 Predicting Process

With the support vector machines trained with cinematography knowledge, the Director module is able to act as a film director and, based on the previous experience, select in real-time the best shots to show the scenes.

To predict the best shot, the director executes the following steps:

- Selects the active support vector machine based on the type of the current scene;
- Extracts the features from the active environment and actors; these features are the same used to create the training database;
- Applies the extracted features to the support vector machine;
- Use the support vector machine output to set the active camera. The result of our support vector machine is the camera shot classified as the best solution to show the scene.

The scenes are composed by different shots; the transition between the shots occurs when an important event happens in the scene, for example when the emotional state of an actor changes or when an actor executes an action. The director detects in real-time these events and executes the predicting process to use the support vector machine knowledge to choose the new shot.

Consider a scene where the actor chases an animal (Figure 9). We have two possible shots for this scene: camera A and camera B. The director detects in real-time the type of the scene and activates the support vector machine for chasing scenes. Every time when a new support vector machine is selected an initial shot must be selected, so the director extracts from the environment the features used by the active support vector machine and apply these features to it; the support vector machine applies then the classification algorithm to determine the shot that best fits the current scene; finally, the director sends this selection to the Cameraman module which activates the selected camera. When a new important event occurs, for instance, while along the chase the actor speaks something, the director executes the prediction process again, and probably that action will influence on the selected shot.


Camera A


Camera B

Figure 9: Possible camera shots for a chasing scene

6. Results

To validate our architecture we run two tests: first the performance test, to check the necessary time to predict a new shot. The second test is the recognition rate, to check the accuracy of the predicted shots. The tests have occurred on an Intel Core 2 Quad 2.40 GHz, 4 GB of memory, using a single core to process the support vector machines.

To test the performance of our proposed solution, we trained our support vector machines with a different number of samples and use them to predict the shots for a sequence of 6 scenes, with a total of approximately 40 different shots. For each shot, we calculate the necessary time for the prediction process. Figure 10 shows performance results in a line chart with the training set size ranging from 10 to 55 samples and the times correspond to the average of the all support vector machines trained with the current number of samples.

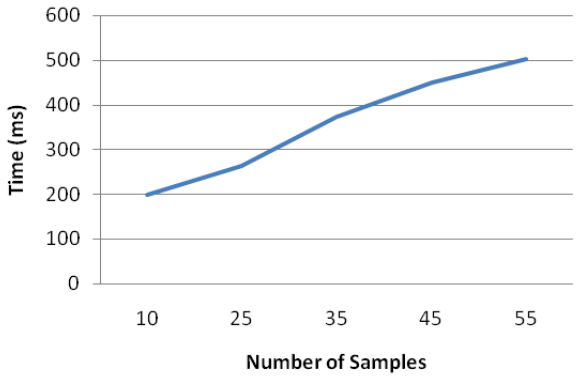


Figure 10: Prediction performance test with different training sets.

To test the recognition rate, for each support vector machine in our system, we created 5 training sets with a different number of samples and, for each one, a testing set with half the size of the corresponding training set. The training sets are used to train the support vector machine and the samples of the current test set are predicted. Correct and wrong predicted shots are then computed. Table 1 shows the computed results of this test with the training set size ranging from 10 to 55 samples. The presented percentages of accuracy correspond to the average of the results obtained for the different support vector machines.

Table 1: Recognition rate with different training sets.

Number of Samples	10	25	35	45	55
Accuracy	92%	94.6%	96.5%	98%	98.6%

Figure 11 shows the results of this test in a line chart.

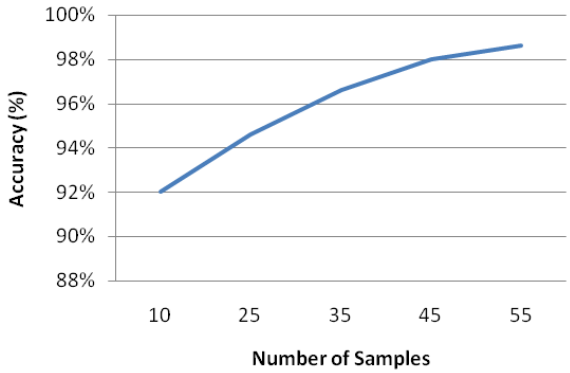


Figure 11: Recognition rate with different training sets.

It is clear that the computational cost grows almost linearly with the number of samples. More samples result in a high accuracy but in slow recognition; few samples result in a fast recognition but in a low accuracy. However, with small training sets we obtain high percentage of correct recognition of the best shots, ensuring high accuracy in the shot selection and without high computational costs.

7. Conclusion

In this paper we have presented an intelligent cinematography director that uses a collection of support vector machines trained with cinematography knowledge to select in real-time the best scene shots in storytelling dramatization. Our methodology is applicable not only to storytelling systems; it can be adapted to other entertainment applications, such as games, virtual worlds or 3D simulations.

In our tests, support vector machines showed to have excellent recognition rate (between 92% and 98%) with small training sets and without high computational cost (less that 1 second to predict). This approach ensures that most of the times the selected shots are the best solution to show the scene in accordance with cinematography principles and rules.

Support vector machine is a powerful machine learning methodology, however, still not widely explored in the area of artificial intelligence for games. In this paper, we have shown that support vector machines can be successfully applied in storytelling to select camera shots in real-time. Extending the use of support vector machines in games and entertainment computing in general is a promising approach to implement other artificial intelligence and machine learning tasks, such as controlling the behavior of non-player characters. Training our support vector machines at real-time in accordance with feedback provided by the users is also an interesting point to be investigated.

Acknowledgements

This work was supported by CAPES/RH-TV-Digital, CAPES/PROCAD, and CNPq. Authors would like to express their gratitude to Laboratório de Computação Aplicada (LaCA) - UFSM.

References

ARIJON, D., 1976. Grammar of the Film Language. Communication Arts Books, Hasting House, Publishers, New York.

CAI, T., HOFMANN, T., 2004 Hierarchical document categorization with support vector machines. In Proceedings of the ACM 13th Conference on Information and Knowledge Management.

CAVAZZA, M., CHARLES, F. AND MEAD, S., 2002. Character-based interactive storytelling. IEEE Intelligent Systems, special issue on AI in Interactive Entertainment, 17(4):17-24.

CHARLES, F., LUGRIN, J., CAVAZZA, M. AND MEAD, S., 2002. Real-time camera control for interactive storytelling. In Proceedings of the Game On, London, UK.

CHRISTIANSON, D. B., ANDERSON, S. E., HE, L., COHEN, M. F., SALESIN, D. H., WELD, D. S., 1996. Declarative Camera Control For Automatic Cinematography. In Proceedings of the AAAI '96, 148-155.

COURTY, N., LAMARCHE, F., DONIKIAN, S. AND MARCHAND, E., 2003. A cinematography system for virtual storytelling. In Proceedings of the International Conference on Virtual Storytelling, Toulouse, France.

DRUCKER, H., WU, D., VAPNIK, V., 1999. Support Vector Machines for Spam Categorization IEEE Trans. on Neural Networks , vol 10, number 5, pp. 1048-1054.

EL-NAQA, I., YANG, Y., WERNICK, M. N.; GALATSANOS, N. P.; NISHIKAWA, R. M., 2002. A support vector machine approach for detection of microcalcifications. IEEE Trans. on Medical Imaging, vol. 21, no. 12, pp. 1552-1563.

FRIEDMAN, J., 1996. Another approach to polychotomous classification. Technical report, Department of Statistics, Stanford University.

GRASBON, D. AND BRAUN, N., 2001. A morphological approach to interactive storytelling. In: Fleischmann, M.; Strauss, W., editors, In Proceedings of the CAST01, Living in Mixed Realities, Sankt Augustin, Germany, p. 337-340.

GUNN, S., 1998. Support Vector Machines for Classification and Regression. Technical Report, University of Southampton.

HALPER, N., HELBING, R., STROTHOTTE, T., 2001. A camera trade-off between constraint satisfaction and frame coherence. Eurographics, volume 20.

HAWKINS, B. 2004. Real-Time Cinematography for Games (Game Development Series). Charles River Media, Inc., Rockland, MA, USA.

HE, L., COHEN, M., AND SALESIN, D. 1996. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In Proceedings of the ACM SIGGRAPH '96, 217-224.

KNERR, S., PERSONNAZ, L., AND DREYFUS, G., 1990. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In J. Fogelman (ed.), Neurocomputing: Algorithms, Architectures and Applications, Springer.

MASCELLI, J., 1998. The Five C's of Cinematography: otion Picture Filming Techniques. Silman-James Press, Los Angeles.

MATEAS, M. AND STERN, A., 2000. Towards integrating plot and character for interactive drama. In Working notes of the Social Intelligent Agents: The Human in the Loop Symposium. AAAI Fall Symposium, p. 113-118.

NEWMAN, R., 2008. Cinematic game secrets for creative directors and producers. Focal Press, UK.

PASSOS, E.; MONTENEGRO, A.; AZEVEDO, V.; APOLINARIO, V.; POZZER, C.; CLUA, E. W. G., 2008. Neuronal Editor Agent for Game Cinematography. In Proceedings of the VII Games and Digital Entertainment Symposium. Belo Horizonte, p. 91-97.

SPIERLING, U., BRAUN, N., IURGEL, I. AND GRASBON, D., 2002. Setting the scene: playing digital director in interactive storytelling and creation. Computer and Graphics 26, 31-44.

TYAGI S., 2008. A Comparative Study of SVM Classifiers and Artificial Neural Networks Application for Rolling Element Bearing Fault Diagnosis using Wavelet Transform Preprocessing. In Proceedings of World Academy of Science, Engineering And Technology Volume 33, p. 319-327.

VAPNIK, V., 1995. The Nature of Statistical Learning Theory. Springer, New York.

WANG, X., ZHONG, Y., 2003. Statistical Learning Theory and State of the Art in SVM. In Proceedings of the 2nd IEEE International Conference on Cognitive Informatics, p. 55 - 60.

YOUNG, R., 2000. Creating interactive narrative structures: The potential for AI approaches. In Proceedings of the AAAI Spring Symposium in Artificial Intelligence and Interactive Entertainment, Palo Alto, California. AAAI Press.

trAIns: An Artificial Inteligence for OpenTTD

Luis Henrique Oliveira Rios, Luiz Chaimowicz

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

Figure 1: *The four transport types available in OpenTTD: aerial, maritime, railroad and road.*

Abstract

Simulation games present several challenges for computer controlled players. As a result of this, most of the artificial intelligence algorithms developed so far, especially for construction and management simulation games, do not give satisfactory results when compared to human performance. In this paper we develop an AI to control an agent of OpenTTD, a open-source clone of Transport Tycoon Deluxe, one of the premier construction and management simulation games. To do this, we create and adapt artificial intelligence techniques to allow their use in a dynamic, multi-agent strategic environment. Named *trAIns*, the developed AI constructs and manages railroad routes, the most challenging transport type in the game. Several qualitative and quantitative experiments comparing *trAIns* with another AI are performed, bringing very good results.

Keywords: Artificial Intelligence, Construction and Management Simulation Games, Path Planning, OpenTTD.

Author's Contact: {lhrios, chaimo}@dcc.ufmg.br

1 Introduction

The advances in hardware and software have allowed game developers to improve the quality of digital games, augmenting their degree of immersion and realism. The increase in game complexity has demanded the development and adaptation of techniques to deal with a great number of variables and details still respecting time constraints. In this context, the research and development of artificial intelligence algorithms have gained importance. The quality of other game elements such as graphics and gameplay has increased players' expectations regarding artificial intelligence. Thus, it is becoming one of the main components of digital games and should have a level of sophistication similar to other game elements.

In digital games, artificial intelligence algorithms are responsible for making decisions and determining actions of game agents - we will call agent a character or institution in the game that is controlled by a computer. A general metric is that these algorithms must generate actions and decisions resulting in behaviors similar to the ones caused by human players [Byl 2004]. A common problem faced by these algorithms is the time constraints. Despite the increase of computational resources available, the response time is critical because most of the games happen in a dynamic, fast paced environment. These requirements demand an adaptation of classical approaches and the development of new AI algorithms.

In this paper, we are interested on artificial intelligence algorithms for simulation games. Among the several existing modalities, we focus on a style called simulation of construction and management (as stated by [Rollings and Adams 2003] taxonomy). Some classical examples are: *Capitalism*, *Caesar*, *SimCity* and *Transport Tycoon*. In this game style, the player's main goal is to construct, manage and expand communities, institutions, companies or empires using limited resources. Because of the complexity involved in constructing and managing resources as well as the time restrictions, artificial intelligence algorithms suitable for these games are not sophisticated - to our knowledge, there are few artificial intelligence techniques developed for simulation games that are able to deal well with these issues. Therefore, when compared to human players, the generated behavior is poor.

Thus, the main objective of this paper is to adapt, implement and evaluate artificial intelligence algorithms to control agents in construction and management simulation games. In particular, this work considers AI algorithms that will be used to control an agent in a game called OpenTTD [OpenTTD 2009].

OpenTTD is an open-source clone of Transport Tycoon Deluxe, a game released in 1994. The main objective is to construct and manage routes to become the transport tycoon. To achieve this, players must build lucrative transport routes connecting industries and cites. There are four kinds of transport types (figure 1): railroad, road, aerial and maritime. Normally, the most used is the railroad since it is capable of carrying much cargo for great distances in a fast way. It is also the most challenging one as will be discussed.

In this paper, we will use the acronym AI to denote a set of algorithms that control an agent in the game. These algorithms are implemented using a specific API and can play against human players. Currently, there are about 13 AIs available for the game, but only four use railroads. These AIs have several problems in common - part of them caused by the naive approaches adopted. For example, they can not build complex railroads, are not able to plan large routes, can not change the railroad track type and use poor algorithms to choose the locomotive engines. Furthermore, the design of tracks constructed by the AI is very different from the ones built by human players. These problems affect the performance of the company controlled by the computer and influence the gameplay.

Thus, the main contribution of this work is the development of *trAIns*, an AI specifically developed for the construction and management of railroads in OpenTTD. Adapting traditional AI algorithms such as A* and proposing new techniques for a dynamic, multi-agent environment, *trAIns* solves most of current OpenTTD AIs' problems and introduces many other improvements. *trAIns* is evaluated qualitatively, analyzing its construction decisions as well as quantitatively, comparing its performance against Admiral AI, the most complete AI current available in the game.

This paper is organized as follows: next section presents OpenTTD, describing its main features. Section 3 discusses several aspects related to the development of intelligent agents for playing OpenTTD. In section 4, we introduce *trAIns* and in section 5 we present the experimental results. Finally, Section 6 brings the conclusions and possibilities for future work.

2 OpenTTD

As mentioned, OpenTTD is an open-source clone of Transport Tycoon Deluxe, a game released in 1994. It was developed using reverse engineering, but now has an expressive number of new functionalities that improves the gameplay.

Players in OpenTTD own a transport company and are responsible for managing it. These players can be humans or computer controlled agents. The primary goal is to become the transport tycoon, that is, to make the controlled company becomes the best one. The criteria used to evaluate a company contemplates aspects like: the total number of vehicles and stations, the number of cargo types transported, the total number of stations and the amount of money the company has. A more general criterion can be the company net worth: the sum of available cash and company's properties value minus the total debts.

So, to be better evaluated on these criteria the company owner needs to build lucrative transport routes. The routes must connect cities and industries. There are some types of cargo (coal and wood, for example) that must be transported among industries. Others, like passengers and mail, must be carried from one city to another. Finally, some cargo must be transported from one industry to a city (examples are: goods and food).

Four different transport types can be used and combined to create routes: aerial, maritime, railroad and road (as shown in figure 1). Each one has pros and cons that vary according to different factors such as the amount of cargo that must be carried, the landscape around the industry/city, the distance between source and destination, the difficulty to construct the ways and stations, and the amount of money available.

The aerial transport, for example, requires a large area of flat ground for the construction of an airport capable of operating with large airplanes. If compared with the capacity of a train, each airplane can carry only a small amount of cargo and is also more expensive. However, there is no need to create paths connecting the airports. An already existing airport can easily receive new airplanes coming from anywhere, which is not always true for the other transport types. Also, as a general rule, airplanes are the fastest vehicles available in the game.

In contrast with the aerial transport, road transport has the cheapest vehicles of the game. They are also slower and able to carry only small amount of cargo. To reach some destination, vehicles must travel using the available roads. Therefore, the company owner that wants to use this kind of transport needs to build the roads connecting the route's source and destination. Road vehicles can automatically share the roads as they are two-way.

On the other hand, railroads are not able to share their tracks automatically. To create railroads capable of supporting more than one train running at the same time, the player needs to use a resource called *signal*. There are 6 different types of signals. The two most important are the block signal (allows only one train to be in the same block at the same time) and the path signal (allows more than one train to enter in a block if their paths do not intercept each other). These resources must be carefully used because misplaced signals and the use of a wrong signal type can cause deadlocks and accidents. That is why railroads are the most flexible transport type.

A carefully planned railroad can operate with dozens of trains and transport a large amount of cargo connecting distant points. The train cost is not too high if compared to other transport types because a single locomotive can drag several inexpensive wagons.

As in aerial transport, the construction of railroad stations demands a large area of flat land. It has also similarities with road transport: both must connect endpoints and offer tunnels and bridges as way to transpose obstacles. Thus, the construction of railroads incorporate almost all the challenges present in the construction of other transport types routes. That is why it has the most complex construction process.

Figure 2: *An OpenTTD screenshot: there two rectangles highlighting some tools available on game menu. The yellow tools predominantly have construction functions. By contrast, the green tools are essentially used for management tasks. It also shows the railway construction tools (the window positioned near the middle). The basic parts provided by the game for railroad track construction have been highlighted using the cyan color.*

2.1 Gameplay

OpenTTD is a construction and management simulator of transport routes. Thus, it has tools used to build the routes and other tools that enable the player to manage the created routes. Figure 2 shows an OpenTTD screenshot. The game menu is depicted at the top of the figure. Some tools have been highlighted. The yellow tools are used in route construction while the green tools are used mainly for management.

The main point of OpenTTD gameplay is that the player must build all elements related to the route. So, suppose that one player chooses to create a route using trains. The first step will be the selection of the industries and/or cites that will be connected by the route. There are some important criteria: production rate and distance between source and destination. After that, he needs to construct the stations and the tracks. Finally, he will buy the proper locomotive and wagons, and program them to execute the route. Other transport types have similar steps but the most powerful and complex is the railroad type.

Management tools are applied to manage the created routes throughout the game. During the game, new vehicle models will be released. All vehicles have some attributes: running cost, maximum reliability (determines the chances of a failure), maximum speed and capacity. Generally, new vehicles are better, since they are faster and are able to carry much more cargo. Therefore, vehicles need to be replaced during the game because they will become outdated. For railroads, it is also possible to change the type of the rail. There are four types available in the game: common, electrified, monorail and maglev. As these new rail types become available, it will be possible to use new kinds of vehicles. Thus, sometimes, changing the railway rail type can be lucrative.

The industry production rates tend to increase during the game. If this industry is used in a route, the number of vehicles must be revised to better transport the production. The production rate can also decrease, requiring a reduction in number of vehicles. It is also possible that another transport company decides to carry cargo from an already explored industry. The production will now be split between the companies forcing the old company to adjust the number of vehicles in its route.

To create the routes, it is important to understand the transported cargo payment mechanism. The payment received for delivering an amount of cargo to some industry depends on some factors. They are: the distance between source and destination, the type of cargo, the number of days that the cargo traveled and the amount of cargo

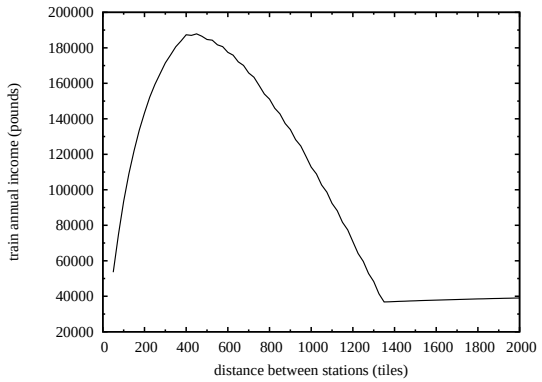


Figure 3: *This curve shows how the annual train income varies according to the distance not considering the inflation and supposing the train travels at constant speed. The used train has maximum speed of 112 km/h and the transported cargo was gold.*

delivered. So, the final payment equation is:

$$payment = dist \times time\_factor \times num\_pieces \times cargo\_base$$

Where $dist$ is the distance between stations, num_pieces is related with the amount of cargo delivered and $cargo\_base$ is the base price of a cargo unit. $Time\_factor$ represents the delay for transporting the cargo. It is inversely proportional to the number of days the cargo traveled. The rate of variation depends on cargo type.

Observing the equation, a trade off can be perceived: long distances will contribute for a bigger delivery payment. However, the delay will increase penalizing the time factor. Thus, there must be an optimal distance.

It is possible to estimate the annual train income for a specific cargo and distance. First, supposing the train travels at constant speed it is possible to estimate the travel time. The train has also a load and an unload time. With this information, one can compute the number of times the train travels the route in one year. Using this approximation we have computed the curve in the Figure 3 that shows approximately what the optimal distance is.

Observing the curve, it is possible to see that the optimal distance is close to 450 tiles (a tile is a cell in the discrete game map). A good AI should be able to build routes connecting industries separated by distances of this magnitude or even bigger distances, considering that trains will become faster during the game.

3 AI for OpenTTD

This section presents several aspects related to the development of an AI for OpenTTD. It first classifies the game environment using [Russell and Norvig 2003] taxonomy. This classification is important to better understand the challenges involved in programming OpenTTD agents. Then, NoAI - the OpenTTD API that enables the development of AI algorithms - is described. Finally, the already existent AIs are discussed.

3.1 Game Environment

Considering a company owner as an agent we can classify the environment where this agent will act. Using [Russell and Norvig 2003] taxonomy, it can be classified as fully observable, strategic, sequential, dynamic, discrete and competitive multiagent.

The game environment is fully observable because an agent using its sensors has access to the complete state of the environment at each point in time. It is capable of seeing all vehicles in the game (including other companies' vehicles), all routes and all existing industries. Moreover, an agent can see the plans being executed by other companies' vehicles. However, some of these characteristics are not implemented in NoAI API yet.

Considering only players' actions the environment is strategic, *i.e.*, the next game state is completely determined by the agent actions and the other players' actions (that can not be predicted). Although, the environment has also some stochastic characteristics mainly related with the economy. New industries can arise randomly. Their production rates changes are also determined randomly. The economy phase, contraction for example, is not predictable. The game disasters (UFO landing, for example), commonly disabled, are an environment stochastic element as well.

The OpenTTD game environment is sequential since current actions will affect future ones. For example, the construction of a railroad route influences future managing decisions regarding industry selection. It also generates new management tasks like: controlling the number of vehicles in the route and choosing the moment to change route vehicles models.

While a player is acting or deciding what he has to do, the game environment is changing: other players can act at the same time, changes such as the construction of a new industry can happen, etc. If a player is constructing a railroad to connect a specific industry, others can do the same and occasionally finish first. The environment, therefore, can be classified as dynamic. This dynamism will always generate some uncertainty in the planning and in its execution.

The game map is divided into small cells called tiles. These tiles are the smallest map units, *i.e.*, the construction of an element always demands at least one map tile. The time is also discrete. Internally it is represented using fractions of days (ticks).

As mentioned, there are various companies competing against each other. Hence, the environment is competitive multiagent. A company can collaborate with another by sending to it some money, but they still compete against each other for resources: industries, cities and map tiles.

Some of the environment characteristics increase the challenges involving the construction of an AI for the OpenTTD. The dynamic environment together with the presence of other agents demand from the AI the ability of fast planning. If planning takes too long, when it finishes, the current game state can be very different from the state initially considered in the planning. Thereby, it will not be valid anymore. Small differences can be resolved using a replanning that, again, needs to be fast.

The route planner needs to be fast while considering a lot of details available in the game such as the construction of bridges and tunnels; the use of terraforming (a tool that enables company owners to modify the land form); the route configuration (winding routes reduce the maximum allowed speed) and the large number of possible paths to connect two points.

Besides, the presence of other agents also demands some flexibility from the algorithms responsible for managing the routes. For example, consider that one company transports some cargo from one industry to a city. If another company decides to transport cargo from this same industry, the number of vehicles in the route should be decreased. Furthermore, variations on industry production rates are not deterministic. A route manager, thus, needs to be able to increase or decrease the number of vehicles in a route according to the changes in the production rates.

These are the main challenges that must be surpassed by the AI that will control a company in OpenTTD.

3.2 NoAI API

An agent has mechanism to perceive the environment (called sensors) and some tools that enable it to change this environment (called actuators). On OpenTTD, sensors and actuators are implemented as an API called NoAI Framework.

The NoAI API allows users to create AIs for the game, programming them in a script language called Squirrel [Demichelis 2009].

This is one of the many improvements introduced by OpenTTD community. Squirrel is an imperative object oriented script language strongly based on Lua language [Ierusalimschy et al. 1996]. This section will discuss some important characteristics of this API and how it works.

The primary principle used in the construction of NoAI Framework is fairness. That is, the actuators must correspond to the tools available for human players and the sensors must generate environment perceptions similar to human players'. As a consequence of this principle, the resources available in both interfaces (for the human players and for the computer controlled players) are equivalent.

To better understand the API internal functioning, one can consider each AI running in the game as a process in an operating system. Each AI will be scheduled using a round-robin scheduler and will be preempted after executing a certain number of instructions. Then, the game will execute its proper functions and call the AIs again, restarting the loop.

The API is composed by various classes that aggregate game functionalities related with some game resource. The class named *AIController* must be extend, so when the AI company is launched in the game, OpenTTD will call the method *Start*. This method must never return, if so the AI process will die.

The classes *AIAirport*, *AIRail*, *AIRoad* and *AIMarine* aggregate the basic functionality related with the four kinds of transport available in the game. For example, the *AIRail* class has a function named *BuildRailTrack(TileIndex tile , RailTrack rail_track)* used to construct a rail on a given tile. Another important function, used to create railroad station, is *BuildRailStation*.

To perceive the game environment one important class is the *AITile*. It has functions like *IsWaterTile*, *GetHeight* and *GetSlope* that can be used to get information about the map. It is important to observe here that when one of these functions is called, it will return a value based on the state of the map when it was executed. Therefore, if the game state changes after this, the AI will need to evoke the function again to note the difference.

The NoAI Framework also works with some events that are equivalent to the news published for human players during the game. When a new company is launched, when a new train becomes available or when a new industry is created the AI event stack will receive the proper information. These events are very important and can be used to help AI decisions process.

All the actuators implemented on the API inform if the corresponding action could be executed successfully. Thus, the AI programmer can always know when some action fails. One can execute an action in the test mode to check if it can be currently executed. Again, the returned value is associated with a specific game state.

3.3 Existing AIs

Currently, there are about 13 published AIs [OpenTTDForum 2009] available for OpenTTD. Most of them are very simple and just create straightforward routes using the aerial and/or the road transport type. Others, more sophisticated, are able to plan complex routes and even combine some transport types. However, only four of them work with trains, the most complex game transport type.

Generally speaking, all these four AIs have some problems. One of these problems, related with the pathfinding, is the limitation in the size of the planned routes. As shown, long routes can be very lucrative especially for trains. Some AIs can not deal with failures during the construction of a route. That is, if the game state changes during the planning the AI will fail in the construction of the route.

Other problem is the absence of rail type changes. This change is very important because it enables the use of new locomotives that are faster and more trustworthy. Differently from human players, none of these AIs is able to create double railways (some AIs create two independent railroad tracks that together compose the route). They neither are capable of making good decisions. Generally, they use poor algorithms to choose the locomotive engines and industries that must be connect by a route. In our tests, *trAIns* AI won easily from some of these AIs because they are very naive and lack powerful management resources. On some tests, they bankrupted on the first years of the game.

Figure 4: *The figure shows a route created by Admiral AI that operates with trains. To compose the two-way railway it uses two independent one-way railroad tracks.*

Among the existent AIs capable of playing with trains, the only one that could generate results similar to *trAIns* was Admiral AI (Admiral AI is able to play with all kinds of transport available in the game except the maritime transport type). It is the most powerful AI available for the game so far. It operates with trains creating routes which distance between source and destination is about 75 tiles. It is able to reuse already built stations, *i.e.*, Admiral AI can connect different source industries to the same destination industry. To be able to construct a two-way railway, it creates two independent railroads tracks that together compose the route (figure 4). Each independent railway is one-way. Thus, Admiral AI railway routes can operate with multiple trains.

Admiral AI is also able to manage the created routes. During the game it adjusts the number of vehicles in a route according to the production of the industry that is connected by the route. Throughout the game, Admiral AI replaces the locomotive types considering the new introduced types. Thus, we used Admiral AI to play against *trAIns* AI in the experiments presented in Section 5.

4 *trAIns* AI

Using the NoAI Framework and the Squirrel language we have developed a new AI for OpenTTD: *trAIns*. It is named *trAIns* because it only plays with trains, *i.e.*, it basically creates and manages train routes that connect industries. It works as follows: if there is some money available, it will decide if it should build a new route or spend the money improving already existent ones.

Route improvement has higher priority and includes increasing the number of trains in a route, changing the locomotive type or changing the rail type. If no route needs to be updated, *trAIns* will create a new route between two industries. Firstly, the source industry and destination industries are selected. If possible it tries to use an already existent destination industry. This is done sharing the same railway to multiple routes using a mechanism called junction. Then, the railways to connect the industries are created. To do this, it executes A* algorithm to find a path between the stations. We use an abstraction called double parts that enables the construction of double railways.

This section will describe in details each part of this process.

4.1 Railway Construction

One important component of *trAIns* AI is the module responsible for the construction of railways. *trAIns* builds only double railroad

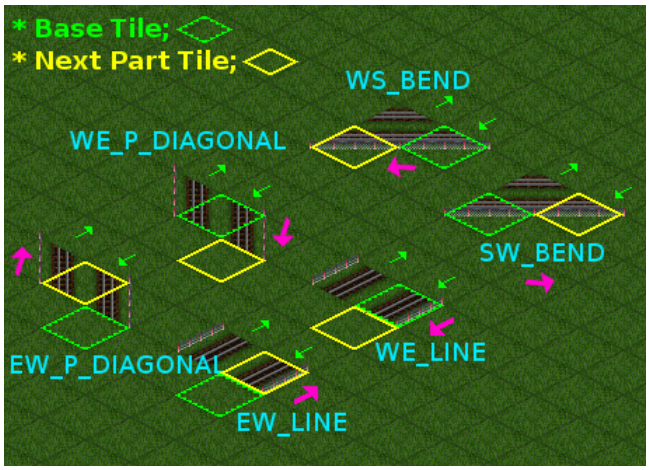


Figure 5: *This figure shows 6 double parts used to construct double railways. Each double part is created by the union of the basic parts provided by the game (figure 2). There are three basic kinds of double parts: bends, diagonals and lines. They have a direction and a orientation as shown in the figure. It also shows the base and next points. The base point is placed over the next point of a part to connect them.*

tracks, which enables many trains to circulate on the same route. As a result of the restriction that on each track of the railroad the train can travel only in one direction, the traffic on double railways is two-way. This approach minimizes traffic jams while allowing a large number of trains on the same railroad. To guarantee that trains will only be able to travel in one direction, each track is signaled using one-way block signals. Next, the process of constructing these railways will be detailed.

4.1.1 Double Railway Parts

OpenTTD provides four basic parts (figure 2) for the construction of railways. We have combined these basic parts to create an abstraction called double railway parts (figure 5). From now on, the term part will refer to a double part created by the combination of the basic parts provided by the game, unless the opposite is mentioned. These parts can be connected to create a double railway. There are 22 parts (bridges and tunnels are also considered parts) that have an orientation and a direction. The connection between parts has a restriction: some parts can not connect with others (*i.e.*, each part has a set of successors). Each part has two important points called *base* and *next*. The base point of a child part (successor part) must be placed over the next point of the parent part (predecessor part) to connect the parts.

4.1.2 Railway Planning

As mentioned, *trAIns* uses A* algorithm [Hart et al. 1968] to plan a path between source and destination points. A* is a classical path planning algorithm that is widely used in games to find shortest paths in a graph or grid. Basically, at each step, A* expands the node x that has the smaller cost $f(x)$ according to the equation $f(x) = g(x) + h(x)$, where $g(x)$ is the actual cost of moving from the source node to x and $h(x)$ is an heuristic function that estimates the cost between x and the destination node. If this heuristic is *admissible*, *i.e.*, if it never overestimates the actual cost, A* can be proved optimal. Details of A* can be found in [Russell and Norvig 2003] and [Bourg and Seemann 2004] among others.

To be able of performing an efficient planning of long railways, the original A* implementation provided as library in OpenTTD was modified. To save time and resources, we removed from the algorithm the code that updates the cost (g) of the nodes in the *open list*. That is, we assume that when a node is inserted in the *open list* it has already the minimum cost, *i.e.*, repeated nodes are visited in an increasing order of cost. Although we have not formally proved it, we believe that A* optimality is preserved, since, in our case, all edges have the same cost. So the first edges to be inserted will have the smallest values of g.

This modified implementation of A* tries to check, as early as possible, if a node has already been visited, that is, if it is already in the *closed list*. The original implementation postpones this verification, first generating the node and then checking if it is on the *closed list*. Moreover, as a consequence of the first change in the code, nodes are closed (marked as visited) in the moment they are inserted in the *open list*.

To create the double railways it considers double parts instead of the basic parts provided by the game during path computation (a node is considered a pair $< tile\ ,\ part >$). Hence, with this approach, *trAIns* needs to execute the search algorithm only one time to build a double railway while other AIs need two executions, each one creating an independent single railway in which trains move just in one direction.

As mentioned, one of the goals of this project is to develop an AI capable of building railways similar to the ones constructed by human players. Part of this goal is achieved by the use of double railroad tracks. However, the railway shape is also an important aspect that can determine the similarities of a human player's railway and a railway created by an AI. The railway tracing is important because it influences the acceleration of locomotives. Depending on the path configuration (number of curves), the train will be forced to decrease its speed. So, is important to try to minimize the number of curves. One possible approach to solve this problem is to punish each direction change using A* function cost (g) as suggested in [Rabin 2000]. But this can increase the time of execution since it augments the number of expanded nodes. The solution proposed here tries to avoid direction changes during tie-break procedures, as explained below.

To allow A* to efficiently plan long railways, it is necessary to carefully choose the heuristics since the number of expanded nodes in A* can be exponential in the length of the solution. The use of appropriate heuristics can attenuate this problem. The h function used by our algorithm is the diagonal distance. The traditional Manhattan Distance Heuristic, generally used in grid environments, is not admissible in our context since using double parts, we may have rails oriented diagonally. When some nodes have the same f value, we have to employ some tie-break procedures to chose which one will be expanded. In case of ties, our algorithm firstly chooses the node x that has the smallest $h(x)$, that is, the node that is supposedly closest to the goal. If all nodes have the same $h(x)$, the node that will minimize direction changes is selected. With this approach, the number of expanded nodes is reduced without sacrificing the solution quality.

Differently from common implementations, our A* implementation does not keep a field to indicate the parent node. Thus, when the execution finishes, if a path is found, the algorithm starts from the goal and chooses the successors with the lowest cost until it reaches the start node. If successors have the same cost it selects the successor that will not cause a direction change. When the start is found, there's a path where is possible (there is no guarantee since the game environment is dynamic) to construct a railway.

If during the railway construction - during the part construction at the position calculate by A* - a change is detected, *i.e.*, it is not possible to create a part at that point, it is necessary to replan the path. The last built parts are destroyed and A* is executed again changing only the start point, since the construction is done in direction to the goal.

The structure generated to represent the path is stored to be used in the future. It is important for changing the rail type and computing junction points.

4.1.3 Bridges

Sometimes, it is necessary to transpose an obstacle (a river, a road, another railway) during the construction of a railway. This can be done using bridges. In general, it is better to avoid bridges by bordering obstacles when possible, since bridges are more expensive (monetarily speaking). But this may significantly increase the num-

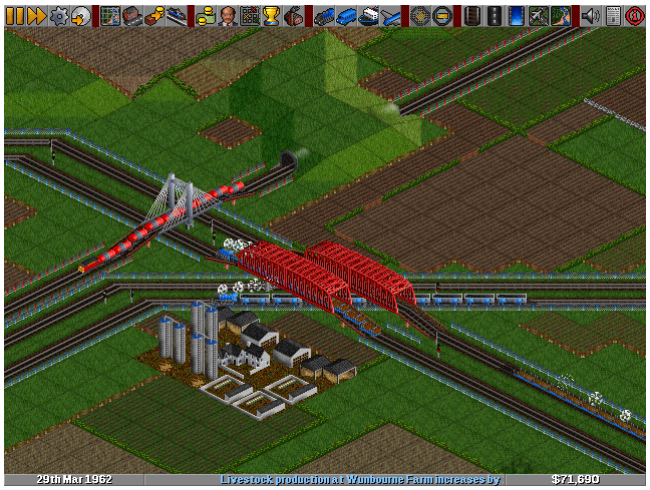


Figure 6: *The double railways were constructed by the trAIns AI. The bridge part was used to transpose other railway. As shown, the bridges that form a bridge part do not need to be aligned. The picture also shows a tunnel that can transpose obstacles too. trAIns AI does not work with tunnels.*

Figure 7: *This figure shows a junction used to create a branch on a railway connecting a Coal Mine to a Power Station. After the construction of the junction, there are two Coal Mines connected to one Power Station. It also shows that two trains can pass through the junction at the same time if their paths are independent,* i.e.*, they do not cross.*

ber of expanded nodes. To avoid this growth, our algorithm always considers bridge construction during path planning instead of trying to avoid the obstacles.

As the railway created by *trAIns* AI are double, bridges need to be double too. Another important restriction is related with the execution time. During the planning of a railway it is not possible to check for every node if a bridge should be build. Therefore, the approach adopted here limits the number of bridge construction tries. It only tries to create a bridge if it detects that there is an obstacle avoiding the construction of the next part. If so, it will try to find an ending point for the bridge that transpose the obstacle and permits the construction of a part after the bridge.

Bridges are modeled as parts. One bridge part can be composed of multiple bridges. Each track in the double part has independent size bridges, that is, the bridges do not need to be aligned in a part neither start and end on neighbor tiles. There is also the possibility of mixing bridges with common rails as shows figure 6. Besides saving processing time, this technique produces good results because the bridges built are very similar to the bridges created by human players.

4.1.4 Junctions

Junctions are important tools that allow the creation of complex railway networks, since they permit the creation of branches. Let's say, for example, that there is a route connecting two industries. Near the source industry there is another industry that produces the same type of cargo (that is, its production can be transported to the destination industry already connected by the route). Without the use of junctions, it would be necessary to create a new railway connecting the new industry to the destination industry. However, it is possible to use the already existing railway by connecting the new industry to it. This connection will be made using a junction as shown in figure 7.

The adoption of junctions on railways permits the concentration of production. The production of multiple source industries can be carried to a single industry. Some industries can only produce cargos by processing other industries production (a Sawmill, for example, uses wood to produce goods). If the production of various industries can be routed to a single processing industry its production rate will be very high. Thus, this industry will be an excellent candidate for a route.

Junctions are also modeled as double parts. There are 12 different junctions and each one can be placed over some specific parts. Thus, to create a branch at specific point of the route, one must choose the proper junction part that fits on the part at that point.

At the junctions, the tracks of the railway cross with each other. So, signals must be placed to avoid accidents. The signals used here are from the path signal type, which allows more than one train to enter in a block if their paths do not intercept themselves. Hence, it permits that two trains use the junction at the same time, in the best case. This approach attenuates traffic jams caused by the necessity of mutual exclusion on junctions.

4.2 Railroad Station Construction

Cargos are loaded and unloaded at stations, which should also be constructed by the company owner. These stations should be compatible with the double railways adopted. We selected a format in which there is a single entry point and a single exit point in each station. Each point connects with one of the tracks that compose the double railway. A station can operate with multiples trains at the same time as the number of platforms can be configured. We use two platforms for unloading station and one for the loading station. The number of unloading platforms is bigger to avoid traffic jams since multiple source industries can share the same destination industry.

To create a station, it is necessary to find a large area of flat land close to the desired industry (so far, *trAIns* only creates routes connecting industries). Unfortunately, there is rarely an available area satisfying these constraints so it is necessary to use the terraforming tool to flat the land. Since the cost of this operation can be very high, the approach adopted here is to test a number of different possible lands to then choose the least expensive. The number of tests is not high since there is a distance restriction between the station and the industry (if this distance is too large the station will not be associated with the industry).

4.3 Management

There are two primarily management tasks that should be treated by the AI: route management and the investment of the company's available money. The first task has higher priority: the AI will only invest the money in new routes if none of the current existing routes needs it. In this case, the last task is treated using a very simple approach: always invest (although some restrictions are adopted to avoid spending money on industries with very low production rates). Another important restriction is that *trAIns* AI must have a minimum amount of money before starting to construct any new route.

The process of creating a new route demands some decisions. One of them is the choice of which pair of industries will be connected

by the route. To solve this problem, *trAIns* AI considers some industry characteristics such as the number of stations around the industry, the cargo type and the amount of cargo already transported. Thus, the algorithm computes a ratio for each industry in the game and then chooses the industry with the highest ratio. This ratio is given by the production that is not already transport divided by the number of stations around the industry (at least one) and by the price paid for one unit of that cargo transported across 20 tiles with just 10 days of delay. That is, the *trAIns* AI tries to select the industry with the largest potential of money generation.

After choosing the source industry, it is necessary to find an industry to where the production must be transported. The optimal distance between two industries follows a curve similar to the one shown in figure 3, but can not be too long as time to plan the railway will increase too much. Firstly, it tries to use an industry that already has a route and railways. The condition to do this is that the distance between industries and the distance between the source industry and the closest part of the railway that forms the already existent route are limited for a given range. If so, a junction will be created. Otherwise, a new railway will be constructed.

Finally, the number of trains and the locomotive type must be chosen. As the number of trains in the route depends on the locomotive type, it is selected first. The selection process is also based on the computation of a ratio. For locomotives, this ratio varies from 0 to 1 and considers aspects such as the price, the maximum reliability, the maximum speed, its weight and power. All these variables are normalized according to the largest value available. Basically, the *trAIns* AI computes the financial cost of the locomotive benefits. All benefits have the same importance.

The created routes must be managed during the game. One of the managing tasks is the decision of the number of trains in a route. The process used to solve this problem estimates the load time and the travel time using the distance between stations, the production rate and the locomotive maximum speed. Thus, the number of trains is given by the load time plus the total travel time divided by the load time. This formula tries to guarantee that there will always be a train waiting to be load.

Another important management problem is the decision of when the locomotive type must be changed. This problem is also related with the rail type change. To solve these problems, *trAIns* calculates the same ratio used to decide with locomotive is the best. If the current locomotive is not the best it will be replaced. If the new locomotive can not operate on the current rails it must be changed as well. To avoid constant locomotive changes the AI only replaces locomotives in intervals of five years.

5 Experiments and Results

We performed some experiments to evaluate the proposed AI. These experiments compared *trAIns* AI with the Admiral AI playing with only trains. Fourteen scenarios have been used: half with flat terrain and the other half with mountainous terrains. All maps have 512x512 tiles, very small amount of seas (they are predomenantly are formed by lands) and the landscape style used was the *temperate* (the game has 4 different landscape styles). Games begin at the first day of 1960 and go for about 15 years. The locomotive failures were disabled and the other attributes were configured using the medium difficulty level. The games were executed in OpenTTD version 16724, available at the game SVN server.

Tables 1 and 2 present a summary of the results. Two metrics were used to compare the performance of the AIs: the company value and the detailed performance rating. The last considers different aspects such as the total number of vehicles, the total number of stations, the minimum and maximum incomes, the minimum profit, the number of cargo types transported and the total amount of money. The evaluation is based on some thresholds that must be reached by the company. Thus, if the company reaches all thresholds it will be evaluated with the maximum value: 1000 (figure 10 shows more details about this evaluation process).

The tables also present the number of routes created (a route is considered a pair of connected industries). They also exhibit the route

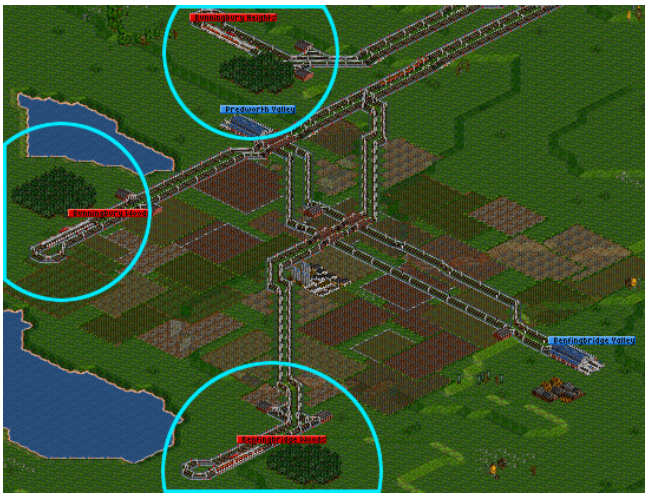


Figure 8: *All highlighted industries have their production transported to the same destination industry. That is, they share the destination station and part of the railway.*

Figure 9: *The figure shows an overview of the routes presented in figure 8. The source industries have been circulated and the destination industry is pointed by an arrow. It is also possible to see that the created railways have a small number of direction changes.*

average size that is computed using the Manhattan distance between the stations. Other selected fields are the total number of trains and the total number of stations.

Considering the company value as metric, *trAIns* AI defeated Admiral AI in all scenarios. On average, it reached a company value about eight times bigger than Admiral AI company value.

One of the causes for this success is the size of constructed routes. The routes created by our AI are longer than Admiral's as shown in the last column. They are also more lucrative. Table 3 shows the ratios: company value per trains and per routes. The *trAIns* AI routes are about 3 times more lucrative than Admiral AI routes. This reiterates the results shown in figure 3. Moreover, long railways demand a large number of trains per route because of the increase in travel time. That is, if the distance between stations increase, to keep the monthly transported cargo rate, it is necessary to operate with more trains on the route. On average, *trAIns* AI had about 2.66 trains per route against about 2 trains used by Admiral AI (table 3).

The use of junctions allowed the sharing of a large number of stations among routes. *trAIns* AI uses on average 1.44 stations per route against 1.86 used by Admiral AI. It also provided a mechanism to create railroad networks as shown in figures 8 and 9. During the experiments we observed that, in some situations, five different source industries shared the same destination industry.

When the terrain type is mountainous, it is more difficult to plan railways. On hilly terrains, the number of restrictions for path plan-

Figure 10: *This figure shows the companies detailed performance rating in match number 6 with flat terrain type. As mentioned, it is computed based on different criteria. It also shows the companies finances.*

AI	Performance rating	Company value (pounds)	# trains	# routes	# stations	Route average size (tiles)
Match 1, finished date: *8 - Jun - 1975.*						
Admiral	841	6,090,193	121	67	110	63.79
trAIns	837	46,124,685	221	81	112	193.04
Match 2, finished date: *1 - Feb - 1975.*						
Admiral	780	4,634,504	98	52	90	59.46
trAIns	786	18,437,790	103	43	65	192.74
Match 3, finished date: *11 - Jan - 1975 .*						
Admiral	775	8,272,364	175	84	129	67.01
trAIns	831	43,477,220	202	83	114	195.19
Match 4, finished date: *4 - Jan - 1975.*						
Admiral	598	3,729,935	79	44	79	65.43
trAIns	812	41,622,238	174	67	96	188.28
Match 5, finished date: *21 - Jan - 1975.*						
Admiral	595	4,073,063	92	51	91	62
trAIns	831	35,198,641	153	61	94	185.34
Match 6, finished date: *1 - Nov - 1975.*						
Admiral	716	4,955,126	97	49	93	61.25
trAIns	831	39,619,536	207	77	104	192.20
Match 7, finished date: *2 - Jan - 1975.*						
Admiral	801	6,591,956	120	62	107	66.50
trAIns	791	19,586,151	143	43	68	195.20

Table 1: *Admiral AI versus trAIns AI: played in scenarios with flat terrain type.*

AI	Performance rating	Company value (pounds)	# trains	# routes	# stations	Route median size (tiles)
Match 1, finished date: *1 - Jan - 1975*.						
Admiral	703	4,581,543	103	47	88	67.74
trAIns	796	23,004,009	123	50	72	179.74
Match 2, finished date: *23 - Set - 1975*.						
Admiral	133	29,700	9	4	10	69
trAIns	823	22,927,440	137	53	74	195.32
Match 3, finished date: *26 - Jan - 1975*.						
Admiral	716	5,408,988	120	65	110	62.35
trAIns	831	21,110,332	200	73	102	190.79
Match 4, finished date: *5 - Aug - 1975*.						
Admiral	439	2,280,652	48	23	46	62.82
trAIns	831	40,170,005	192	70	98	194.10
Match 5, finished date: *7 - May - 1975*.						
Admiral	285	3,030,691	24	14	28	72.28
trAIns	837	34,544,942	180	64	91	207.57
Match 6, finished date: *26 - Jul - 1975*.						
Admiral	332	1,283,846	29	15	27	60
trAIns	831	33,816,215	173	63	91	198.28
Match 7, finished date: *5 - Jan - 1975*.						
Admiral	468	1,462,148	34	16	34	61
trAIns	816	17,162,685	121	49	73	201.28

Table 2: *Admiral AI versus trAIns AI: played in scenarios with mountainous terrain type.*

AI	$\frac{\#\text{stations}}{\#\text{routes}}$	$\frac{\#\text{trains}}{\#\text{routes}}$	$\frac{\text{company value}}{\#\text{routes}}$	$\frac{\text{company value}}{\#\text{trains}}$	Company Value (pounds)	Relative company value
Scenarios with flat terrain type.						
Admiral	1.73	1.9	92,941	48,914	5,478,163	1
trAIns	1.45	2.67	527,191	200,083	34,866,609	6.36
Scenarios with mountainous terrain type.						
Admiral	2	2.02	97,247	50,560	2,582,510	1
trAIns	1.43	2.65	399,024	149,679	24,585,847	9.52
All scenarios.						
Admiral	1.86	1.96	95,094	49,737	4,030,336	1
trAIns	1.44	2.66	463,107	174,881	29,726,228	7.94

Table 3: *This table summarizes some statistics related with the matches presented in tables 1 and 2. The ratios that used the company value are in pounds.*

ning increases. The terrain landscape also causes a decrease in average locomotive acceleration. So, the total company value tends to reduce. As table 3 shows that Admiral AI was more affected by land format.

If the criterion used to compare the AIs is the performance rating, *trAIns* AI still defeats Admiral AI. The aspects evaluated by this criterion cover various game details. It is based on some thresholds and if the company reached the minimum values it will be ranked using the high value: 1000.

The railways created by *trAIns* AI do not have too many direction changes, in spite of the absence of punishment for direction changes. This is a very important result because enables the creation of large railways without degenerating their quality. Figure 9 shows a railway created by *trAIns* AI.

6 Conclusion

Artificial intelligence is one of the main components of a game and largely influences its quality. In OpenTTD, artificial intelligence algorithms are mainly responsible for controlling game agents, specifically, the companies controlled by the computer. These algorithms must generate actions and decisions so that agents behave similarly to the human players.

In this work, we presented *trAIns*, an AI for OpenTTD. The main motivation for its creation was the lack of good AIs capable of playing using trains. The existent AIs have some common problems: they can not deal with complex railroads, are not able to plan large railroads, can not change railroad track type, use poor algorithms to choose the locomotive engines and also construct very differently from human players. These problems affect the performance of the company controlled by the computer and also degrade game's quality.

trAIns AI presented some approaches to better deal with these problems. A careful implementation of A* search algorithm increased the performance without lowering solution quality. Despite the use of simple function costs, the generated railways do not have too many direction changes and are similar to human players' railways. Double railways enabled the use of various trains on the same route and with the adoption of the junctions the sharing of stations were also possible. Finally, the decision processes implemented in *trAIns* were able to satisfactorily manage the transport company during the entire game.

In the future, we intend to improve the AI decision and planning processes. We believe they can be refined with the adoption of optimization techniques. However, these techniques must be adapted to generate fast responses. Another way to upgrade these processes is the adoption of some techniques, like GOAP [Orkin 2004], commonly used in digital games.

The pathfinding algorithm can also be improved. There are some approaches capable of performing a replanning without the necessity of recomputing the whole solution. These approaches, for example [Koenig et al. 2004; Koenig and Likhachev 2002], can be adapted and used for programming an OpenTTD AI. Another possibility is the use of real time algorithms such as [Koenig and Likhachev 2006] to generate fast solution for the game.

Finally, there are some other game resources that still can be implemented in the AI. The use of tunnels is one of them. Like bridges, they can transpose some kinds of obstacles and are important in the game. Another important resource is the adoption of terraforming to decrease the number of curves and altitude changes.

Acknowledgments

The authors would like to thank the financial support provided by CNPq and Fapemig in the development of this work.

References

BOURG, D., AND SEEMANN, G. 2004. *AI for Game Developers*. O'Reilly Media, Inc., July.

BYL, P. B.-D. 2004. *Programming Believable Characters for Computer Games (Game Development Series)*. Charles River Media, Inc., Rockland, MA, USA.

DEMICHELIS, A., 2009. Squirrel. http://squirrel-lang.org/default.aspx, June.

HART, P. E., NILSSON, N. J., AND RAPHAEL, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on 4*, 2, 100–107.

IERUSALIMSCHY, R., DE FIGUEIREDO, L. H., HENRIQUE, L., WALDEMAR, F., AND FILHO, W. C., 1996. Lua - an extensible extension language.

KOENIG, S., AND LIKHACHEV, M. 2002. D*lite. In *Eighteenth national conference on Artificial intelligence*, American Association for Artificial Intelligence, Menlo Park, CA, USA, 476–483.

KOENIG, S., AND LIKHACHEV, M. 2006. Real-time adaptive a*. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, ACM, New York, NY, USA, 281–288.

KOENIG, S., LIKHACHEV, M., AND FURCY, D. 2004. Lifelong planning a*. *Artif. Intell. 155*, 1-2, 93–146.

OPENTTD, 2009. Openttd. http://www.openttd.org/en/, June.

OPENTTDFORUM, 2009. Transport tycoon forums - noai discussion. http://www.tt-forums.net/viewforum.php?f=65, June.

ORKIN, J. 2004. Applying goal-oriented action planning to games. *Game Programming Gems*, 217–228.

RABIN, S. 2000. A* aesthetic optimizations. *Game Programming Gems*, 264–271.

ROLLINGS, A., AND ADAMS, E. 2003. *Andrew Rollings and Ernest Adams on Game Design*. New Riders Publishing, May.

RUSSELL, S. J., AND NORVIG, P. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education.

VhCVE: A Collaborative Virtual Environment Including Facial Animation and Computer Vision

Henry Braun, Rafael Hocevar, Rossana B. Queiroz, Marcelo Cohen, Juliano Lucas Moreira, Julio C. Jacques Júnior, Adriana Braun, Soraia R. Musse, and Ramin Samadani*

Graduate Programme in Computer Science
PUCRS - Av. Ipiranga, 6681, Porto Alegre, RS, Brazil

*Hewlett Packard Laboratories
1501 Page Mill Rd., Palo Alto, CA, USA

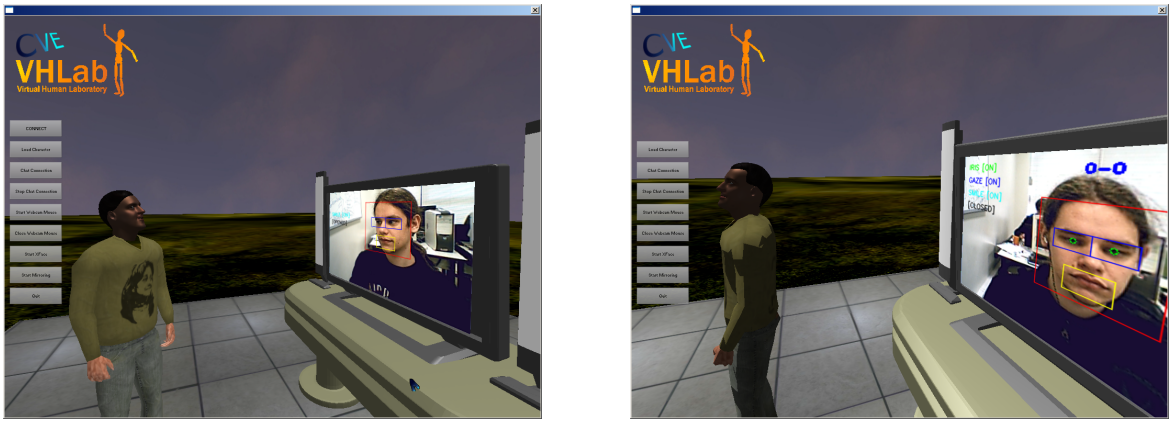


Figure 1: *Snapshpts from VhCVE platform.*

Abstract

In this paper we present a platform called VhCVE, in which relevant issues related to Collaborative Virtual Environments applications are integrated. The main goal is to provide a framework where participants can interact with others by voice and chat. Also, manipulation tools such as a mouse using Computer Vision and Physics are included, as well as rendering techniques (e.g. light sources, shadows and weather effects). In addition, avatar animation in terms of face and body motion is provided. Results indicate that our platform can be used as a interactive virtual world to help communication among people.

Keywords:: CVE, Facial Animation, Avatars, Computer Vision

Author's Contact:

{henry.braun, rafael.hocevar}@cpph.pucrs.br
{rossana.queiroz, marcelo.cohen,
julio.silveira,soraia.musse,adriana.braun}@pucrs.br
ramin.samadani@hp.com

1 Introduction

Many years ago, at the beginning of Virtual Reality (VR) area, people expected to have a new interface where they could interact with others, in a parallel world. The application which probably summarizes this expectation is the CVE (Collaborative Virtual Environment). CVEs are three-dimensional computer-generated environments where users are represented by avatars and can navigate and interact in real-time, independently of their physical location [Frécon 2004].

In CVEs, VR and Computer Graphics (CG) technologies are used to immerse multiple individuals in a single shared space. Such environments support a range of activities, e.g. virtual conferencing [Frécon and Nöu 1998] and games [1]. Although new technologies are available, CVEs still present relevant challenges, such as human-computer interaction in the virtual world, real-time rendering and animation, how to control and interact with avatars, among others.

In our platform we use OpenGL for low level rendering process (such as facial animation) and the graphics engine Irrlicht [2] for shading and other functions in the graphics pipeline. The advantage of using a graphics engine for rendering is the optimization and the faster coding. Other aspects are also important in CVEs, such as:

- Avatars animation (body and face);
- voice tools;
- chat tools;
- functions to provide object interaction (Physics);
- human-computer interaction based on Computer Vision (CV);
- real-time frame rates, among others.

In this work, we propose a modular approach, i.e. a number of separate modules are composed together in order to provide the CVE functionalities. The advantage is the possibility of integrating several toolkits, by combining their output in the same application. Our platform helps to create an application for real-time visualization of virtual humans, allowing the best possible interaction among connected people. The CVE is called *Virtual Humans Collaborative Virtual Environment*-(VhCVE).

The paper is organized as follows: related works are described in the next Section, followed by an overview of the VhCVE architecture and its key components. Section 4 presents some results obtained and, finally, section 5 describes the ideas for new components and final remarks.

[1] http://secondlife.com/

[2] http://irrlicht.sourceforge.net/

2 Related Work

Nowadays, a rather well known CVE is Second Life. It offers the exploration of a 3D virtual world, allowing the user to create his/her own avatar, as well as walking and flying to different 3D environments. In this system, it is also possible to use voice and text communication for more realistic human interactions. Other well known CVE is Playstation Home [3]: integrated with every PS3 system, it brings a large community of users allowing them to chat, walk around, see movies, customize their avatars and play games. In the Home world it is possible to see several advertisements such as movie trailers and game releases.

Few surveys have been published in literature about CVEs. In 2004, Emmanuel Frécon [Frécon 2004] presents a very complete overview of CVEs since 1990s. Also, Frécon discusses proposed standards as well as system trends. In 2008, Wright [Wright and Madey 2008] describes APIs, frameworks and platforms used to build CVEs.

The report by [Wright and Madey 2009] led to another publication. The authors propose the survey with two main goals: first, to concisely review several prominent, active desktop-VR technologies, and, second, to recommend the technology or technologies most well suited to building a CVE.

According to the classification proposed in [Wright and Madey 2009], in this paper we describe a CVE platform which uses Irrlicht, OpenCV, OpenGL, and other components, as later described.

3 CVE Architecture

VhCVE performs real-time visualization of virtual humans in 3D environments using a set of libraries and toolkits, where most of them are open source. Figure 2 illustrates architecture components and subcomponents. The components are the main modules in the architecture: Application Manager, Core and State Manager, while subcomponents are related to specific purposes. All subcomponents have an initialization process and most of them contain an update function, e.g. Webcam and Network Managers which are related with image capturing and packet communication, respectively.

The visualization is performed using Irrlicht rendering engine. This engine includes important functionalities and interfaces for communication among components. Irrlicht is an open source engine, and uses either DirectX or OpenGL in order to create 3D scenarios and animations. Other components are described next. One possible output of CVE is the **Video Recorder**, responsible for calculating the elapsed time between each frame and grab a screenshot of the scene. The files can be compressed and filtered using any video processing software. Next Section presents components and subcomponents of VhCVE, while in Sections 3.2 and 3.3 Computer Vision and Facial Animation features are described.

3.1 Components and Subcomponents

There are three main components which are responsible for executing and managing the subcomponents. The **Core** component coordinates the subcomponents initialization and it is also responsible for the display functions. The **State Manager** defines the finite state machine used to control VhCVE, sending events/request to be treated by the **Application Manager**. This last component send tasks to be executed into the subcomponents, as required. In next sections we present further details about subcomponents organized in three main topics: Interfaces, Characters and Visualization.

3.1.1 Interfaces

In DirectX or OpenGL environments, the Graphics Processing Unit (GPU) hardware is represented by a software entity called *device*. The **Device Manager** allows to access the required device from any other subcomponent. One of integrated devices aims to perform physic calculations. Physics is an important part of games, for this

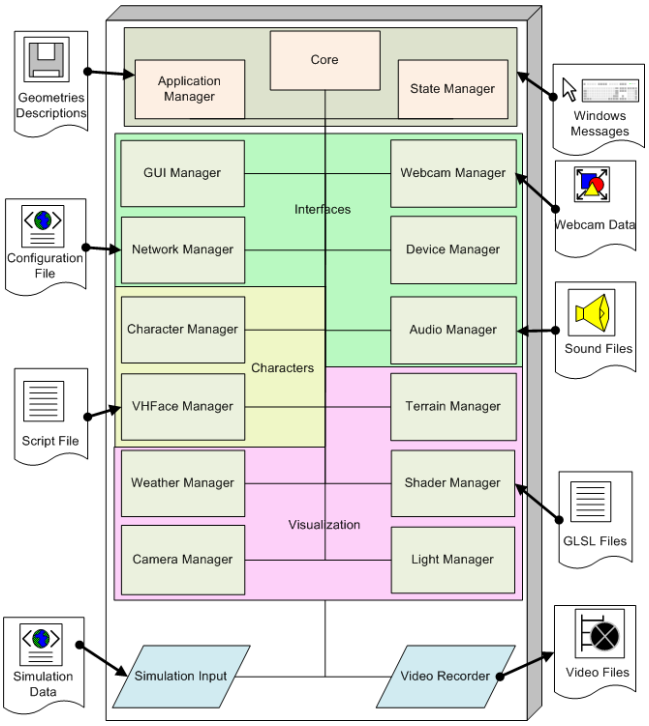


Figure 2: *CVE architecture overview.*

reason we use a library called IrrPhysx [4], which is a PhysX wrapper for the Irrlicht graphics engine.The main idea of this wrapper is to abstract the PhysX SDK methods from the user. Instead, we just have to use a simple interface into Irrlicht. Using PhysX, we are able to simulate the behavior of several kinds of objects like rigid and articulated bodies, cloths, fluids and terrains, which can be interesting in CVE applications. Figure 3 shows an example of rigid bodies with animation.

Figure 3: *Rigid bodies with animation.*

VhCVE has a simple graphics user interface (GUI) allowing access to the framework functionalities in an easy and fast way. The **GUI Manager** subcomponent is based on Irrlicht GUI toolkit classes, and also is responsible for creating and updating GUI components such as labels and buttons.

To perform the Internet connection and voice over IP communication there is a component called **Network Manager**, which uses an input file containing the server IP and port for connection (it is used in the CVE initialization). RakNet is a cross-plataform C++ game network engine [5]. This network engine is easily integrated with Irrlicht. RakNet is responsible for creating the peers, receiving and sending data packets to the server. This server keeps receiving the players positions, rotations, text messages (illustrated in Figure 4)

[3] http://www.us.playstation.com/PS3

[4] http://chris.j.mash.googlepages.com/irrphysx

[5] http://www.jenkinssoftware.com/

and facial status. After that, it sends all the data to the players in order to provide visualization.

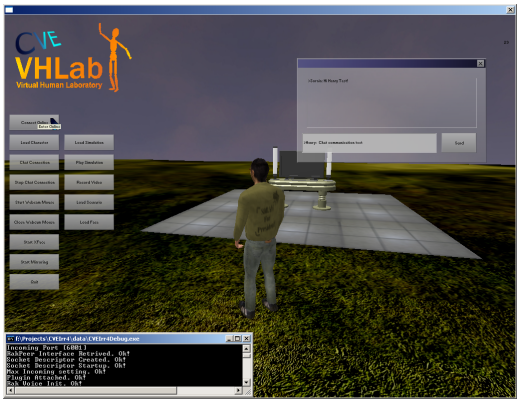


Figure 4: *Chat communication.*

The RakVoice toolkit is a feature of RakNet. This toolkit allows voice communication in real-time among connected users into the chat room. The RakVoice is attached into the RakNet device, after that it is necessary to specify the size of the audio that will be encoded at a time. RakVoice provides the means to encode and decode sound data. In order to listen to the sound, a sound engine is required to provide other audio features such as mute, play and stop. Every time a user speaks in the microphone the audio is buffered and sent to the server, which in turn is responsible for sending it to the other players. Our framework is not capable yet to treat avatars proximity in terms of spatial sound, so in this case the voice communication works for everybody online in the room.

To handle sound output, Irrlicht allows the use of a library called *Ambiera IrrKlang Audio Library*[6]. The **Audio Manager** performs the environment audio and provide support to different audio file formats such as MP3, OGG and WAV. This is the default library for sound output enabling stereo and 3D sound sources. It is important that CVEs support full 3D environmental sounds in a intelligent way. Every sound emitting entity must have a radius distance attached, determining the reachability and the volume of the emitted sounds in order to prevent mixing of distant sounds, hence producing unrealistic environments.

Any webcam can be used to capture images. Then the **Webcam Manager** handle these images as frames and process them using OpenCV allowing to create applications such as CVMouse and Virtual Mirror, explained in Sections 3.2.1 and 4.

3.1.2 Characters

The virtual characters are structured in four parts: body model, bones, animations and facial animation. The **Character Manager** manages this structure in memory and it is in charge of all actions related to characters along the simulation, except the facial animation. The body animation is executed through Irrlicht *AnimatedMeshSceneNodes*. The bone system allows attaching objects to the characters, such as a briefcase or a hat. The facial animation uses other subcomponent called **VHFace Manager** to handle the character's facial expressions. Further details are explained in Section 3.3.

The **Simulation Input** is a module responsible for integrating our CVE with external simulators. It receives the simulation data file and it is responsible for reading and parsing it. This file has XML format and defines a simulated scenario containing positions of virtual characters, among other data. With this feature it is possible to have NPC (Non Player Characters) interacting into the CVE world, e.g. crowd simulation [Musse and Thalmann 2001].

[6] http://www.ambiera.com/irrklang/

3.1.3 Visualization

The **Light Manager** allows determining how the environment illumination should be. It is possible to add and modify light sources customizing the scene and increasing the environment realism. The shadows created by the light sources are also controlled by this component.

In order to increase the graphic quality and realism of the visualization, we can use the **Weather Manager**. This component is capable of creating weather effects such as fog, rain, snow and clear weather. For better visualization results the **Shader Manager** allows the use of different rendering techniques e.g. bump mapping, cartoon shader (illustrated in Figure 5). These shaders are written in OpenGL Shading Language (GLSL) [Rost 2005].

Figure 5: *Cartoon shader example.*

For terrain generation we can use the **Terrain Manager** subcomponent, built upon Irrlicht *TerrainSceneNodes*. This subcomponent manage terrains created by a height map input and also integrate the terrain mesh with PhysX.

The **Camera Manager** subcomponent allows the user to explore the scene in a practical way. It provides three types of camera navigation, also making it possible to change navigation styles at any point of the application. The possible styles are: i) FPV (First person view), allowing to navigate like a first person point of view; ii) TPV (Third person view) aims the camera at the main avatar; and SCV (Static camera view), which fixes the camera, enabling the mouse to control the framework menus.

3.2 OpenCV

OpenCV [Bradski and Kaehler 2008] is largely used by scientific communities for facial tracking among other applications. In this work we propose to integrate Irrlicht and OpenCV in order to provide interaction tools based on computer vision algorithms. Indeed, this integration is very easy since OpenCV and Irrlicht has similar "frame loops", providing a solution without extra callbacks implementation. The image captured through OpenCV can be dealt in Irrlicht display loop, eliminating the need of a new thread or callback function.

3.2.1 CVMouse

The objective of *CVMouse* is to work as a pointer/scratch interface improving the human-computer interaction. Using a webcam and computer vision algorithms, *CVMouse* is able to simulate the states of the mouse. The idea is to track a learned-based color distribution and generate a mouse state, based on previous information.

The *CVMouse* states are, but not limited to: (i) capturing mouse position; (ii) clicking the mouse with the left button; (iii) clicking and holding the left button; (iv) releasing the left button. Basically only the first two stages could be used in machine-interaction applications, but the following ones are implemented with the objective of using them in drawing applications, such as a pencil in a painting system (or to drag and hold things).

In this work we use the YCbCr color space as color representation, but probably many other color spaces could be used, such as HSV or Lab. The idea is to use a color invariant feature to be more robust with illumination changes. Another used information is the area of the tracked object. This can be used to estimate the distance to the

object from the camera. With the information of the position of the object through time and their distance from the camera, we can simulate the states of a virtual mouse.

To learn the color distribution, we fix a colored object in a specific region, as shown in Figure 6, and for the channels Cb and Cr we compute their median value and standard deviations. We do not use the 'Y' information, with the objective to be lighting invariant. It is important to notice that the color of the adopted object must not be visible in any other location of the scene, because the tracking system will track the largest object with the detected color.

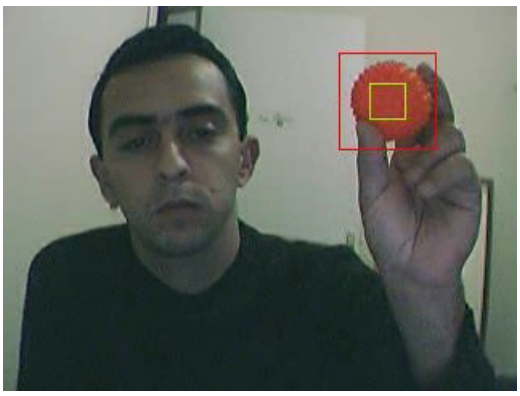


Figure 6: *Learning color distribution.*

Creating a color model Let S be a vector (with size = n) composed by the pixel values captured over the predefined region, in the Cb channel. Firstly we compute their median and standard deviation values (m and σ, respectivally).

Before create a color model, we remove some outliers, in a simple way, as described in Eq. 1:

$$S_{new(i)} = \begin{cases} S_j & \text{if } \|S_j - m| \leq 2 \cdot \sigma, \text{ (for } j = 1 \text{ to } n) \\ & \text{else} \end{cases}, \quad (1)$$

then we recompute the standard deviation (σ) over the new vector S_{new}. Thus, we can create a more robust color model. We apply the same method to the Cr color channel. After the outliers removal approach, our color model is composed of $C(m_1, \sigma_1, m_2, \sigma_2)$, where m_1 is the median of channel 1 and σ_1 is their standard deviation (respectively for the second channel).

Background subtraction For each pixel (x, y) in the whole image captured from the camera, we compute the absolute difference over the median values (for the channels Cb and Cr). We define the foreground pixels as the ones whose difference are lower than a predefined threshold ($K = 6$, obtained by experiments), as can be seen in equation 2:

$$B_{Cb(x,y)} = \begin{cases} 1 & \text{if } \|Cb(x, y) - m_1\| \leq K \cdot \sigma_1 \\ 0 & \text{else} \end{cases}. \quad (2)$$

The final binary image is composed by an AND operator over the two channels (B_{Cb} and B_{Cr}).

Some morphological operators (closing and opening) are used to eliminate small artifacts and to close small holes in the resulting binary image.

States definition To simulate the states of the mouse, we monitor the position of the tracked object in time and their size. In our application we use the last N frames ($N = 30$, obtained by experiments) as mouse tail, to generate the states of the simulated mouse. In a general way, each state is described as follows:

- state (i) capturing mouse position: the centroid of the biggest object (after the background removal) is defined as mouse position in each frame;
- state (ii) clicking the mouse with the left button: if, in the last N frames, the position of the mouse do not change very much (it means that it stop to move) and its area increases at a predefined threshold (T_c) and after decrease approximately to the initial area, we assume that the mouse was clicked ($T_c = 3\times$ initial area, obtained by experiments);
- state (iii) clicking and holding the left button: if, in the last N frames, the position of the mouse do not change very much (it means that it stop to move) and its area exceeds a predefined threshold (T_c) and remains, we assume that the mouse was clicked and hold;
- state (iv) releasing the left button: the same as state (ii).

When the user simulates the click of the mouse with the *CVMouse*, the position of the cursor usually changes a lot. To prevent that the user clicks in erroneous positions, we fix the last detected position of the object after the system detect that the object stopped to move. In this way, the user can click in a very specific location. Figure 7 shows an example of object tracking.

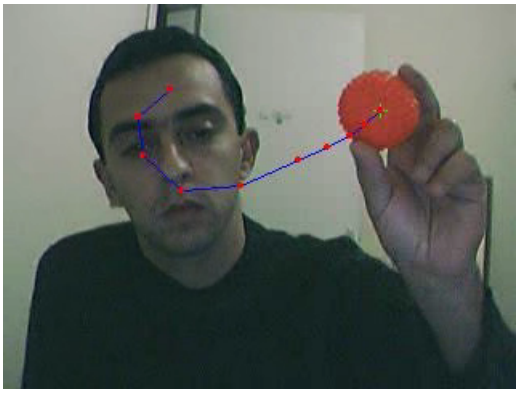


Figure 7: *CVMouse tracking an object.*

3.3 VHFace - Facial Animation

Another challenge in the project of a CVE is to provide real-time generated facial animation to the avatars. The facial animation module, called VHFace Manager, integrates the framework proposed by Queiroz et al. [Queiroz et al. 2009], based on XFace core libraries [Balci 2004]. The framework follows the MPEG-4 Facial Animation (FA) standard [Pandzic and Forchheimer 2003] for parameterization and animation of faces. The standard specifies a set of 84 feature points (FPs) located on the 3D face mesh. A subset of them acts as control points for the 68 Facial Animation Parameters (FAPs), also defined in the standard.

The two first FAPs describe high-level actions (6 facial expressions and 14 visemes) and the remaining deal with specific regions of the face, describing low-level actions, such as "raise left cornerlip" and "close top left eyelid". The FAPs are encoding as numerical values, which are measured by a set distances of key-features of the face, called FAPU (Facial Animation Parameter Units).

A FAP-based animation provides, for each animation frame, the variation of the FAP values. Thus, for each frame, we have a stream of these values. In order to optimize the sending of these streams through the network, a bit mask is also sent indicating which of the 68 FAPs are active (i.e. had values changed) in the frame.

Having the FAP stream, it is necessary to deform the face skin vertexes to produce the animation. Each FAP acts over one FP and its neighborhood (influence zone) vertexes, producing a deformation in the mesh. This means that each FAP value is scaled by its FAPU to provide the displacement of the FP vertex and the vertexes of its influence zone can be deformed through the application of different functions, such as cosine [Balci 2004] and radial basis [yong

Noh et al. 2000; Wu et al. 2006] functions. The information about the FAPU, FPs and their influence zones of a 3D mesh (the Face Definition Parameters – FDP) are described in files called FDP.

In this context, the VHFaceManager has two main modules (see Figure 8):

- The CV2FAP/FDL2FAP module, which receives as input Computer Vision data or high-level face actions and maps it to FAPs, according to Queiroz et al. [Queiroz et al. 2009] methodology. The high-level face actions include facial expressions, lip synchronization and eye behaviors, using the FDL script language (Facial Description Language), also described in [Queiroz et al. 2009].
- The FAP2MeshDeformation module, which receives the FAP streams provided by the CV2FAP/FDL2FAP module or received by the network and performs the deformation in the 3D mesh, according to the FDP model data.

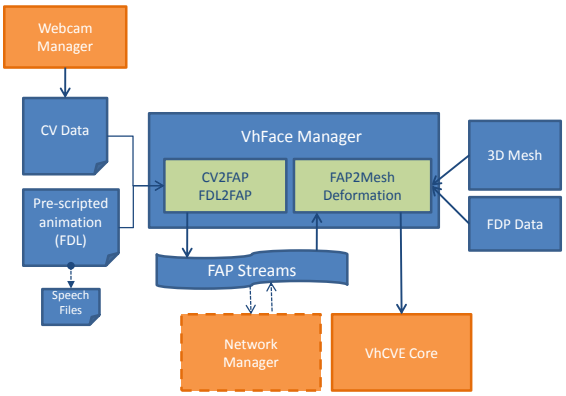


Figure 8: *Diagram of the VhFaceManager Architecture*

Note that FAP values are independent of the 3D model. These values are scaled by the FAPU, providing the displacement of the affected FPs. The influence zone of each FP is affected by a deformation function, also according to the FAP direction. So, a FAP-based animation can be performed by different face models.

4 Results

This section presents some preliminary results aiming to explore the features robustness of the proposed framework. Figure 9 shows an application in a city environment which allows the user to explore it with an avatar.

Figure 9: *VhCVE system display.*

Using the CVMouse model presented in Subsection 3.2.1, we built an application that allows the user to draw in the screen by moving a colored real object, as showed in Figure 10.

We also implemented an application called "Virtual Mirror", in which the characters "mimic" the facial movements of the user. It explores the functionalities of the VHFace and Webcam Manager modules, using facial feature detector algorithms (for face, eye and mouth regions) from OpenCV [Viola and Jones 2001; Castrillón et al. 2007]. The mapping of the detected features to simple events (opened mouth, closed mouth, smile and direction of the horizontal gaze) and then to FAP animations is implemented according to the methodology of Queiroz et al. [Queiroz et al. 2009]. Once information from face components is acquired, it is used to animate the avatar's face, as shown in Figures 1 and 11.

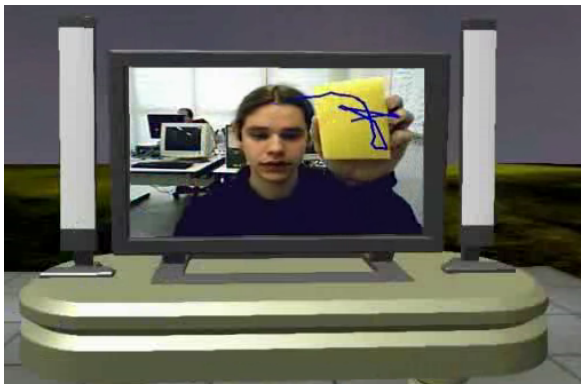


Figure 10: *Illustration of CVMouse-based application.*

Figure 11: *Illustration of Virtual Mirror application, integrating OpenCV and VHFace.*

As a result of preliminary performance tests, the framework bottleneck is visible when rendering more than 110 characters. The virtual human animation update is bound to the computer processing unit (CPU), requiring a large amount of CPU time. On the other hand, with static virtual humans (without body animations), the amount of rendered characters has increased to 360 while keeping a frame rate of 24 frames per second. Each virtual human contains 3686 triangles. The results were obtained executing VhCVE on a Intel E8500 Core 2 Duo, 4GB RAM DDR2, equipped with two GeForce 8800GTS OC 320MB in SLI mode.

5 Final Remarks

This paper presented a framework for collaborative virtual environments integrating several toolkits and computer vision algorithms. Moreover, it is possible to use the framework physics and network features to create different styles of multiplayer games, such as first person shooters or online role playing games.

The obtained results were acquired with the current VhCVE version. For the next version, we plan to include new techniques in order to improve the graphics quality and to increase the number of characters rendered in real-time. For instance, new shaders could be used to provide better lighting and also to offer new effects such as realistic water and fire. Also different techniques for shadow creation and weather effects could be implemented. Finally, the use of LOD (level-of-detail) and culling techniques, combined with impostors and rendering acceleration techniques [Ciechomski 2006], could increase the amount of characters rendered in real-time.

References

BALCI, K. 2004. Xface: Mpeg-4 based open source toolkit for 3d facial animation. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, ACM Press, New York, NY, USA, 399–402.

BRADSKI, D. G. R., AND KAEHLER, A. 2008. *Learning opencv, 1st edition*. O'Reilly Media, Inc.

CASTRILLÓN, M., DÉNIZ, O., GUERRA, C., AND HERNÁNDEZ, M. 2007. Encara2: Real-time detection of multiple faces at different resolutions in video streams. *J. Vis. Comun. Image Represent. 18*, 2, 130–140.

CIECHOMSKI, P. S. D. H. 2006. *Rendering massive real-time crowds*. PhD thesis, EPFL.

FRÉCON, E., AND NÖU, A. A. 1998. Building distributed virtual environments to support collaborative work. In *VRST '98: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 105–113.

FRÉCON, E. 2004. *A Survey of CVE Technologies and Systems*. SICS Technical Report T2004:03. Swedish Institute of Computer Science.

MUSSE, S., AND THALMANN, D. 2001. Hierarchical model for real time simulation of virtual human crowds. *IEEE Transactions on Visualization and Computer Graphics 7*, 2, 152–164.

PANDZIC, I. S., AND FORCHHEIMER, R., Eds. 2003. *MPEG-4 Facial Animation: The Standard, Implementation and Applications*. John Wiley & Sons, Inc., New York, NY, USA.

QUEIROZ, R. B., COHEN, M., AND MUSSE, S. R. 2009. An extensible framework for interactive facial animation with facial expressions, lip synchronization and eye behavior. *Comput. Entertain. (to appear)*.

ROST, R. J. 2005. *OpenGL(R) Shading Language (2nd Edition)*. Addison-Wesley Professional.

VIOLA, P., AND JONES, M. 2001. Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on 1*, I–511–I–518 vol.1.

WRIGHT, T. E., AND MADEY, G. 2008. A survey of collaborative virtual environment technologies. Tech. rep., University of Notre Dame - USA.

WRIGHT, T. E., AND MADEY, G. 2009. A survey of technologies for building collaborative virtual environments. *The International Journal of Virtual Reality 8*, 1, 53–66.

WU, Z., ZHANG, S., CAI, L., AND MENG, H. M. 2006. Real-time synthesis of chinese visual speech and facial expressions using mpeg-4 fap features in a three-dimensional avatar. In *INTERSPEECH-2006*.

YONG NOH, J., FIDALEO, D., AND NEUMANN, U. 2000. Animated deformations with radial basis functions. In *VRST '00: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 166–174.