Real Time Visualization of Crowd Dynamics Scenarios

Dan Razvan Ilies
Technical University of Cluj-Napoca
Str. G. Baritiu 28, 400027, Cluj-Napoca, Romania
danilies92@gmail.com

Adrian Sabou
Technical University of Cluj-Napoca
Str. G. Baritiu 28, 400027, Cluj-Napoca, Romania
adrian.sabou@cs.utcluj.ro

Dorian Gorgan
Technical University of Cluj-Napoca
Str. G. Baritiu 28, 400027, Cluj-Napoca, Romania
dorian.gorgan@cs.utcluj.ro

ABSTRACT
This paper presents an approach to real-time simulation of crowd dynamics on GPU enabled computing architectures. We discuss challenges with parallelization of agent-based models, implementing parallel simulation algorithms, visualization and interaction with the simulated scene and, most importantly, ensuring communication and synchronization between all these processes. Our main objective is to provide interactive simulation of realistic models such as pedestrian dynamics, in which large crowds move and interact among themselves and with the environment. Simulation parameters like scene complexity, scene composition, as well as the number of agents are varied in order to simulate different scenarios and to assess the impact on performance.

Author Keywords
Visualization; Crowd dynamics; Interactive simulation; Social forces; Social models; Graphics Processing Unit.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms
Algorithms; Human factors.

INTRODUCTION
Applications in computer animation and simulation vary from simple scenarios like physically-based simulations, which focus on visual quality, perceived realism and computation efficiency, to complex scenarios which require an accurate reproduction of measured parameters. The latter are usually encountered in critical systems' simulation, based on heavy numerical computation, with the aim of ensuring exact calibration with real-world conditions. Such examples can be encountered in Computational Fluid Dynamics, modelling turbulent flows or crashworthiness tests. All these applications require a huge amount of processing power and utilize powerful techniques borrowed from fields such as applied mathematics, numerical analysis, computational physics or mechanical engineering in order to satisfy the ever increasing expectancies that users have with regard to simulation systems.

Over time, sequential computing devices such as the Central Processing Unit (CPU) have evolved significantly. At first, there was little change in algorithms and techniques from one version to another, the performance mostly being improved by raising the operating frequency. Once the thermal barrier was reached, the parallel era began. Thus CPUs evolved into powerful multicore architectures and this direction led to creating a new type of computing device, dedicated to massive parallel computation, the Graphics Processing Unit (GPU). This new computing direction brought forth the need for efficient parallel techniques and algorithms in order to fully utilize existing resources.

GPUs were at first used to process massive parallel data involved in Computer Graphics and complex graphical algorithms, but, because of their massively data parallel architecture, they drew the attention of the High-Performance Computing (HPC) community and researchers began using them for general purpose parallel computation. Thus the General Purpose Computation on the GPU (GPGPU) current began. However, programming GPUs for general purpose computation proved difficult to learn, so GPU vendors began making their devices more flexible and easier to program through interfaces such as CUDA and frameworks such as OpenCL.

Among applications to benefit from GPU-based acceleration are discrete, particle-based or agent-based simulations, their inherently parallel nature making them perfectly suitable for GPU-based implementations. Thus, in the last decade, research work in this domain has produced new, highly efficient techniques for simulations, especially for the most time consuming parts. However, when adding the real-time attribute that such simulations usually require, along with the need to transform raw processed data into a form visually meaningful to the users, we end up with probably one of the most complex class of applications attempted to be implemented on such architectures.

This paper presents an approach to real-time simulation of crowd dynamics on GPU enabled computing architectures. We discuss challenges with parallelization of agent-based models, implementing parallel simulation algorithms, visualization and interaction with the simulated scene and, most importantly, ensuring communication and synchronization between all these processes. Our main objective is to provide real-time simulation of realistic models such as pedestrian dynamics, in which large crowds move and interact among themselves and with the environment. Simulation parameters like scene complexity, scene composition, as well as the number of agents are varied in order to simulated different scenarios and to assess the impact on performance. The rest of the paper is organized as follows: the following section highlights on existing related works, the third section presents an overview of the model used for our simulations, section four focuses on techniques used for achieving interactivity and real-time visualization, while section five presents some experimental scenarios and performance measurements.

RELATED WORKS

The term of crowd dynamics refers to a system of behaviors and psychological processes that arise in a social group or between several such groups. Studying the dynamics of crowds can be useful for understanding behaviors in decision making, information diffusion or disease spreading in society, as well as in many other contexts. Such applications can be seen in domains such as: psychology, sociology, political sciences, business or education.

The idea that macroscale behavior can be deduced from microscale interactions between individuals has become key concept in understanding human behavior in crowds. Examples of such collective behaviors have been represented in several ways, one of them being the social forces model.

A social forces model sufficiently complex to allow for realistic computer-based simulations of human crowds has been introduced by Helbing and Molnar [3]. They claimed that pedestriam motion can be described in such a way as to be the result of social forces. These forces are not directly exerted only by the environment, but are rather a measure of the internal motivation of each individual that arrises in order to guide the individual towards certain actions or movements.

After the initial introduction of this model, different improvements and variations started to emerge, since it was found that it could also be applied to a variety of domains other than pedestrian dynamics, like biology, for simulating the behavior of microscopic particles. Different variations of attraction and rejection forces lead to new models being created that described the behavior of large crowds of particles even better than the initial one and that would realistically reproduce several observed phenomena. Thus several important works emerged which present collective patterns, [8] [10] [6] and [4].

However, the assumptions that the above mentioned model relies on and the exact form of the presented social forces has never been measured or validated empirically, even if the functions describing the interaction between individuals could definitely influence the behavioral patterns that were the outcome of the simulations. This has been proved by a series of studies on different animal species [1]. The most accurate and correct studies were limited to calibrating the assumed parameters for the interaction forces in order to minimize the errors in predicting individual behaviors.

Regarding simulating social models on high-performance architectures, Joselli et al. [5] present a case study in which they evaluate the performances of a simulation system with one CPU and two GPUs. The model that they use is an agent-based one, in which each individual has several properties and interact with their environment. They used a series of data structures and algorithms in order to accelerate the computation, most of them done in the GPU. Among the used techniques are spatial hashing, in which each agent is assigned a hash code based on its location in the simulated scene. The authors have chosen to split the scene among the two GPUs in such a way as to allow each one to process part of the scene and to synchronize common areas. Their experiments show a speedup up to 1.8 for a very large number of agents. However, the authors do not specifically discuss real-time visualization and techniques that might improve performance when interactivity is required.

Sabou et al. [9] present an extension of particle models with regard to their initial purpose and propose a solution for simulating sociophysics models interactively using a particle-based visual approach. An existent agent-based "small-world" model is mapped on a particle-based grid and its evolution in time is simulated on a high-performance graphics cluster in order to model technology adoption and consumer behavior. Several experimental scenarios validate the initial hypothesis that particle-based models can be extended beyond their original scope and evaluate the system's performance and scalability.

SIMULATING CROWD DYNAMICS

As seen in the previous section, agent-based models contain individuals that interact in a given environment. The agents may be either distinct computer programs or distinct parts of the same program, with the purpose of representing social actors – persons, organizations or nations. These agents are programmed to react to the computing environment that they are placed in, a model of a real environment in which the agents would interact.

One crucial aspect of agent-based modeling is that agents need to be able to interact, i.e. exchange information carrying messages and to behave accordingly with what they learn from these messages. The messages can be, for instance, specific dialogs between persons, but also indirect means of transferring information such as observing another agent or detecting the effects of another agent's actions. The possibility to model such interactions is the main way in which agent-based modeling differs from other computational models.

Of course, keeping in mind that this is a visual simulation, the agents will have to be visually represented in one way or another. In our particular case, we have chosen a representation in a bi-dimensional virtual scene in which each entity is described by its *X* and *Y* coordinate. This way, users can observe in real-time the movement of agents and their interactions.

We will now describe the construction of the simulated scenario and the different types of forces that act on agents.

Defining the simulation scenario

Besides the main actors of the simulation, which are the agents, in order to obtain a realistic model, we had to introduce several other elements. Thus, we will have:

Scene boundaries

It was decided that all simulations were to take place in a restricted environment which can be defined to the user's best suiting. This way, the one that initiates the simulation has the possibility to choose the horizontal as well as the vertical limits that will constitute the boundaries of the simulated scene. Agent movement is restricted between these user defined limits, providing for a more controllable scenario and a better visualization experience. The boundaries will be drawn as simple straight walls through which agents cannot pass.

Obstacles

Besides the aforementioned boundaries, the interior of the scene will contain different obstacles that will influence the agents' trajectories. The obstacles can have various shapes, ranging from simple walls to complex polygonal objects. Same as with boundaries, the users can control obstacle placement as they see fit, both their position and their shape.

Social forces

Each agent has a clearly defined objective during the simulation. Furthermore, during their movement towards their objective, there will be a series of interactions, both agent-agent and agent-other elements. The model for the forces that act upon agents and the equations of movement are the same originally proposed by Helbing and Molnar [3]. As they said, it is often believed that human behavior is chaotic or unpredictable, but, for relatively simple situations, certain behavior patterns can be created, among which, the social forces model. Due to the fact that pedestrians are already used to a multitude of situations, their reactions are most often automatic, based on their similar previous experiences. Thus, the velocity and the direction of each pedestrian could be represented as a vector quantity $\vec{F_\alpha}(t)$, which is the so called social force and which represents the effect of several other forces that the environment and other pedestrians generate. In what follows, we will briefly describe the types of forces that influence the pedestrians' movement.

Attraction forces towards the objective

This is the main force that drives the agent towards its goal. Normally, agents will take the shortest route, which is a straight line, unless they encounter obstacles, in which case they will temporary modify their objective in order to avoid them. The formula that computes these forces is:

$$\vec{F}_\alpha^0(\vec{v}_\alpha, v_\alpha^0\vec{e}_\alpha) := \frac{1}{\tau_\alpha}(v_\alpha^0\vec{e}_\alpha - \vec{v}_\alpha) \qquad (1)$$

where $\vec{v}_\alpha$is the current speed, $v_\alpha^0\vec{e}_\alpha$ is the desired speed and $\tau_\alpha$is the relaxation time (i.e. the delay in agent acceleration).

Repulsion forces from obstacles

On their way towards the objective, an agent ca encounter different types of obstacles. Normally, even in real life, pedestrians keep their distance to obstacles such as walls or other kind of objects, thus we require a formula to express a force of repulsion coming from obstacles. This formula is the following:

$$\vec{F}_{\alpha B}(\vec{r}_{\alpha B}) := -\nabla_{\vec{r}_{\alpha B}} U_{\alpha B}(\|\vec{r}_{\alpha B}\|) \qquad (2)$$

where $\vec{r}_{\alpha B}$is the distance between the agent's current position and the obstacle and $U_{\alpha B}(\|\vec{r}_{\alpha B}\|)$ is a monotonic decreasing potential that scales the repulsion force with the distance between the agent and the obstacle.

Repulsion forces from other agents

Similar to the previous case, when a repulsion force is generated by the obstacles, repulsion forces are generated from other agents. When agents are in proximity, this repulsion force appears that is intended to keep agents from violating each other's "personal space". The repulsion depends on the distance between agents and their relative velocities. For great distances, this force is negligible, but as agents get closer, it will increase exponentially. The formula to compute these forces is:

$$\vec{F}_{\alpha\beta}(\vec{r}_{\alpha\beta}) := -\nabla_{\vec{r}_{\alpha\beta}} V_{\alpha\beta}[b(\vec{r}_{\alpha\beta})] \qquad (3)$$

where $V_{\alpha\beta}[b(\vec{r}_{\alpha\beta})]$ is a monotonic decreasing function with the same role as before and *b* is the small radius of the elliptic shaped personal space of the agent.

Acceleration structure based on hash codes

In order to speed-up the neighbor search when dealing with a large number of agents, we apply a spatial hashing technique. The simulated world is split into a grid of cells (Figure 1), each agent belonging to a single such cell at any given time. Thus, we have to establish a relation between the agent and the cell, based on the agent's position.

The basic idea is to firstly determine the position of an agent in the scene and secondly we must search for the cell that contains that position. To speedup calculations, each agent is assigned a hash code, computed using the following formula:

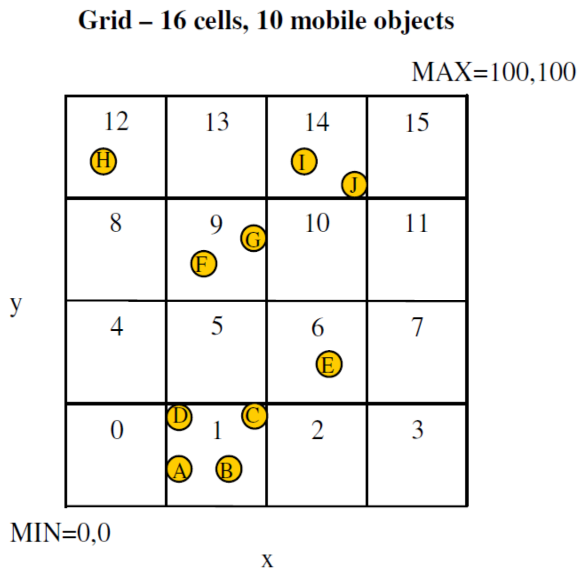

Figure 1. Example grid for 16 cells and 10 agents [2].

$$HashCode(agent) = \left((x * p1)\ xor\ (y * p2)\right) mod\ p3 \quad (4)$$

where

$x = agent.x/cell\_size$

$y = agent.y/cell\_size$

and *p1*, *p2* and *p3* are large prime numbers.

The role of this formula is to assign a code to each agent based on its position in the scene. The purpose of these codes is to use them to obtain an agent-cell association, which explains why the code computation depends on the size of the cells dividing the simulation space. Code computation for each agent must occur at each simulation step since agents modify their positions as long as their objective was not reached. Moreover, to be able to associate agents and cells, the latter need to have an assigned code, computed using the same formula. Since cells do not modify their positions, their codes can be precomputed.

Thus, after computing the aforementioned codes, each agent will be assigned the hash code of the cell that it belongs to, making it easy to quickly determine the neighbors for each agent during the simulation process. In order to guarantee a unique code for each cell, the prime numbers must be much larger than the total number of cells. Code computing is done using the specified formula for better performance and computing speed.

REAL-TIME SIMULATION AND VISUALIZATION

Accelerating computation using the GPU

GPUs are electronic components specially designed to execute a huge number of operations in parallel. Their initial purpose was to create raster images in a framebuffer to present through a display device, but they have recently started to be used more and more for applications and systems designed to offer a huge degree of parallelism.

GPU-accelerated computing is a technique that uses a GPU together with a CPU to accelerate scientific, analytic or engineering applications. This path was opened by NVIDIA in 2007 and a level was reached in which GPUs power entire data centers, especially power efficient ones serving Universities and small and medium enterprises [7]. GPUs accelerate a wide range of applications, from applications in the auto industry to mobile phone apps, drones or robots, offering superior performances.

The way this acceleration works is by taking over the intense workloads from the CPU and running them in the GPU, while the rest of tasks continue running in the CPU.

In our case, the main and most complex entities are the agents. Most computation is done around them, the rest of the scene being mainly static. Thus, fast processing of agents would lead to better overall simulation performance. Seeing as all agents require the same set of operations at one time, processing them on the GPU is the best choice. Each agent will be processed by a different GPU thread and necessary data will be transmitted between host and device.

After deciding on all elements required for host-device communication, the simulation can be attempted. We must firstly configure all simulations and visualization parameters and after that we can start an infinite loop. This simulation loop contains two main phases: the computation phase and the visualization phase.

The computation phase is done in a distinct function which is called from inside the infinite loop and contains a series of commands to be executed, either in the CPU or on the GPU. Computing new agents' positions is done in three distinct steps:

1. The first kernel computes the hash codes for all agents using their positions in the scene.

2. The second step consists of computing the hash structures that allow for fast discovery of all agents inside of a specified cell, using the second kernel.

   This way, we get two data structures with the same dimension as the number of cells, the first one indicating which is the first agent inside a certain cell and the second one indicating what is the last agent in that cell, with regard to a data structure in which agents are ordered by their hash code.

3. The third step is the actual computing of forces, new positions and velocities for all agents, using the third kernel.

Once computation is finalized, the scene must be prepared for rendering.

Rendering is done as a two-step process. The first step involves rendering all elements in the scene besides the agents (scene boundaries, obstacles, grid of cells, etc.). The second step involves rendering the agents themselves. As the number of agents becomes sufficiently large, the computing process is no longer the only issue that has an impact on performance, the sheer size of the crowd imposing penalties upon the rate at which visual information is rendered. Seeing as the positions of all agents are computed on the device and stored into OpenCL buffers, they would normally have to be copied back and forth between host and device in order to render them, generating a large number of memory transfers. The solution is to combine the GPU computing and the GPU rendering by using an interoperability mode between OpenCL and OpenGL.

Combining computation and visualization

In order to be able to utilize the GPU for both GPGPU computation and as a traditional rendering pipeline, we must avoid unnecessary memory transfers while switching between operating modes, since the general purpose computation and the rendering process basically use the same data, namely particle positions.

This calls for an interoperability solution between OpenCL and OpenGL that can be achieved through a special data structure called a Vertex Buffer Object (VBO) and which allows for OpenGL data manipulation by OpenCL, without the need to transfer data back and forth (Figure 2). The VBO is an extension for OpenGL intended to improve performance by providing benefits of vertex arrays and display lists while avoiding downsides of their implementation. VBOs allow vertex array data storage in high-performance graphics memory on the server side and efficient data transfer. Using VBOs, the number of function calls and redundant usage of shared vertices can be reduced.

TEST SCENARIOS

In this section we present in detail certain test scenarios that were executed to validate and evaluate our solution. The experiments were carried out by varying different elements in the scene such as the number of agents, the number and size of obstacles or simply by creating some special scenarios.

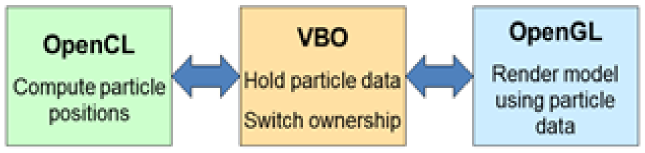

Figure 2-OpenGL/OpenCL interoperability.

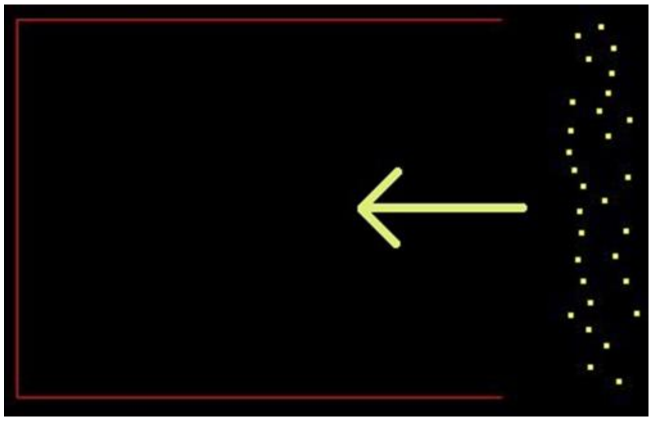

Figure 3 - Experiment 1

Besides the proper functioning of our application, we wanted to emphasize the differences in performance that appear when executing just on the CPU executing on hybrid CPU/GPU architectures, the differences being quite notable. In order to be able to run these tests, we implemented a version of our application that runs entirely on the CPU, this way avoiding all the GPU configuration overhead.

Testing with regard to the number of agents

The first type of test and one of the most important for noticing the differences in performance was done by increasing the number of agents in the scene.

The number of agents has the greatest impact on application performance, because agents are the main actors and all processing is done around them.

Except for computation for splitting the scene in a grid of cells and computation strictly regarding the scene, which are quite few, all other computation is done in order to compute forces, velocities and new positions of each agent. Thus, a huge impact on overall performance was to be expected.

In order to best emphasize the performance gain obtained on GPUs, besides observing the simulated scenario and the evolution of the simulation, we ran the simulation both on the CPU and on the GPU and compare the results. The first test consists of a simple simulation scenario with few agents placed on the right side (Figure 3). We computed the time required for all agents to migrate to the left side of the scene.

After several successive runs using both application versions, a significant increase in processing time is noticed on the CPU-based one. Table 1 shows the computed execution times.

Number of agents	CPU execution time	CPU+GPU execution time
10	3 s	5 s
100	6 s	5 s
250	8 s	6 s
500	14 s	7 s
750	43 s	8 s
1000	63 s	9 s

Table 1 - Execution times

These test easily prove the performance gain obtained by parallelizing the application. We will also compare performance results between the two versions of the application in subsequent tests.

Testing with regard to scene partitioning

When dividing the scene into a grid of cells in order to speed up the neighbor searching procedure, the number of cells (and thus their size) can vary, depending on scene complexity and size, or number of processed agents. There is no general formula to determine the optimum number of cells, thus we shall try to determine them empirically.

When using smaller cell sizes (Figure 4) we obtain the advantage of processing only a small part of the scene when computing agent interactions. Due to the fact that a smaller region surrounding each agent is taken into account, the number of neighboring agents is relatively small, thus generating less computation. Since computation for each agent is handled by a single GPU thread, this should count pretty much.

On the other hand, when using a larger cell size (Figure 5), even if the neighborhood is larger, the data structures holding hash info is considerably smaller. Thus, searching for neighboring cells as well as neighboring agents is faster, which should account for a performance gain, even if the number of neighbors for each agent is larger than in the previous case.

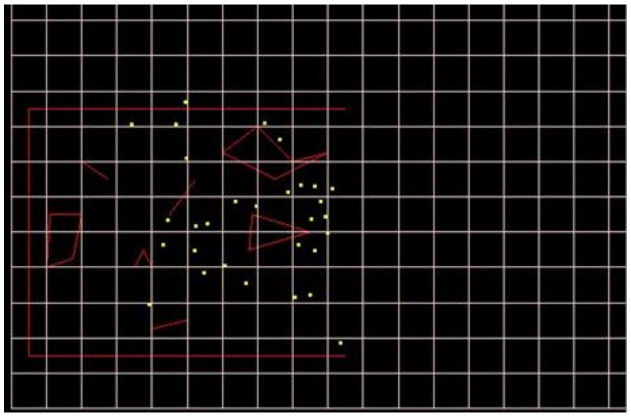

Figure 4 - Experiment 2 - Small cell size

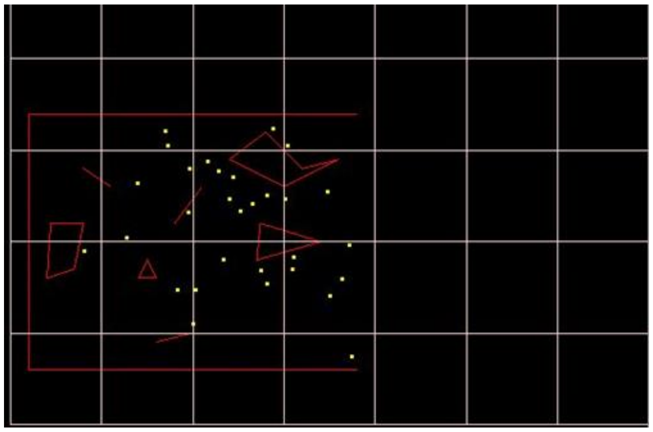

Figure 5 - Experiment 2 - Large cell size

Experiments showed that the best performance results are obtained for a balanced partitioning of the scene, which means that the relative dimension of the cells with regard to the scene should be chosen in a way as to ensure that the entire scene is covered by approx. 50-100 cells. Even if for a small number of agents the differences are not obvious, for large number of agents, this will impact on execution times.

Testing with regard to agents' trajectories

The third types of tests were carried out in order to validate the correctness of the implemented model and of interactions between simulation elements. This was carried out by generating the agents in several ways and varying their objectives.

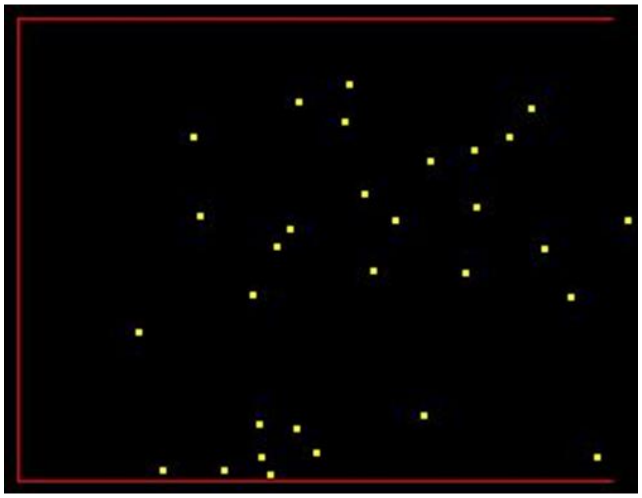

Figure 6 - Experiment 3-1 - Random agent generation

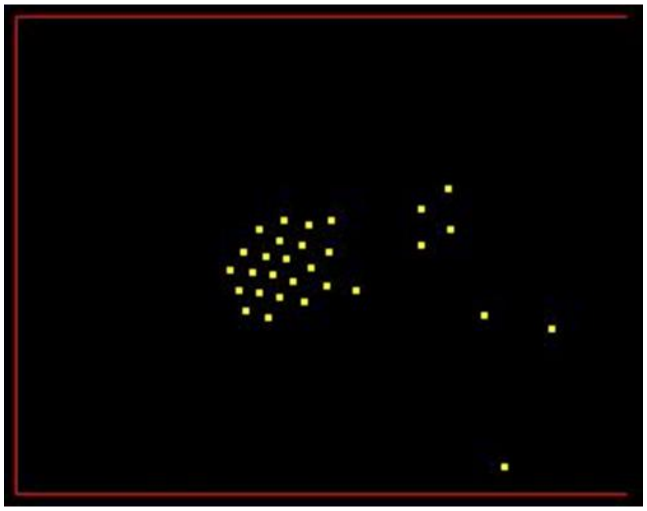

Figure 7 - Experiment 3-2 - Clustered agents

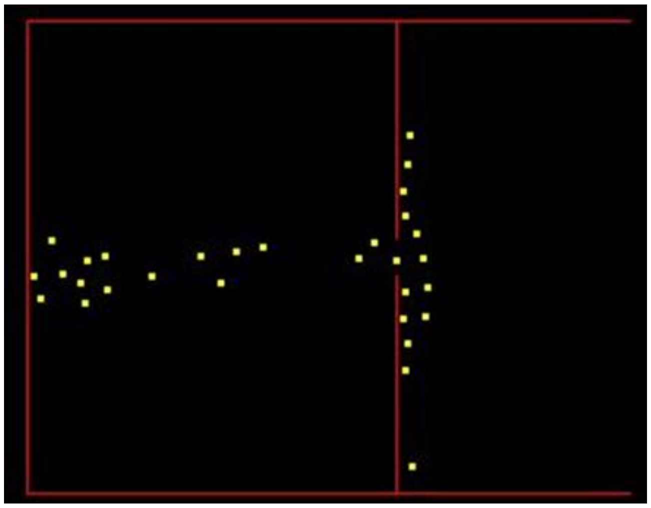

Figure 8 - Experiment 3-3 - Agents forced through a small opening

For the first test, we generated all agents at random positions in the scene with random objectives (Figure 6). Although the movement in the scene was chaotic, no interaction problems were detected.

For the second test, the agents were programed to cluster in the middle of the scene (Figure 7) in order to test their behavior in a crowded environment, but without supplementary difficulties posed by obstacles, with many agent-agent interactions. The simulation was once again without problems, although a small decrease in performance was noticed when all agents were in close proximity.

For the third test, all agents were forced through a tight opening (Figure 8). This test extends the previous one, but this time with difficulties posed by obstacles. The agents' behavior was consistent with the previous test.

It is worth mentioning that all three tests were carried out on both the CPU and the GPU version of the application and the results were consistent with performance measurements in the first experiment, namely the simulation time for the CPU version increases considerably with the number of agents, while for the GPU version, the increase in execution time is considerably smaller.

Testing with regard to scene complexity

Last but not least, we followed the impact that the scene

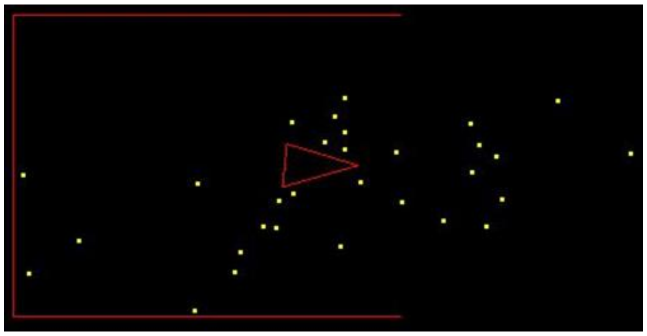

Figure 9 - Experiment 4 - Scenario 1

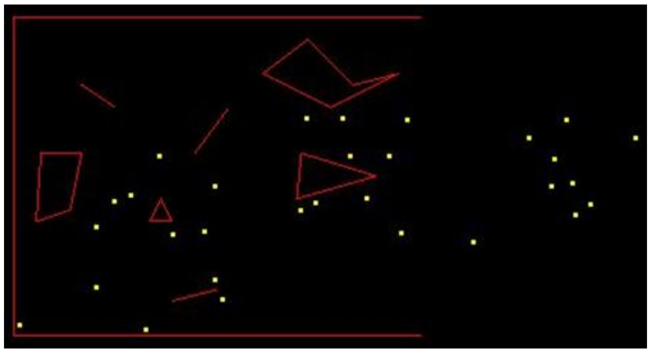

Figure 10 - Experiment 4 - Scenario 2

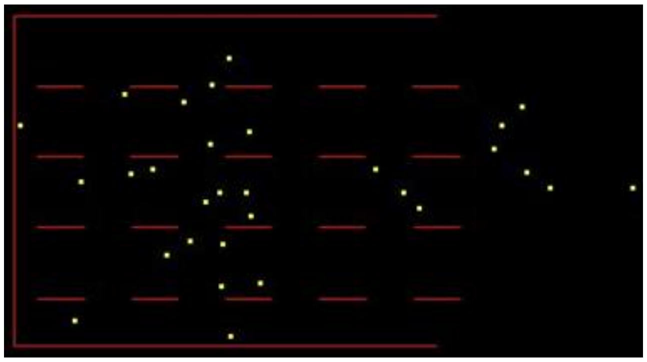

Figure 11 - Experiment 4 - Scenario 3

complexity has on overall simulation performance. Several scenarios were generated, ranging from simple ones, with 1 or 2 obstacles (Figure 9) to complex ones containing a much larger number of obstacles of different shapes (Figure 10, Figure 11). All agents were generated in the same positions for all scenarios in order to observe just the influence of scene complexity.

Results show that even if the scene is far more complex, there is an insignificant increase in simulation times when compared to differences in times for the previous experiments, when we were varying the number of agents.

CONCLUSIONS

Agent-based models are one of the best methods for realistic simulations of a complex environment or system which are usually non-linear and for which no simple and intuitive solutions exist that can offer precise results. Even so, the complex computation during the simulation remains an issue, due to the fact that the complexity of the studied system leads to heavy calculations in order to obtain results close to the real world.

Simulation techniques based on GPUs are an excellent solution to these problems, due to their parallel architecture, capable of executing hundreds and thousands of difficult operations at once. Test results showed that there is a considerable improvement in performance when using at least a GPU for simulations. However, as the complexity of the studied systems increases, a single GPU ceases to be sufficient, thus requiring more powerful architectures such as GPU clusters in order to carry out these simulations.

This paper presented key concepts for designing a real-time crowd dynamics simulation and visualization system that works in a hybrid CPU/GPU architecture, as well as having the potential to be extended for multiple GPU equipped nodes.

The simulation system that was presented proved the advantages that such a hybrid CPU/GPU architecture can have over traditional CPU-based architectures. Test results show significant improvements in all simulated scenarios, with the most significant one occurring when the number of agents was large. This is good news, since, with these simulations, the main entities are the ones that matter and that are wished to be present in a large number. This confirms that the approach presented is a promising one for interactive simulation of large crowds. Future development plans include porting the application to a GPU cluster in order to accelerate the computation process even further and to allow the user to interact in real-time with the simulated scenario.

ACKNOWLEDGMENTS

The research has been carried out in the Computer Graphics and Interactive Systems Laboratory of the Computer Science Department, in the Technical University of Cluj-Napoca, and partially supported through PN-II-PT-PCCA-2013-4 project funded by MEN-UEFISCDI, Contract no. 344/2014, PECSA - Experimental High Performance Computation Platform for Scientific Research and Entrepreneurial Development.

REFERENCES

1. Iain D. Couzin, Jens Krause, Richard James, Graeme D. Ruxton and Nigel R. Franks. 2002. Collective Memory and Spatial Sorting in Animal Groups. *Journal of Theoretical Biology, vol. 218, no. 1*.
2. Erin J. Hastings, Jaruwan Mesit and Ratan K. Guha. 2005. Optimization of Large-Scale, Real-Time Simulations by Spatial Hashing. *In Proceedings if the 2005 Summer Computer Simulation Conference 37 (4), 9-17*.
3. Dirk Helbing and Peter Molnar. 1995. Social force model for pedestrian dynamics. *Phys. Rev. E 51, 4282*. http://dx.doi.org/10.1103/PhysRevE.51.4282.
4. Anders Johansson, Dirk Helbing and Pradyumn K. Shukla. 2007. Specification of the Social Force Pedestrian Model by Evolutionary Adjustement to Video Tracking Data. *Advances in Complex Systems, vol. 10*.
5. Mark Joselli, Jose Ricardo Da Silva and Esteban Clua. 2014. An Architecture for Real Time Crowd Simulation Using Multiple GPUs. *In Proceedings of the 2014 Brazilian Symposium on Computer Games and Digital Entertainment (SBGAMES)*.
6. Taras I. Lakoba, D. J. Kaup and Neal M. Finkelstein. 2005. Modifications of the Helbing-Molnár-Farkas-Vicsek Social Force Model for Pedestrian Evolution. *Simulation, vol. 81, no. 5*.
7. NVIDIA. CUDA. 2015. Retrieved July 21, 2015 from http://www.nvidia.com/object/cuda_home_new.html.
8. Alessandro Pluchino, Cesare Garofalo, Giuseppe Inturri, Andrea Rapisarda and Matteo Ignaccol. 2014. Agent-Based Simulation of Pedestrian Behaviour in Closed Spaces: A Museum Case Study. *Journal of Artificial Societies and Social Simulation 17 (1) 16*.
9. Adrian Sabou, Dorian Gorgan and Ioan Radu Peter. 2014. Interactive Particle-based Simulation of Sociophysics. *In Proceedings of the 2014 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*.
10. Andreas Schadschneider, Ansgar Kirchner and Katsuhiro Nishinari. 2002. CA Approach to Collective Phenomena in Pedestrian Dynamics. *In Proceedings of the 5th International Conference on Cellular Automata for Research and Industry*.