An architecture for real time crowd simulation using multiple GPUs

Mark Joselli
Pontifícia Universidade Católica do Paraná
PUC-PR

José Ricardo da S. Junior and Esteban Clua
Media Lab
Universidade Federal Fluminense

Figure 1: *Real time crowd simulation*

Abstract

Computing and presenting crowd simulation in real-time requires an intensive processing effort, since it is necessary processing the behavior and render of each entity. The advent of GPU computing has enabled the development of many strategies for accelerating these simulations. In this paper we propose an architecture for multiples GPUs for crowd simulation, that allows a massive number of entities to be processed and rendered in real time. Also, we implement a representative case-study based on the behavior of a crowd during the street carnival from Rio de Janeiro from which we run benchmarks and compare the benefits achieved using more the presented architecture.

Keywords:: CUDA, GPU Computing, Crowd Simulation, Flocking Boids, Multi GPU

Author's Contact:

mark.joselli@pucpr.br
{jricardo, esteban}@ic.uff.br

1 Introduction

Realism in video-games is not only a matter of perfect graphics, but also includes the search for real behaviors and physics [Joselli et al. 2012a]. In a typical natural environments it is common to find a huge number of entities, such as plants, dynamic particles, animals and even people. It is also the case for other densely populated systems that appear in games like sport arenas, soldiers in a battle and people wandering in a city. Recently, it is being common for games to be set in open worlds, requiring the simulation of a living organisms in order for them to be as more immersive as possible. Games that need to present any of these systems usually have a very limited number of independent entities, mostly of them having a predictable behavior to allow its processing in real time.

Crowd simulation are becoming frequent on computer games, like Gran Theft Auto V [Rockstar 2013], and digital films, such as The Lord of the Rings [Aitken et al. 2004] trilogy. Typical examples of crowd simulation usage are in the simulation of the behavior of herbs of animals [Reynolds 1987a], people walking on the street [van den Berg et al. 2008], soldiers fighting in a battle [Jin et al. 2007] and spectators watching a performance [nVidia 2008].

In order to achieve a real immersion, the behavior must mimic the one observed in a real crowd, achieving also a real time frame rates. Obtaining such results requires dealing with processing steering, interaction and behavior of entities between different objects. Additionally, the rendering process requires drawing a huge number of entities, consuming even more computation time. Many approaches employed for performing crowd simulation are very simple, capable of dealing only with fewer entities for video-games, since the crowd simulation is only one of the many tasks that must be handled during a complete simulation.

Processing and rendering large scale crowds in real time demands a lot of processing power. PC clusters can be used as an alternative to alleviate this processing. Unfortunately the communication efficiency and maintainability of this solution becomes the bottleneck. In contrast, modern GPUs are accessible by a wide range of people and have a high processing power in relation to CPU. GPU is now a low-cost ultra-large-scale parallel computing platform that is being used by a wide broad of different kinds of simulations, such as weather forecast [Michalakes and Vachharajani 2008], chemistry [Ufimtsev and Martinez 2008] and behavioral AI algorithms [Joselli et al. 2009d].

The first applications of GPUs performing general-purpose computation (GPGPU) had to rely on the adaptation of graphics rendering APIs, leading to a difficult learning curve and sometimes not very efficient data structures for the proposed solutions. CUDA [nVidia 2009] 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. Real-time simulation that explores this programming model on the GPU is a promising line of research, since it can speedup the high cost computation solution of the behavior simulation.

GPUs are a collection of SIMD processors designed to run streamed graphics pipelines. It is a computational model where the processing of each pixel is independent of the others and usually requires localized memory reads. There are rules of thumb to create efficient streamed applications, where, the most important one is to organize the data streams in a way that maximizes the memory read performance based on locality. These rules tend to result in more efficient usage of available memory and read ahead mechanisms of these devices.

Nowadays, it is being common to have systems with more than one GPU, being it for solving general purpose problems [Kim et al. 2011; Pharr and Fernando 2005] or used for improvement in rendering processing, such as the *Scalable Link Interface (SLI)* [nVidia 1998] introduced by NVidia. However, for the former, tasks such as the responsible to distribute the workload across multiple GPUs, manage the data exchange between the main memory and these de-

vices, and guaranteeing consistency between the multiple copies of data is enrolled to developers, making development for these architectures more difficult to build.

In this paper we present an architecture for crowd simulation using multiple GPUs in the same machine or located on a cluster of GPUs, allowing more realistic crowd simulation in the scene. Additionally, this architecture can use the number of GPUs available during the simulation. As far as the authors of this work knows, this is the first real time crowd simulation to use this kind of architecture. Besides that, we also present a simulation that is part of a game (based on the boids emergent behavior) to verify the performance of the proposed architecture.

The remainder of this paper is organized as follows. After referring to related crowd simulation works, in Section 2, we describe the crowd simulation and the Neighborhood grid acceleration structure employed in this work in Section 3. Detailed computation performed for our crowd simulation is presented in Section 4. Next we evaluate our architecture using a test case in Section 5 and present the results in Section 6. Finally, in Section 7 we present the conclusions of the paper.

2 Related Work

The first known agent-based simulation for groups of interacting animals is the work proposed by Craig Reynolds [Reynolds 1987a], 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 of 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 since 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 frames per second (fps) in a Playstation 2 hardware. 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. 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.

Because crowd simulation process each individual separately, they are very good for parallel processing. 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.

There are also important works that adapted the crowd simulation to the GPGPU architecture. Chiara et al. [Chiara et al. 2004] show a boids implementation on the GPU with the spatial hashing phase on the CPU. This work also implements some physical behavior with obstacle avoidance similar to the one by Reynolds [Reynolds 2006]. The work of Courty and Musse [Courty and Musse 2005] also presents a mix of GPU-CPU execution to simulate the behavior of a crowd with the influence of gaseous phenomena. This work uses force fields pre computed on the CPU to avoid collisions among the individuals on the crowd and the scenario (walls and obstacles). A more recent work in the GPGPU field present by Shopf et. al. [Shopf et al. 2008], which introduces 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. Also this last work also make collision avoidance with a similar approach as the continuum crowds [Treuille et al. 2006]. Another available works is [Passos et al. 2008; Passos et al. 2010; Joselli et al. 2009d; Joselli et al. 2009; Joselli et al. 2012b] where it presents another data structure based on the Extended Moore Neighborhood in the Cellular Automata theory instead of the traditional spatial hashing which can simulate and render up to 1,000,000 simple primitives with interactive frame rates. Even though this data structure seems promising, it does not appear to be precise enough for collision detection. There are a lot of others work on crowd simulation available on the literature, but most of them does collision avoidance approach, like [Sud et al. 2007; Kwon et al. 2008; Shao and Terzopoulos 2005; van den Berg et al. 2008], instead of a collision detection and response.

That are many works that deals with particle simulation using multiples GPUs. Fluid simulation using multiple GPUs are done by Da Silva et al. [Junior et al. 2012] using Smoothed Particle Hydrodynamics. Besides that, Ramada et al. [Hamada and Nitadori 2010] uses a cluster of GPUs connected in a network for performing astrophysical N-Body gravitational simulation. Unfortunately, it is inexistent crowd behavior that deals with the use of multiple GPUs.

The management of a the GPU architecture can be hard and there has been some works that deals exclusive with this management. The work by Joselli et al [Joselli et al. 2010a; Joselli et al. 2010b] presents an architecture for game loops is able to implement any game loop model and distribute tasks between the CPU and the GPU. Also, the work by Zamith et al. [Zamith et al. 2011] presents a game task that can be divided among several computers on the same network, using the available CPUs and GPUs. These works do not seen capable of managing diving the jobs, when there is multiples GPUs on the same computer.

There are also some works that deals with GPUs in a cluster environment. Fan et. al. [Fan et al. 2004] proposed an architecture of a GPU cluster and demonstrate it feasibility by a flow simulation using the Boltzmann model with 30 GPUs. Abdelkhalek et. al. [Abdelkhalek et al. 2009] designed a parallel simulator in order to solve the acoustic wave equation on a GPU cluster, using a finite difference approach with up to 8 GPUs. Doost et. al. [Doost et al. 2012] also uses an multi-GPU architecture to solve a acostic wave simulation in cluster of GPUs. Hartley et. al [Hartley et al. 2008] use a cluster of collaborate GPUs and CPUs in order to analyze biomedical images. These works are important, since they deal with the management of jobs in a multi-GPU ambient, but they are hard to implement in a real-time simulation, that has to deal with the visualization of the scene and also have a minimum response time, which is our case with a crowd simulation.

3 Crowd simulation

In a crowd, each entity has a particular vision in order to correct move and interact with the environment. To mimic this fact, our entities have a limited field of view, parameterized by an angle. Obstacles and other entities outside this field of view are not considered in the simulation. Figure 2 shows a comprehensible representation of this field of view.

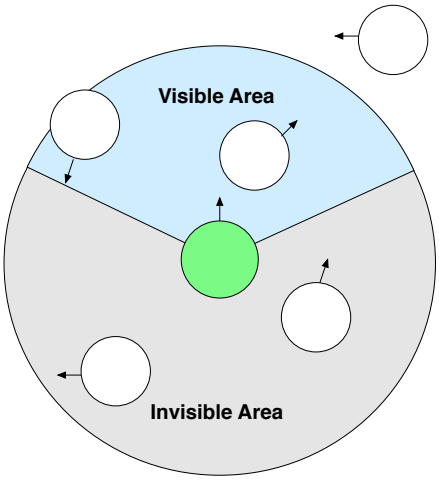

Figure 2: *The visual field of a entity*

For situations where two entities are very close to each other up to collide, a special case is triggered. In this situation, an entity takes into account neighbors even if they are outside of the their field of view. If collisions where allowed to happen, the simulation could become unstable since neighbor entity coming from behind would suddenly appear in front of another. It is possible to think of this as a collision detection for a prevention system, having the same effect as a movement made by animals that, even not seeing each other, would have gotten into a sudden contact.

As stated before, performing the referred processing has complexity of $O(n^2)$ for a collection of n entities using a brute force method, leading to an expensive operation, even for a small set of them. In order to avoid this time complexity, we employs an acceleration structure based on hash tables [Harris 2005] for locating nearby entities, which also allows the usage of an unbounded world. This acceleration structure requires a predefined number of slots, called **bucket**. Each of these buckets has two variables, indicating the starting and ending offset in the array containing the index for a entity, as shown in Figure 3 for an eight bucket hash grid. A bucket that does not have any entity is set with a special flag, to avoid its wrong computation during simulation.

Before the hash table processing, a preliminary operation produces a hash key for each entity by using the absolute position of the entity through the use of the algorithm proposed by Teschner et al. [Teschner et al. 2003]. This algorithm receives *p1, p2 and p3*, which are the greatest prime number used to minimize hash key conflicts (chosen in our tests as *73856093*, *19349663* and *83492791*, respectively). It also receives the parameter *cell_size*, that represents the imaginary grid's cell size.

Following the original algorithm, the same hash key is produced for entities that are located at symmetrical positions in the world, causing unnecessary data processing. In order to avoid this observed problem, we introduced a new parameter in the proposed algorithm named *world_limits*. It is included in the hash key generation formula and represents the world's bounding box, calculated at each time step. The algorithm with our modifications is presented in Algorithm 1.

Following, after each entities' hash key calculation, it is necessary to sort these entities based on its calculated hash key. This operation is done in GPU by using the radix sort algorithm [Huang et al. 2009], presented in CUDPP library [1]. This way, using a imaginary cell size equals the vision radius K_r, only 27 buckets are processed during entities' processing (three grid cells in each dimension).

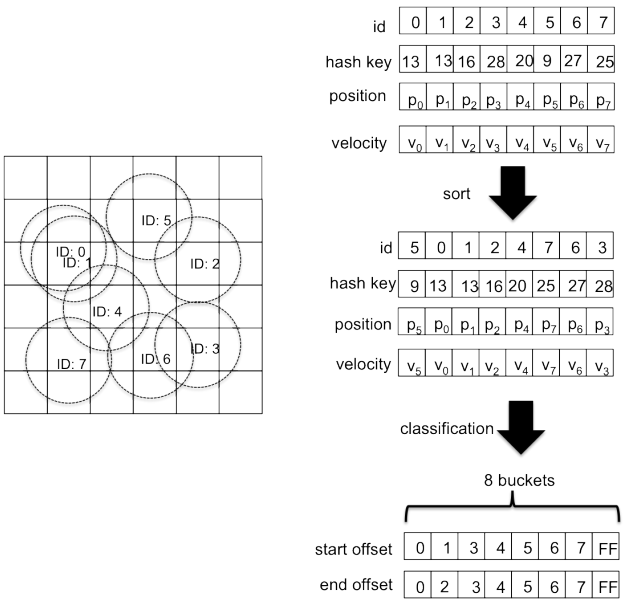

Figure 3: *Process of generating an acceleration data structure based on a hash table.*

Algorithm 1: Algorithm for hash key genenation.

Input: float3 pos, float3 cell_size, int num_buckets, float3 world_limits
Output: unsigned int hash_code
int x ← (int) ((pos.x + world_limits.x) / cell_size.x);
int y ← (int) ((pos.y + world_limits.y) / cell_size.y);
int z ← (int) ((pos.z + world_limits.z) / cell_size.z);
hash_code ← ((x * p1) xor (y * p2) xor (z * p3)) mod num_buckets;

4 Multiples GPUs architecture

Processing entity neighbor requires knowing which cells are close to a entity being processed in order to steer and avoid collision. This fact leads to a dependency between them, as presented in Figure 4.

The first strategy to allows more than one GPU to process entities is to split up the domain among the available GPUs, being entities located in the border processed differently from the ones located inside the grid. In this case, entities that are not in the edge of the grid can be processed normally, as all its dependency data is available on the same GPU. On the other hand, entities that are located in the edge cannot be processed, as half of its dependency data are located in another GPU's memory. Figure 5 shows this technique for 2 GPUs in a host. For the cases where GPUs are distributed over nodes in a network, this data can be transferred using NVidia GPU Compute Direct with CUDA language [nVidia 2009], which enables peer GPU to GPU memory communication over a network. For local GPUs, this data can be shared using Per to Per (P2P) communication, which enables a single view of the whole GPU's memory in the host.

Processing entities using a set of GPUs located at independent nodes in a network is done by a collection of ordered tasks, according to Algorithm 2. In order to avoid the complexity of the code for minor performance, in a first attempt, only behavior tasks that requires more computational effort are distributed among various GPUs.

[1] Available at http://gpgpu.org/developer/cudpp

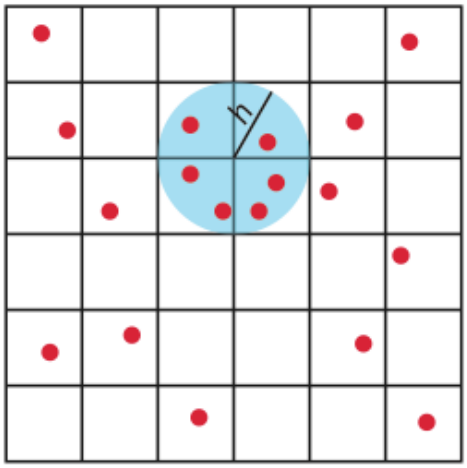

Figure 4: *Entities' data dependency located in different cells.*

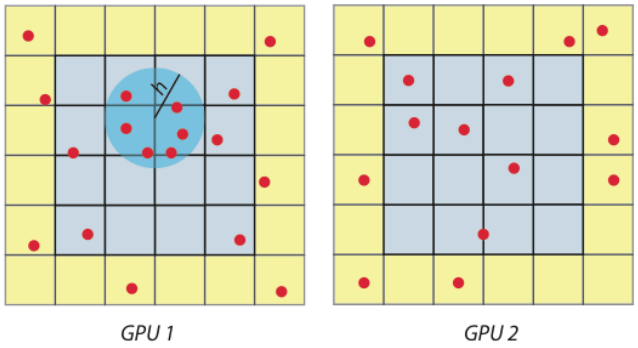

Figure 5: *Simulation domain distributed over 2 GPUs. Entities located at yellow cells need to be transferred between them before its computation, while entities at blue cells can be processed independently.*

As can be seen in Algorithm 2, the first task that needs to be done is called **entities' structuring**. Mainly this task calculates a hash code for each entity in order to group them in the same cell. This task is performed by only one GPU, as concurrency data writing is needed. It is important to state that this task is not an intrinsic part of crowd simulation, used here only to avoid de $O(n^2)$ complexity. Additionally, beside this task, all subsequent computations are done in parallel by available GPUs.

For all subsequent tasks, processing entities requires data access to all neighborhood entities inside the *kernel radius*, such as position, type and velocity. To allow data sharing, they are done by a collection of synchronized steps, according to Figure 6.

In the first step, after entity have been grouped, the main GPU, which stores all the entities in the simulation, starts to divide, equally, all of them among the available GPUs and starts sending their respective data for processing. As long as all GPUs receives its data, immediately it starts a kernel to perform the calculation of all the behavior. During this calculation, it is important to notice that only entities that are not in the grid's edge can be processed, as data in the grid's edge are not yet available. In parallel, each GPU in the node starts sending entities located at its edge to it's neighborhood GPU. This information is made available for each GPU by the *master GPU*, the ones who manages all tasks performed by each GPU.

After the end of the previous step, each GPU is responsible to check if all required information is available. In this case, after finishing the previous step, each GPU starts the second step. In this second step, entities located in the edges are processed, using the same tasks that were performed in the entities in the previous step.

Finally, in the *integration* task, new velocities and position are calculated for all entities and sent back to *master GPU*. It is important to notice that, absent for the integration task, all the other one do not need to synchronize, allowing the GPU to stay busy all the time.

Algorithm 2: Algorithm describing the high level steps during the simulation for various GPUs.

```
Input: entities, num gpus
entities structuring(GPU master);
synchronize();
amount entities per gpu = entities / num gpus;
for availabe GPU do
    share-data(edge-position-entities);
    calculate-behavior-forces(inner-entities);
end
synchronize();
for availabe GPU do
    calculate-behavior-forces(edge-entities);
    integrate(inner-entities);
    share-data(edge-density-entities);
end
synchronize();
for availabe GPU do
    integrate(edge-entities);
    share-data(all-entities-positions);
end
synchronize();
```

In the first version, our approach are able to deal with GPUs located on the same machine, using the P2P communication among the GPUs. Even with this restriction, our architecture leads to a better speedup, as communication with CPU memory is avoided. For future versions, we will allows crowd simulation to be performed over various GPUs located in different hosts over the network.

5 Evaluation

For the purpose of this work, we have choose to validate the proposed technique by implementing a behavior algorithm based on a well-known flocking boids algorithm [Reynolds 1987b]. This algorithm is able to provides good visual results, near to real world behavior observation of huge groups of animals and people. As a test scenario, the street Carnival of Rio de Janeiro is simulated, which in the real word can have over a million of people in some cases. This simulation is going to be used as part of a game called Máquina do Carnaval to be release soon .

The proposed model simulates a crowd of people interacting with each other and avoiding random obstacles in the space. In real life, people normally tend to form groups, which are typically their friends and colleagues. They also try to follow the "Carnival block", which is the name of the group that organizes the party, giving direction to the crowd and playing the music. In our simulation, both characteristics can be observed.

In order to achieve a believable simulation, our approach try to mimic what is observable in the Carnival: many crowd behaviors resemble state machines and cellular automata, where a combination of internal and external factors define which actions are being taken and how they are made. A state machine is employed to decide which actions are going to be taken, which are performed themselves by a cellular automaton algorithm. In this proposal, internal state is represented by the entity type and external ones corresponds to the visible neighbors, depending from where the entity is looking at (direction), and their relative distances.

Based on these ideas, the simulation algorithm uses internal and external states to compute these influences for each entity: grouping (some people tend to form groups of friends); repulsion (people tend to stay away from others by a variable distance); leader following (most people follows their leader, represented by the "Carnival block"); avoid obstacle (the crowd avoid obstacles in their path); avoid all (people who tries to avoid all the obstacles and also the "Carnival block" which can happen in situations where she/he does not belong to the party or wants to leave the event). Additionally, there are multiplier factors which dictate how each influence type

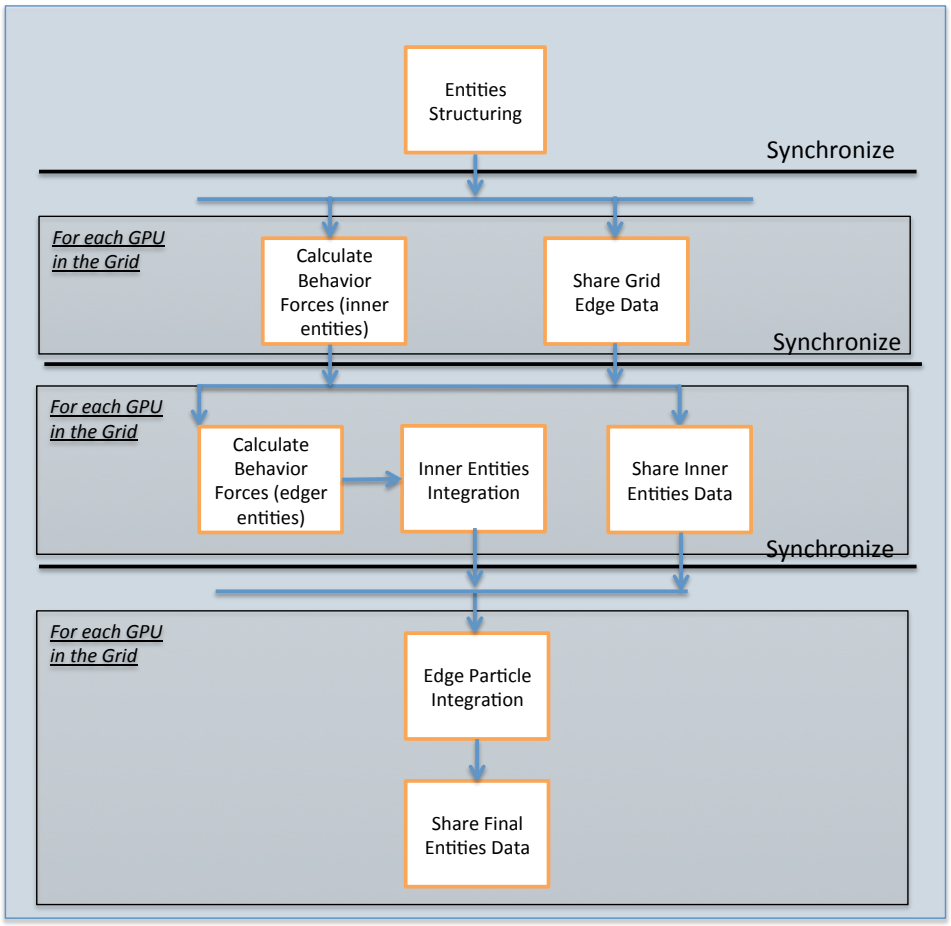

Figure 6: *Crowd simulation's tasks distribution over our multi GPU architecture.*

may get blended to another, applied in each step. In order to enable a richer simulation, these factors are stored independently for each type of entity and entity group in separate arrays.

This scenario was implemented using CUDA technology [nVidia 2009]. The loop of the proposed architecture will process the following simulations steps/methods:

- Get the user's input, sending it to the GPU;
- Make a spatial hash on the individuals, i.e, a broad phase of the collision detection;
- Calculate the narrow phase of the collision detection, i.e, apply the collision in each individual;
- Applying the gravity force on the crowd individuals;
- Use the broad phase to determinate the neighborhood of the crowd individuals;
- Apply the crowd behavior type in each individual, as forces;
- Present the results, i.e., render the crowd and scenario.

The data that is changed during the user input step is encapsulate in special structure, in order to keep the communication between the GPUs and the CPU to a minimum (which can be a bottleneck of a simulation [Krueger 2008]), and then sent to the GPU.

The narrow phase of the collision detection, the application of the gravity force, the determination of the individuals and the application of the crowd behavior is done in one single CUDA step, in order to optimize the loop.

The render is responsible for present the results to the user. In this work it uses OpenGL for doing that. The crowd individuals can be represented in difference ways: as impostors, or as simple primitives. Impostor is a way of render a lot of individuals because it only rendes a quad texturing with a pre-render figure of the crowd individual. And the simple primitives is very fast but it is not very suitable for graphics applications.

6 Results

For the tests, a PC running Ubuntu 10.10 Linux distribution equipped with an Intel i7 3.07GHz using 8 GB of RAM was used. This PC has two NVidia GeForce GTX 580 with 1 GB DDRAM each, where 512 cores were used for the simulation. Tests using different configurations for performing the simulation were also performed: the first one uses sprites, while the second one uses simple primitives (for performance tests). To assure that results are consistent, each test was repeated ten (10) times and the standard deviation of the average times was confirmed to be within 5%.

Table 1 shows the simulation of crowd made by using both a single GPU and our multi GPU architecture. A graph showing the curve of this simulation can be seen in Figure 7. The column labeled FPS represents the *frames per second* which measure the time necessary to update and render the simulation. *Speedup* is measured by the relation of FPS Multi GPU over FPS Single GPU.

According to the presented result, it is possible to see that using more than one GPU has increased the overall performance of the simulation. In this case, growing the number of available GPUs decreases the time to perform this simulation, allowing more entities to be used, or even a more complex behavior, in order to grow up the realism of the simulated crowd. In Figure 8, a screenshot of the simulation can be seen, where Figure 8(a) uses primitives for rendering while Figure 8(b) uses sprites. In this simulation, a to-

Table 1: *Results of the simulation using both a single GPU and our multi GPU architecture.*

#	Single GPU		Multi GPU		
Entities	FPS	Time	FPS	Time	Speedup
4,096	1103	0.90 ms	1158	0.86 ms	1.05
16,384	588	1.70 ms	676	1.48 ms	1.15
65,536	211	4.74 ms	280	3.57 ms	1.33
262,144	38	26.3ms	60	16.6ms	1.58
1,048,576	6	166.7ms	11	90.9ms	1.83

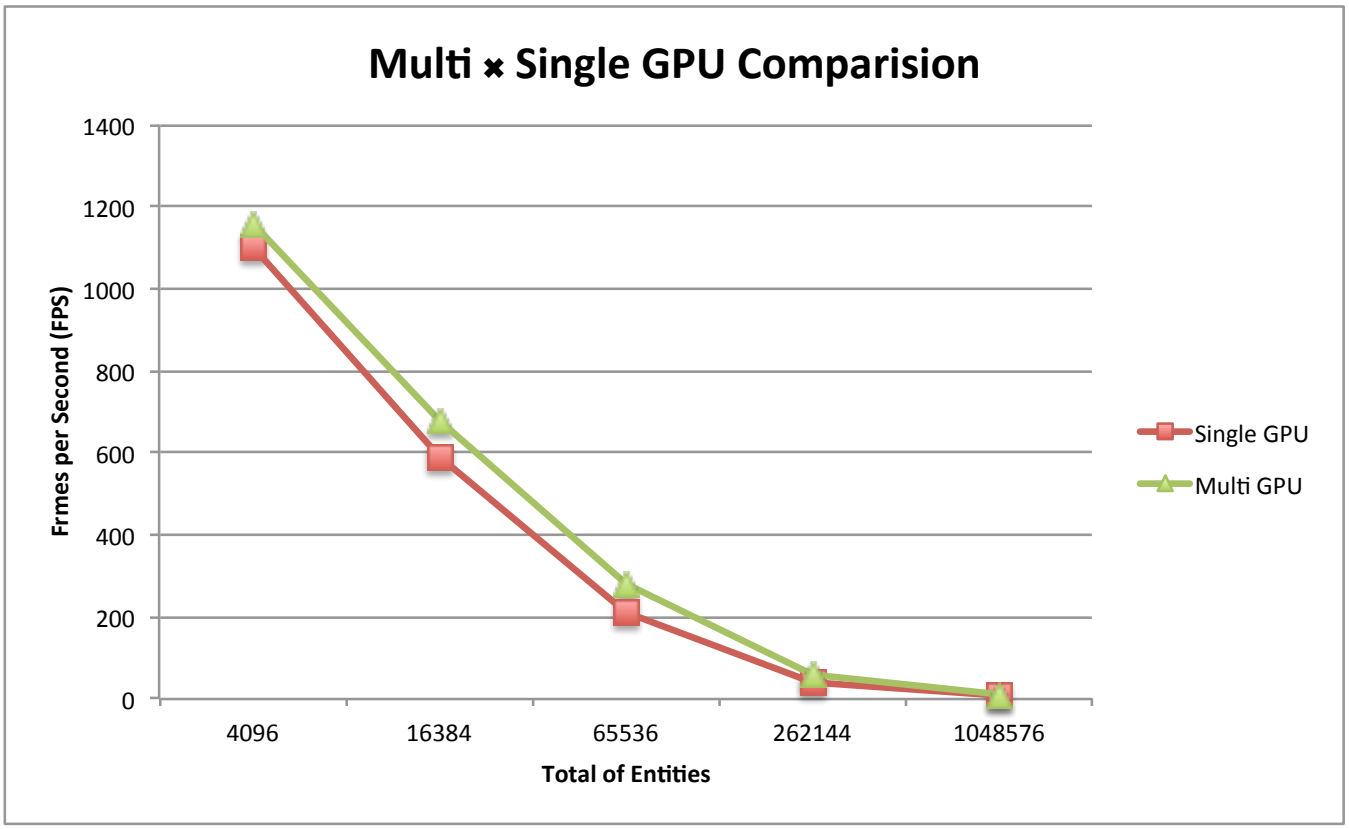

Figure 7: *Crowd simulation comparison using single and multi GPU architecture.*

tal of 256,000 entities (only 56,000 entities appears in the scene) is employed.

7 Conclusions and Future Work

Our proposed simulation using a multi GPU architecture reaches a speedup of almost twice the implementations of the most recent full GPU based approaches, allowing the simulation of more complex behavior in real time games and other kinds of simulation. The main acceleration factor comes from the fact that many complex and time consuming tasks can be split up among a collection of GPUs during the simulation processing.

According to the results presented, the proposed architecture is being extended in order to enable the use Dynamic Parallelism of the Kepler GPUs architecture, which allows dispatch of CUDA kernel inside the kernel being processed. This way, we can avoid spend time processing empty cells during the simulation.

Additionally, due to the capacity of GPUs, an architecture that sends data for processing according to each device capability would be very beneficial for the whole simulation. This way, entities are sent for GPUs with less capability, avoiding possible bottlenecks.

In the future, we plan to measure the scalability of this architecture when more than two GPUs are used both in a local connection as well as across the network.

Acknowledgements

Special thanks to Secretariat of Culture of the State of Rio de Janeiro for sponsoring this project. The developer Team for assisting in this study and for the inspiring and fruitful cooperation. Also thanks to Daniel Mafra, Vynicius Moraes Pontes, Rafael Langoni Smith and Licnio de Souza Ribeiro for all the hard-work on this project. We also like to thanks Eduardo Soluri for his help in the beginning of the project.

References

ABDELKHALEK, R., CALANDRA, H., COULAUD, O., ROMAN, J., AND LATU, G. 2009. Fast seismic modeling and reverse time migration on a gpu cluster. In *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*, IEEE, 36–43.

AITKEN, M., BUTLER, G., LEMMON, D., SAINDON, E., PETERS, D., AND WILLIAMS, G. 2004. The lord of the rings: the visual effects that brought middle earth to the screen. In *ACM SIGGRAPH 2004*, ACM Press, New York, NY, USA, SIGGRAPH: ACM Special Interest Group on Computer Graphics and Interactive Techniques.

CHIARA, R. D., ERRA, U., SCARANO, V., AND TATAFIORE, M. 2004. Massive simulation using gpu of a distributed behavioral

(a) Optimization mode.

(b) Normal mode.

Figure 8: *Crowd simulation for the Carnival scenario.*

model of a flock with obstacle avoidance. In *Vision, Modeling, and Visualization (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.

DOOST, M. S., SADJADI, S. M., DA SILVA, J., ZAMITH, M., JOSELLI, M., AND CLUA, E. 2012. Architecture of request distributor for gpu clusters. In *Applications for Multi-Core Architectures (WAMCA), 2012 Third Workshop on*, IEEE, 13–18.

FAN, Z., QIU, F., KAUFMAN, A., AND YOAKUM-STOVER, S. 2004. Gpu cluster for high performance computing. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, IEEE Computer Society, 47.

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/`.

HAMADA, T., AND NITADORI, K. 2010. 190 tflops astrophysical n-body simulation on a cluster of gpus. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE Computer Society, Washington, DC, USA, SC '10, 1–9.

HARRIS, M. 2005. Fast fluid dynamics simulation on the gpu. In *ACM SIGGRAPH 2005 Courses*, ACM, New York, NY, USA, SIGGRAPH '05.

HARTLEY, T. D., CATALYUREK, U., RUIZ, A., IGUAL, F., MAYO, R., AND UJALDON, M. 2008. Biomedical image analysis on a cooperative cluster of gpus and multicores. In *Proceedings of the 22nd annual international conference on Supercomputing*, ACM, 15–25.

HUANG, B., GAO, J., AND LI, X. 2009. An empirically optimized radix sort for gpu. *Parallel and Distributed Processing with Applications, International Symposium on 0*, 234–241.

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., PASSOS, E. B., ZAMITH, M., CLUA, E., MONTENEGRO, A., AND FEIJO, B. 2009. A neighborhood grid data structure for massive 3d crowd simulation on gpu. *Games and Digital Entertainment, Brazilian Symposium on 0*, 121–131.

JOSELLI, M., PASSOS, E. B., ZAMITH, M., CLUA, E., MONTENEGRO, A., AND FEIJO, B. 2009d. A neighborhood grid data structure for massive 3d crowd simulation on gpu. *Games and Digital Entertainment, Brazilian Symposium on*, 121–131.

JOSELLI, M., ZAMITH, M., CLUA, E., LEAL-TOLEDO, R., MONTENEGRO, A., VALENTE, L., FEIJO, B., AND PAGLIOSA, P. 2010. An architeture with automatic load balancing for real-time simulation and visualization systems. *JCIS - Journal of Computational Interdisciplinary Sciences*, 207–224.

JOSELLI, M., ZAMITH, M., CLUA, E. W. G., MONTENEGRO, A., LEAL-TOLEDO, R. C. P., VALENTE, L., AND FEIJÓ, B.

2010. An architecture with automatic load balancing and distribution for digital games. In *Games and Digital Entertainment (SBGAMES), 2010 Brazilian Symposium on*, IEEE, 59–70.

JOSELLI, M., SILVA JUNIOR, J. R., ZAMITH, M., SOLURI, E., MENDONCA, E., PELEGRINO, M., AND CLUA, E. W. G. 2012. Techniques for designing gpgpu games. In *Games Innovation Conference (IGIC), 2012 IEEE International*, 78–82.

JOSELLI, M., PASSOS, E. B., JUNIOR, J. R. S., ZAMITH, M., CLUA, E., AND SOLURI, E. 2012. A flocking boids simulation and optimization structure for mobile multicore architectures. *SBGames 2012*.

JUNIOR, J. R. D. S., JOSELLI, M., ZAMITH, M., LAGE, M., CLUA, E., SOLURI, E., AND TECNOLOGIA, N. 2012. An architecture for real time fluid simulation using multiple gpus. *SBC-Proceedings of SBGames*.

KIM, J., KIM, H., LEE, J. H., AND LEE, J. 2011. Achieving a single compute device image in opencl for multiple gpus. In *Proceedings of the 16th ACM symposium on Principles and practice of parallel programming*, ACM, New York, NY, USA, PPoPP '11, 277–288.

KRUEGER, J. 2008. A gpu framework for interactive simulation and rendering of fluid effects. *IT - Information Technology 4*, (accepted).

KWON, T., LEE, K. H., LEE, J., AND TAKAHASHI, S. 2008. Group motion editing. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, 1–8.

MICHALAKES, J., AND VACHHARAJANI, M. 2008. Gpu acceleration of numerical weather prediction. *IEEE International Symposium on Parallel and Distributed Processing*, 1–7.

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, 1998. Scalable link interface. Avalible at: `http://www.nvidia.com/hardware/technology/sli`.

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`.

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.

PASSOS, E. B., JOSELLI, M., ZAMITH, M., CLUA, E. W. G., MONTENEGRO, A., CONCI, A., AND FEIJO, B. 2010. A bidimensional data structure and spatial optimization for supermassive crowd simulation on gpu. *Comput. Entertain. 7*, 4 (Jan.), 60:1–60:15.

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.

PHARR, M., AND FERNANDO, R. 2005. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional.

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, 25–34.

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, 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.

ROCKSTAR, 2013. Grand theft auto v. [CD-ROM].

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, 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.

SUD, A., GAYLE, R., ANDERSEN, E., GUY, S., LIN, M., AND MANOCHA, D. 2007. Real-time navigation of independent agents using adaptive roadmaps. In *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 99–106.

TESCHNER, M., HEIDELBERGER, B., MUELLER, M., POMERANETS, D., AND GROSS, M. 2003. Optimized spatial hashing for collision detection of deformable objects. In *In Proceedings of VMV'03*, 47–54.

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.

UFIMTSEV, I. S., AND MARTINEZ, T. J. 2008. Quantum chemistry on graphical processing units. 1. strategies for two-electron integral evaluation. *Journal Chemistry Theory Computation 4 (2)*, 222 – 231.

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., VALENTE, L., JOSELLI, M., CLUA, E., TOLEDO, R., MONTENEGRO, A., AND FEIJ, B. 2011. Digital games based on cloud computing. In *SBGames 2011 - X Simpsio Brasileiro de Jogos para Computador e Entretenimento Digital*.