Simulation of Large Crowds Including Gaseous Phenomena

Nicolas Courty, Soraia Musse

Simulation of Large Crowds in Emergency Situations Including Gaseous Phenomena

Nicolas Courty
SAMSARA – University of Bretagne Sud
Campus de Tohannic – 56000 Vannes – France
Nicolas.Courty@univ-ubs.fr

Soraia Raupp Musse
CROMOS – PIPCA – Unisinos
Sao Leopoldo – Rio Grande do Sul – Brazil
soraiarm@exatas.unisinos.br

Abstract

Crowd animation and simulation have been widely studied over the last decade for many purposes : populating collaborative virtual environments, entertainment and special effects industry and finally simulating behaviors and motion of people in emergency situations for safety systems. This last topic is addressed in this paper. We propose an original enhancement of a well known physics-based animation model which allows to consider influence of gaseous phenomena such as smoke or toxic gases in the behavior of the crowd. In order to get real time performances we also propose an implementation of this framework on modern graphics hardware, which allows to simulate crowds of thousands individuals at interactive framerate.

1 Introduction

Crowd behavior and motion of virtual people have been studied and modeled in computers with different purposes. Virtual crowds can populate collaborative virtual environments to increase their credibility. Different approaches have been proposed in order to animate crowds for the entertainment industry. Finally, research on safety systems, where crowds are used to simulate behaviors of people in emergency situations, provide useful tools that can help the design of buildings and open-spaces. This last topic is addressed in this paper. While a lot of previous works focus on the behaviors of escaping crowd, few works have also dealt with potential risks that occur from smoke or gases-filled environments. Though, it has been often observed that during a fire, smoke can be as deadly as fire [15]. Moreover, most people are not aware of the dangers of prolonged inhalation of smoke, and this constitutes another motivation for building simulation systems that mix crowd simulation with smoke or gas propagation. We believe that such systems can help designing fire safety installations.

We thus propose in this paper an original enhancement of a well known physics-based animation model (Helbing's model [13]) which allows to consider gaseous phenomena such as smoke or toxic gases in the behavior of the crowd. The choice of a macroscopic model for crowd simulation (instead of a behavioral, individuality-based system) derivates from the typical emergency situations we are dealing with. Such situations include a lot of people in panic state, where individual specific behaviors tend to disappear. It has also appeared that an efficient simulation system should include possibilities of interaction with the crowd, so that a potential user may dynamically change the conditions of the simulation. For such a system to be interactive, it is necessary to provide real time performances for large crowds and smoke or gas simulation. Therefore we present as a second major contribution of this paper an implementation of our framework based on modern graphics hardware, which permits simulations of high density crowds at interactive framerate.

The remaining of this paper is organized as follow: first we review related works. In section 3, we present our simulation engine, then its implementation on the GPU (section 4). Finally, section 5 is dedicated to experimental results and conclusion.

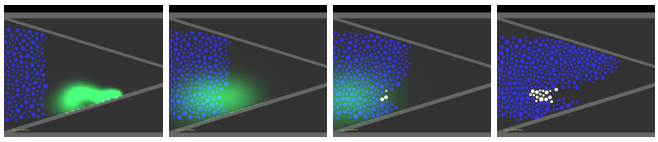

Figure 1. A toxic gas is spreading while people are trying to escape. People in white have absorbed a deadly dose of gas.

2 Related Works

Some authors have discussed how to simulate virtual crowds. Reynolds [25] described a distributed behavioral model for simulating flocks of birds formed by actors endowed with perception skills with many purposes. In fact, the birds (or boids) maintain proper position and orientation within the flock by balancing their desire to avoid collisions with neighbors, to match the velocity of neighbors and to move toward the center of the flock. Reynolds work shows realistic animation of groups by applying simple local rules within the flock structure.

Tu and Terzopoulos [29] have worked on behavioral animation for creating artificial life, where virtual agents are endowed with synthetic vision and perception of the environment. The repertory of fishes behaviors relies on their perception of the dynamic environment, and the fishes reactions are not entirely predictable because they are not scripted. Bouvier [3] used particle systems adapted for studying crowd movements where human beings are modeled as an interactive set of particles. The motion of people is based on Newtonian forces as well as on human goals and decisions. They introduced the concept of decision charges and decision fields modeled by using notions of the so called decision charges of a person, interacting with a surrounding decision field in the same way an electric charge is influenced by an electric field.

Brogan and Hodgins [5, 6] have used dynamics for modeling the motion of groups with significant physics. They reproduced movements of legged robots, bicycle riders and point-mass systems based on dynamics, considering a collision avoidance algorithm, which determines the desired position for each individual, given the locations and velocities of the visible creatures and obstacles. Indeed, a perception model to determine creatures and obstacles visible to each individual in the group precedes the displacement algorithm. Recently, same authors proposed a method to explore an approach for generating reactive path following based on the users examples of the desired behavior. The examples are used to build a model of the desired reactive behavior. The model is combined with reactive control methods to produce natural 2D pedestrian trajectories. Then the system automatically generates 3D pedestrian locomotion using a motion-graph approach [21].

Musse and Thalmann presented a hierarchical model to describe crowds with different levels of control: from guided to autonomous ones [22]. The behavior of crowds is based on rules dealing with the information contained in the groups of individuals. Helbing [13] proposed a model based on physics and socio-psychological forces in order to describe the human crowd behavior in panic situations. This model has been extended in order to include different individualities in the particle systems providing groups behavior, which is attained as an emergent function of local interactions between individuals [4].

In this paper, we used the original Helbing Model [13] as a basis. In order to get real time performances, we chose to exploit the computational power of the latest graphics boards. Though first designed for computer graphics industry, graphics processing units (GPU) have revealed over the last years to be high performance computing platforms at low cost. With their increased programmability possibilities and an higher precision arithmetic processing, it is now possible to consider execution of non-graphic applications on such boards. The reasons for their high performances on data-parallel computations comes from their *streaming* architecture, which allows several data to be processed at the same time according to the same computation *kernel*, and their dedicated *high-speed memory*. As a consequence, the resolution of a dynamic equation for crowd simulations is then well adapted to this process, as far as many individuals are likely to be treated at the same time. Several researchers have already explored those possibilities for a wide variety of computationally expensive problems: image and volume processing, sparse matrix and multigrid solvers, algebra operator or physics simulations [8, 16, 26, 11, 1]. Those examples demonstrate the effective computation power of those units. Recently, some authors have started to use GPU in an animation context (particle engine) [14, 17].

3 Simulation Engine

Helbing [13] proposed a generalized force model based on socio-psychological and physical forces in order to describe the human crowd behavior in panic situations. This model generates realistic phenomena as arc formations in exits or increasing evacuation time with increased desired velocities. This model is based on a particle system and involves different components: how to reach the desired destination, how to avoid other particles and how to avoid collisions with the environment.

Those three elements are combined into a classical Newtonian equation involving masses m_i of the N members of the crowd, thus designing a dynamical system. Provided that a given person i wants to move into a particular direction $\mathbf{e_i}$ with a desired velocity v_i^d, this person tends to adapt its own velocity v_i within a certain characteristic time τ_i. At the same time, this person tries to keep a velocity-dependent distance from the other pedestrians j, and elements of the environment W (such as walls). The resulting forces represent *interaction forces* that will be respectively noted $\mathbf{F_{ij}}$ and $\mathbf{F_{iW}}$ in the remaining. For each iteration, the acceleration for each person is given by the equation:

$$\frac{d\mathbf{v_i}}{dt} = \frac{v_i^d \mathbf{e_i} - \mathbf{v_i}}{\tau_i} + \frac{\sum_{j(\neq i)} \mathbf{F_{ij}}}{m_i} + \frac{\sum_W \mathbf{F_{iW}}}{m_i} \quad (1)$$

In our implementation, solving this dynamical system is performed through the use of the *Verlet Leapfrog* integration scheme, which permits an accurate integration while decreasing the importance of the choice of the time step [24]. Moreover, comparing to the traditional Verlet integration algorithm where positions for three consecutive time steps are required, it only requires to store one set of positions and one set of velocities for each member of the crowd, which is interesting for performance purposes. Let us note $\mathbf{A} = \frac{d\mathbf{v_i}}{dt}$ the acceleration computed from equation 1. The following equations defining the algorithm allow us to compute the velocity half a step ahead the current step n ($n + \frac{1}{2}$), and the new position $\mathbf{r_{n+1}}$ of the person at the next step:

$$\mathbf{v_{n+\frac{1}{2}}} = \mathbf{v_{n-\frac{1}{2}}} + \mathbf{A}\delta t \tag{2}$$

$$\mathbf{r_{n+1}} = \mathbf{r_n} + \mathbf{v_{n+\frac{1}{2}}}\delta t \tag{3}$$

Let us finally note that if the velocity for the current time step is required, it can be computed as:

$$\mathbf{v_n} = \frac{1}{2}(\mathbf{v_{n+\frac{1}{2}}} + \mathbf{v_{n-\frac{1}{2}}}) \tag{4}$$

The following chapters present an original approach to integrate gaseous phenomena inside Helbing's model. We first present some general considerations about behaviors of people involved in emergency situations including smoke, then we present some modifications to equation 1 and finally a way to deal with injuries caused by inhalation of smoke or toxic gases.

3.1 Behaviors of people *wrt.* gaseous phenomena

In most cases, people are afraid by fire, and tend to escape from such situations as fast as they can. Considering gaseous phenomena such as smoke, it has recently appeared that people have a biased idea of the danger induced by traveling through smoke and inhalating for a long period their toxic components. For instance, during the world trade center disaster, it has been observed that a lot of people would enter smoke-filled staircases and travel through smoke for extended amounts of time (see [23] for details). Moreover, when dealing with invisible or inodorous gases, most of people are not aware of the presence of toxic components in the air, and behave normally. In this context, it is somehow difficult to design a pertinent psycho-social force that modify people's behavior. We decided to take into account two parameters: a repulsion force which is expressed as a function of the surrounding smoke concentration, and an health status that modify the velocity of member of crowds.

3.2 Designing new psycho-physical forces

In order for the crowd to react to smoke, a force $\mathbf{F_s}$ that influence motions of the members of the crowd should be computed. We decided to design this force as a function of the surrounding concentrations. The expression for this force can be written as a function of the gradient of the concentration T of the smoke:

$$\mathbf{F_s} = A(\nabla T) \tag{5}$$

where A is a scalar parameterizing the force. In its discretized form, this equation can be written as:

$$\mathbf{F_s} = A(T_{i-1,j}(t)\mathbf{u_{1,0}} + T_{i+1,j}(t)\mathbf{u_{-1,0}} + T_{i,j-1}(t)\mathbf{u_{0,1}} + T_{i,j+1}(t)\mathbf{u_{0,-1}})$$

where $\mathbf{u}_{\alpha,\beta}$ is a unit vector (α, β). Figure 2 is a graphical representation of the computation of this force.

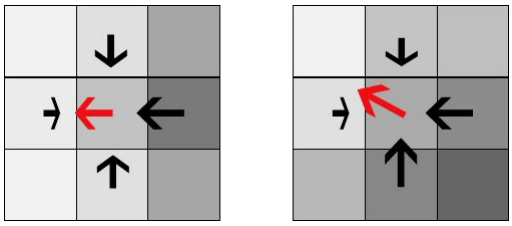

Figure 2. Computing forces from the smoke based on the concentration information. A linear interpolation between 4 unit vectors is performed, weighted by concentrations of surrounding cells.

The resulting acceleration $\frac{\mathbf{F_s}}{m_i}$ is finally added to equation 1. As a result members of the crowd are not able to go through areas where the concentration of smoke is too elevated. Moreover, they are repulsed in the direction of propagation of the smoke. A person inside the smoke may therefore have an erratic behavior, but will tend to go out of the smoke by going in the areas of lesser concentration. An illustration of this behavior is given in figure 3.

3.3 Injuries caused by smoke or gas inhalation

Smoke inhalation stands for a significant cause of death in fire victims [15]. To account for the different symptoms of the possible injuries induced by smoke or toxic gases, an health status has been added to individuals that constitute the crowd. This parameter H_i represents a global health status of the person i, varying from 1 (good health) to 0 (death). This parameter decrease over time until 0 with respect to the concentration of smoke present in their location. This can be modeled as the following equation:

$$H_i(t + \delta t) = H_i(t) - \beta T(t)\delta t$$

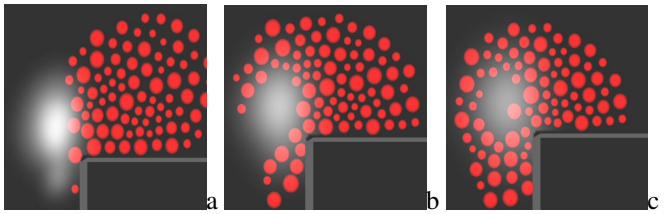

Figure 3. A group of individuals trying to find its way through smoke (a) the group is stopped by smoke (b) some of them are trying to avoid the smoke (c) finally, while the smoke is dissipating, some of them manage to go across.

where β is a scalar regulating the toxicity with respect to the concentration. This parameter can account for different types of chemical products for instance.

In order to take into account effects from prolonged smoke or gas inhalation (such as headache, disorientation and global weakness), we simply modify the desired velocity with respect to the health status parameter:

$$v_i^d(t+\delta t) = -\gamma H_i(t) v_i^d(t) \quad (6)$$

where γ is a scalar regulating the importance of the health status in the modification of the velocity. As a result, people that have been in contact for a long period with the smoke have a slower way to travel through the environment (see figure 4 for an example).

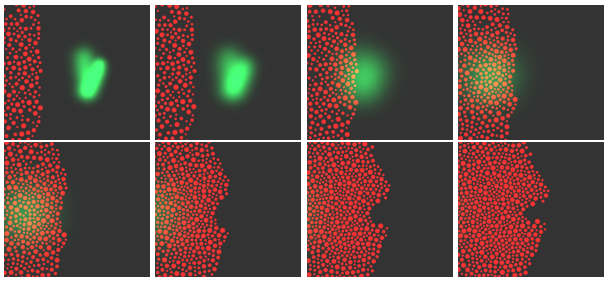

Figure 4. People that have stayed for a long period in the gas have diminished capabilities and are moving slower than others, which explains the end "forked" shape of the crowd.

4 Using graphics hardware to simulate the crowd

In this section, we first present an overview of the resolution of the equation onto the graphics card (section 4.1). We then present the way to handle environment-dependent forces (section 4.2), the way to compute inter-agents forces section 4.3), and finally the gaseous phenomena simulation (section 4.4).

4.1 Overview

The resolution of this dynamical system can be related to a technique used to perform cloth simulation developed by Green [10]. It uses *Render-To-Texture* capabilities of modern graphics hardware, through the use of *pbuffer*, which are off-screen rendering buffers. Those buffers can contain float elements, *i.e.* they can be considered as arrays of 4-dimensional vectors of float values. Let us note that those buffers can be as well considered as input textures for rendering. In order for data to be processed, a textured quad is rendered in the float pbuffer, exactly fitting the size of the viewport. This results in a direct correspondence between elements of the texture and the pbuffer. Hence, elements of the texture (input) are used to compute a new value (output) stored in the pbuffer. This computation is performed onto the graphical processing unit via a simple program named *fragment shader*. This technique has been widely used for general purpose computation on graphics boards (a good review can be found in [9]). When multi-texturing is allowed, several textures can be used as input in this calculus. In our case, the crowd is represented as float textures containing information about individuals involved in the crowd. Figure 5 shows a representation of those textures. For dynamical data such as position (changing over time), the texture is in fact associated to a float pbuffer (in which new positions are computed). The resolution of the dynamical system is composed of three different rendering steps as shown in figure 6. The first step computes acceleration for each member of the crowd. The two velocities pbuffer are required to compute the acceleration allowing to reach the

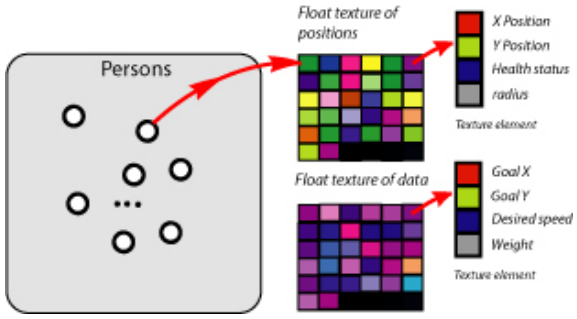

Figure 5. Representations of the crowd as float textures. In this figure, two float textures are represented : one is related to positions, the other to personal data.

goal (first term of equation 1). It also uses the current position pbuffer, as well as input textures data (as described in next sections). From this acceleration and the previous velocities ($\mathbf{v_{n-\frac{1}{2}}}$), the next velocities ($\mathbf{v_{n+\frac{1}{2}}}$) are computed (step 2 in figure 6). Finally the new positions are computed within the integration process described in equation 3 (step 3). At the end of this multi-step rendering process, the previous and next velocities pbuffers are swapped (*ping-pong buffering*), as well as the current and next positions pbuffers. The next parts of this section explain how environment and individuals inter-dependent forces are computed and taken into account in the computation of acceleration.

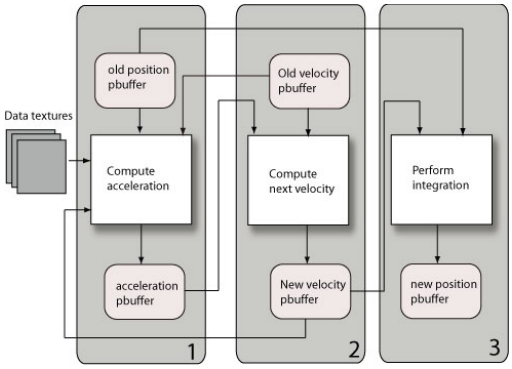

Figure 6. The three rendering steps for the resolution of the dynamical system. Arrows stand for "used as input in".

4.2 Computing environment dependent forces

Environment dependent forces stand for all the forces exerted by the environment onto the agent, thus modifying its current direction. They usually represent repulsion forces from walls and obstacles. Those forces, noted $\mathbf{F_{iW}}$ in equation 1, are expressed in Helbing's model as:

$$\mathbf{F_{iW}} = (Aexp^{B(r_i - d_i)} + \lambda K(r_i - d_i))\mathbf{d_{iW}} \qquad (7)$$

where $\mathbf{d_{iW}}$ is a normalized vector in the opposite direction of the obstacle. A, B, and K stand for scalar values parameterizing the repulsion forces. λ is a function equal to zero if $d_i > r_i$, or one otherwise. Hence, the last term of the sum is taken into account for regions next to walls.

Computing for each iteration those forces for each person represents one of the bottleneck of this type of simulation. Given an environment with a lot of walls, one has to consider special algorithms to determine which are the walls that potentially has an influence over one person's motions, and that is not acceptable when dealing with large crowds. For those reasons, we decided to precompute those forces as a *force field* represented in a float texture on the GPU memory.

Representing the force field as a texture. In order to do so, a discretization of the environment along a regular grid is performed. In each point of the grid the sum of all the forces from every walls is computed. The resulting bi-dimensional vector is stored as a texture unit in a float texture, in the *red* and *green* components. Though, as stated in equation 7, the forces depend on the radius of the crowd members. To cope with this problem, an approximation of the exact force is performed. We compute the force field for the minimum and maximum radius, and store the two force fields in one unique texture (on the 4 *rgba* components). At runtime, in the fragment shader program responsible for the computation of acceleration, the exact position of the person allows to compute a look-up value in this texture (*i.e.* which texel corresponds to the current position of the person). Then, given its radius, a linear interpolation between the *rg* and *ba* components is performed.

Obstacles and dynamic environments In order to increase potential interaction with the crowd, it can be useful to design a set of tools to add/remove obstacles, or even move dynamically those obstacles within the environment. In our framework, those types of interaction are possible. Two types of obstacles have been designed, based on the same methodology: moving and dynamic obstacles. Moving obstacles are of arbitrary form and can be displaced according to user's interaction. At the beginning of the simulation, the force field of those obstacles are generated as a float texture. Those texture are then blended with the environment force field texture *wrt.* the position defined by the user. The result of this blending is computed on the CPU and uploaded to the graphics card memory. Dynamic obstacles represent obstacles that have a particular behavior along time during simulation. As an example, let's consider a turnstile turning around its axis. It exists two methods to describe this behavior along time. First, it is possible to compute before the simulation all the force fields corresponding to this animation (their number depends on the desired precision and in our case symmetry factors). The second method (that may be more time consuming) consists in computing at each frame the corresponding force field. The advantage of using such a method for small obstacles comes from the gain in memory occupation. At runtime, the corresponding force field is composed with the environment texture following the blending method described above. Figure 7 shows an example of those dynamic obstacles (a turnstile).

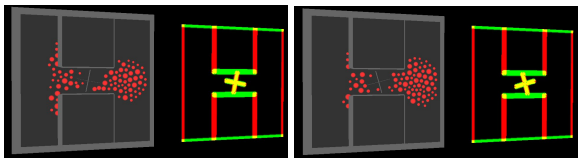

Figure 7. Dynamic obstacle: a turnstile is regulating the flow of people within a corridor. Right parts of both pictures is the corresponding environment force field float texture (with false coloring)

4.3 Computing inter-agents forces

Those forces are applied by agents on other agents. Its equation is similar to the equation 7. They usually represent repulsion forces between individuals. Those forces, noted $\mathbf{F_{ij}}$ in equation 1, are expressed in Helbing's model for two individuals i and j as:

$$\mathbf{F_{ij}} = (Aexp^{B(r_{ij}-d_{ij})} + \lambda K(r_{ij} - d_{ij}))\mathbf{n_{ij}} \qquad (8)$$

In this case, r_{ij} is the sum of the radius of the two individuals, *i.e.* $r_{ij} = r_i + r_j$, d_{ij} is the distance between those two individuals, and $\mathbf{n_{ij}}$ a normalized vector pointing from pedestrian j to pedestrian i.

In terms of complexity, computation of all those forces normally requires an $o(n^2)$ algorithm. This problem, known as the *n-body forces problem*, has been widely studied. However, it is possible to note that it exists a radial cut-off in the definition of the force: at a distance $d_{ij} > 2$ meters, the value of the force is almost zero. In this configuration, algorithm such as the *Barnes-Hut* algorithm [2] (which allows an $o(n\ log\ n)$ complexity) are not optimal. In our case we developed two methods that have a globally linear complexity.

CPU-based method The first method is entirely executed on the CPU. It consists in a discretization along a regular grid of the environment. Each element of this grid contains a double-chained list of the individuals of the crowd inside it. For a given person, one has to compute the forces for all the individuals inside the same case, and as well a given number of adjacent cases. The step of discretization determines the number of adjacent cases that have to be considered. At each iteration, the position on the grid of the different pedestrians has to be updated. This technique ensures that for a given person, no more than k individuals will have to be considered, hence resulting in an $o(k\ n)$ complexity. When all the forces have been computed, a float texture is generated with those forces, and passed as input in the acceleration fragment program.

GPU-based method It is also possible to render all the individuals with a quad textured by their respective "force field" (as for the environment). When blending is on, the sum of the forces is computed at each point of the grid through this rendering step. Afterward, in the fragment shader that computes acceleration, a correct lookup function makes it possible to get the resulting force.

Comparison After experimenting those two methods, we have come up with the conclusion that the gain provided by a GPU implementation was not obvious in every case. Hence, the second method's complexity depends upon the resolution of the grid. We plan to publish detailed benchmarks of those methods and comparison with existing methods such as [14] in subsequent publications.

4.4 Gaseous phenomena simulation

Recently, there have been some works allowing to simulate and visualize smoke in real time [7, 12, 18, 19]. In [12, 18, 19], the overall optimization comes from the use of the GPU. Hence, to provide an interaction tool with the crowd based on the propagation of the smoke, we decided to implement on the GPU a very simple diffusion model combined with a simple advection scheme, already described in [12]. A major advantage comes from the fact that all the computations take place onto the graphics board, which avoids time-consuming memory transfer to CPU, and runs effectively in combination with our architecture. Let us finally note that our model does not guarantee a physically correct propagation of gas. We designed this very simple model to test the interactions between gas and the crowd, but it would be more consistent to use a model respecting the Navier-Stokes equation (including complex boundary conditions). Those aspects are part of the extensions of our work.

Diffusion In this context, in order to compute the diffusion of smoke within the environment, the diffusion for a cell $T_{i,j}(t)$ is given by:

$$T_{i,j}(t + \delta t) = T_{i,j}(t) + \frac{c_d}{4}\nabla^2 T_{i,j}(t)\delta t \qquad (9)$$

where c_d is the coefficient of diffusion, and $\nabla^2 T_{i,j}(t)$ the Laplacian of the concentration, which can be written in its discretized form as:

$$\nabla^2 T_{i,j}(t) = T_{i-1,j}(t) + T_{i+1,j}(t)$$
$$+T_{i,j-1}(t) + T_{i,j+1}(t) - 4T_{i,j}(t)$$

Using a similar method described in [12], a simple fragment shader is built which computes diffusion of the smoke within a grid of a given resolution. In order for the smoke

to diffuse *wrt.* the environment, a simple texture of the environment is built with only one component per texel which indicates whether a wall is present or not. Then, inside the corresponding fragment shader program, this texture is addressed to know if the neighboring values are walls or not.

Advecting gas field It has appeared that it could be of interest to be possible to advect the gas field *wrt.* some velocity fields in order to simulate ventilation systems. In this case, the gas density at a given position $\mathbf{r}$ must be displaced along the grid according to a velocity field $\mathbf{U}$ (in our case, constant over time). This can be simply expressed as:

$$T_{(\mathbf{r}+\mathbf{U}_\mathbf{r}\delta t)}(t+\delta t) = T_\mathbf{r}(t)$$

To solve this equation, we use an implicit method described in [27] and implemented for GPU in [12]. This method consists in inverting the problem, and expressing the new density T at a given cell with position $\mathbf{r}$ as:

$$T_\mathbf{r}(t+\delta t) = T_{(\mathbf{r}-\mathbf{U}_\mathbf{r}\delta t)}(t) \qquad (10)$$

In case of discrete grid (such as in a GPU implementation), the quantity $T_{(\mathbf{r}-\mathbf{U}_\mathbf{r}\delta t)}$ is obtained via bilinear interpolation with surrounding cells.

5 Implementation and Performance

In this section, the tools used for the implementation of our framework are presented, and results about the performances of our system are presented.

5.1 Rendering crowds

Displaying large crowds can be a difficult problem when dealing with thousands of individuals. Hence, displaying articulated virtual humanoids composed with around 1000 polygons can only bring to crowd populated with less than one thousand if interactive rate is a major constraint. A new set of technique has recently been developed by Tecchia *et al.* [28] based on impostors. Though the results are very goods and look attractive, it is still difficult to handle crowd with more than $10K$ individuals. We decided therefore to use techniques that are traditionally used in particle rendering. This technique is similar to [14, 17] and shall not be described in this paper. Though, let us note that those techniques are using vertex shaders capabilities, which run in an effective way with our architecture.

5.2 Performance study

In order to implement our library, we used the CG toolkit [20] for the fragment shaders, in combination with the NVIDIA pbuffer class. The computer used for tests was an Athlon XP $2500+$ equipped with a GeForce 5900. The library was written in C++ with OpenGL for interfacing with the video card. When considering interactions with smoke, our system proves to be much faster than a CPU optimized version, as shown in figure 8. Those results where obtained within a complex evacuation scene, where the computation of smoke was performed on a 692×256 grid, including three dynamical obstacles. A picture of this simulation can be seen in figure 9. As a conclusion, it is possible to animate and interact with crowd composed by more of ten thousands individuals at interactive rates.

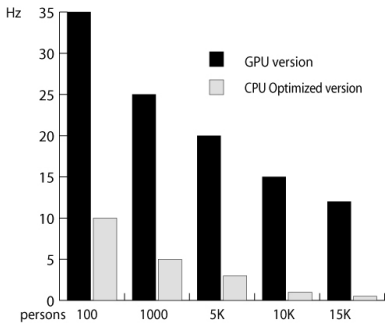

Figure 8. Comparison between our implementation and a CPU optimized one on a complex scene with smoke and interactions with dynamic obstacles.

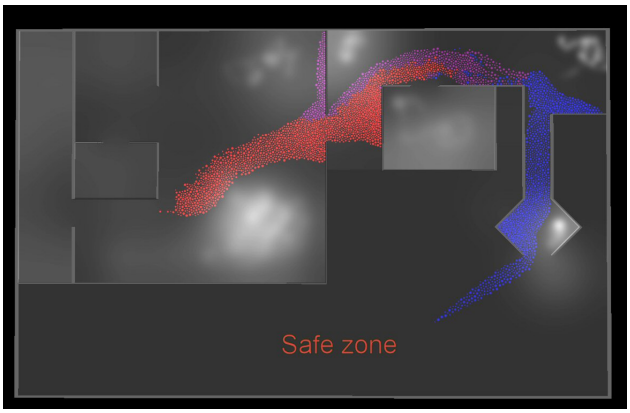

Figure 9. A complex evacuation scene (under smoke conditions) populated by 4000 individuals within 3 different groups, running at 25 fps. Groups are following precomputed paths in the environment.

6 Conclusion and Future Works

This paper presents an original model allowing to take into account gaseous phenomena in crowd simulation. Based on Helbing's model, our framework integrates new forces which account for difficulties that people may encounter while traveling through smoke-filled environments, and also the possible harm caused by smoke. In order to provide interactive performances for our simulation, a GPU implementation of our framework is also presented, and allow to consider crowds constituted with several thousands of people in smoke-filled environments within real time. Moreover, this characteristic allows interactions with the crowd, which constitutes a plus for potential applications. Our model stands for, to our knowledge, the first attempt to mix crowd and gas simulation, and though the results seem quite attractive, we still need to perform more validations, notably by comparing our results to real cases.

References

[1] C. H. A. Lefohn, J. Kniss and R. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. *Proc. of IEEE Visualization 2003*, 2003.

[2] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324(6270):446–449, 1986.

[3] E. Bouvier, E. Cohen, and L. Najman. From crowd simulation to airbag deployment: particle systems, a new paradigm of simulation. *Journal of Electronic Imaging*, 6(1):94–107, Jan. 1997.

[4] A. Braun and S. R. Musse. Modeling individual behaviors in crowd simulation. In *Proc. of Computer Animation and Social Agents (*CASA'03*) (New Jersey, USA)*, May 2003.

[5] D. Brogan and J. Hodgins. Group behaviors for systems with siginificant dynamics. Technical Report 95-18, Georgia Institute of Technology. Graphics, Visualization and Usability Center, 1998.

[6] D. Brogan, R. Metoyer, and J. Hodgins. Dynamically simulated characters in virtual environments. *IEEE Computer Graphics and Applications*, 18(5):58–69, Sept./Oct. 1998.

[7] R. Fedkiw, J. Stam, and H. Jensen. Visual simulation of smoke. In E. Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, pages 15–22. ACM Press / ACM SIGGRAPH, 2001.

[8] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys. A multigrid solver for boundary value problems using programmable graphics hardware. In W. Mark and A. Schilling, editors, *Proc. of ACM SIGGRAPH/Eurographics Conf. on Graphics Hardware (EGGH-03)*, pages 102–111, Aire-la-ville, Switzerland, July 26–27 2003. Eurographics Association.

[9] GPGPU. General purpose computation on GPUs. http://www.gpgpu.org, 2004.

[10] S. Green. NVIDIA cloth simulation. http:// developer.nvidia.com/ object/ demo_cloth_simulation.html, 2003.

[11] M. Harris, W. Baxter, T. Scheuermann, and A. Lastra. Simulation of cloud dynamics on graphics hardware. In W. Mark and A. Schilling, editors, *Proc. of the ACM SIGGRAPH/Eurographics Conference on Graphics Hardware (EGGH-03)*, pages 92–101, Aire-la-ville, Switzerland, July 2003. Eurographics Association.

[12] M. Harris, G. Coombe, T. Scheuermann, and A. Lastra. Physically-based visual simulation on graphics hardware. In *Proc. of the 17th Eurographics/SIGGRAPH workshop on graphics hardware (EGGH-02)*, pages 109–118, New York, Sept. 1–2 2002. ACM Press.

[13] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407(1):487–490, 2000.

[14] P. Kipfer, M. Segal, and R. Westermann. Uberflow: A GPU-based particle engine. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Graphics Hardware (EGGH-04)*, pages 115–122, 2004.

[15] N. Kirchner and H. Savolainen. Triage of fire smoke intoxicated victims in a disaster situation. *The Internet Journal of Rescue and Disaster Medicine*, 1(2), 2000.

[16] J. Krueger and R. Westermann. Linear algebra operators for gpu implementation of numerical algorithms. *ACM Trans. on Graphics (TOG)*, 22(3):908–916, 2003.

[17] L. Latta. Building a million particle system. In *Proc. of Game Developers Conference(GDC-04)*, 2004.

[18] W. Li, X. Wei, and A. Kaufman. Implementing lattice boltzmann computation on graphics hardware. *The Visual Computer*, 19(7-8):444–456, Dec. 2003.

[19] Y. Liu, E. Wu, and X. Liu. Real-time 3d fluid simulation on gpu with complex obstacles. In *Proc. of IEEE Pacific Graphics 2004*, Seoul (Korea), Oct. 2004.

[20] W. Mark, R. Glanville, K. Akeley, and M. Kilgard. Cg: a system for programming graphics hardware in a c-like language. *ACM Trans. on Graphics (TOG)*, 22(3):896–907, 2003.

[21] R. Metoyer and J. K. Hodgins. Reactive pedestrian path following from examples. *The Visual Computer*, 20(10):635–649, 2004.

[22] S. R. Musse and D. Thalmann. Hierarchical model for real time simulation of virtual human crowds. In *IEEE Trans. on Visualization and Computer Graphics*, volume 7(2), pages 152–164. IEEE Computer Society, 2001.

[23] G. Proulx and R. Fahy. Account analysis of WTC survivors. In *Proc. of the 3rd Int. Symp. on Human Behaviour in Fire*, Belfast, Sept. 2004.

[24] O. Rapaport. *The Art of Molecular Dynamic*. Cambridge University Press, New-York, 1996.

[25] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics:* Proc. of SIGGRAPH '87, 21(4):25–34, July 1987.

[26] A. Sherbondy, M. Houston, and S. Napel. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. *Proc. of IEEE Visualization 2003*, 2003.

[27] J. Stam. Stable fluids. *Computer Graphics*, 33(Annual Conference Series):121–128, 1999.

[28] F. Tecchia, C. Loscos, and Y. Chrysanthou. Image-based crowd rendering. *IEEE Computer Graphics and Applications*, 22(2):36–43, Mar./Apr. 2002.

[29] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *Proc. of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 43–50. ACM SIGGRAPH, ACM Press, July 1994.