GPU Generation of Large Varied Animated Crowds

Isaac Rudomin[1], Benjamín Hernández[1,2], Oriam deGyves[3], Leonel Toledo[3], Ivan Rivalcoba[3], and Sergio Ruiz[2]

[1]Barcelona Supercomputing Center, Spain

[2]Tecnológico de Monterrey, Campus Ciudad de México, Mexico

[3]Tecnológico de Monterrey, Campus Estado de México, Mexico

{isaac.rudomin, benjamin.hernandez}@bsc.es, {A00465730, ltoledo, A01167172, sergio.ruiz.loza}@itesm.mx,

Abstract. We discuss several steps in the process of simulating and visualizing large and varied crowds in real time for consumer-level computers and graphic cards (GPUs). Animating varied crowds using a diversity of models and animations (assets) is complex and costly. One has to use models that are expensive if bought, take a long time to model, and consume too much memory and computing resources. We discuss methods for simulating, generating, animating and rendering crowds of varied aspect and a diversity of behaviors. Efficient simulations run in low cost systems because we use the power of modern programmable GPUs. One can apply similar technology using GPU clusters and HPC for large scale problems. Such systems scale up almost linearly by using multiple nodes. One must combine parallel simulation and parallel rendering in the cluster with interaction and final rendering in lighter clients. However, in view of the latest developments such as the new family of mobile multicore chipsets and GPU-based cloud gaming platforms, the pieces are almost there for this kind of architecture to work.

Keywords. Simulation, real-time crowds, rendering and animation.

Generación de grandes multitudes animadas y variadas en el GPU

Resumen. En el artículo se presentan los pasos para simular y visualizar multitudes masivas variadas y animadas en tiempo real, usando el procesador gráfico (GPU). En particular, se discutirán los métodos para la simulación de comportamientos, nivel de detalle, animación y generación de personajes variados. Dada la arquitectura de estas técnicas, se pueden extender a clusters de GPU o en sistemas de cómputo de alto rendimiento (HPC). Estos sistemas son escalables casi linealmente si se incrementa el uso de nodos, sin embargo se deben combinar técnicas de simulación y rendering paralelos. Sin embargo, dados los avances tecnológicos recientes como plataformas de cloud gaming, estas técnicas están listas para funcionar en dichas plataformas.

Palabras clave. Generación, simulación, animación, visualización, multitudes, tiempo real.

1 Introduction

The development of graphical models and the incorporation of intelligent behavior made it possible to create virtual environments with many intelligent characters interacting. The use of these types of models can be seen in many applications. One example is videogames—where one can design more challenging games with many characters of different physical appearance and each with its own behavior. Another area where these models can be used are in the design of strategies for people entering or leaving an environment or structure—a stadium, an auditorium, an entertainment center, even in case of emergency. Also, they are used for simulation of vehicular traffic for construction of the new roads taking into account behavior of the drivers.

Similar methods can be applied to other social or health applications involving agents (epidemics, infection) and for financial agent based applications.

The problem is that crowd simulations need a scalable multi-agent system architecture that simultaneously supports the simulation of hundreds of thousands (or millions) of complex autonomous agents and the rendering of all those agents as diverse, animated characters, even in crowded scenes, yet achieving good frame rates.

Today there is no simple process for simulating and visualizing large and varied crowds in real time for consumer-level computers. Animating varied crowds using a diversity of models and animations (assets) is complex and costly. One would require models that are expensive if bought, take a long time to model, and anyway consume way too much in memory and computing resources. As solutions to these problems, we have developed methods for simulating and animating crowds of varied aspect and a diversity of behaviors. They can be used for simulations for PCs with consumer level graphic cards (GPUs). Efficient simulations run in low cost systems because we use the power of modern programmable GPUs.

One can apply similar technology using GPU clusters and HPC for large scale problems. Such systems scale up almost linearly by using multiple GPUs. In what follows we review our methods for crowd simulation, level of detail rendering, generation and animation of varied character families, and what we do in the way of parallelizing for larger systems.

2 Simulation and Collision Avoidance

A key component in every crowd simulation is the synthesis of behavior; for a crowd simulation will be accurate in the measure it is able to faithfully emulate both individual and group human behavior. Collision avoidance refers to the anticipated movements with which an agent avoids colliding with other agents present in their environment.

There is extensive research focusing on collision avoidance behaviors for virtual agents. The main methods are:

- “Flocking”, by Craig Reynods [1], uses three rules: separation alignment and cohesion to steer the agents. This can be done using forces for each rule and summing the effects.
- “Social forces”, by Dirk Helbing [2, 3], which is very similar to the previous method in that it uses forces, including forces for avoiding other pedestrians, others for avoiding walls, etc.
- Reciprocal Velocity Obstacles [4] applies ideas from robotics to crowd simulations: it finds velocities that will cause collisions and avoids them. Planning takes place in velocity space.

All these methods require data from the nearest neighbors of each agent. Exhaustive proximity queries can be prohibitive as the number of agents increases, since the naïve methods are $O(n^2)$. Researchers [5, 6] have explored several techniques and data structures to lower the complexity of proximity queries, which effectively reduce this to $O(n\log n)$.

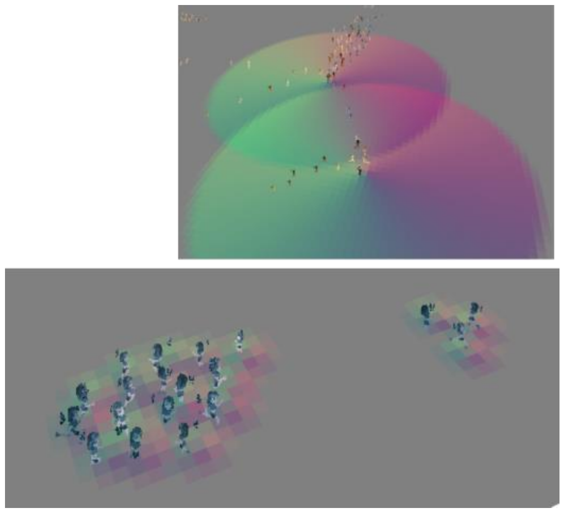


Fig. 1. Using world space maps to encode forces that attract or repel agents from each other

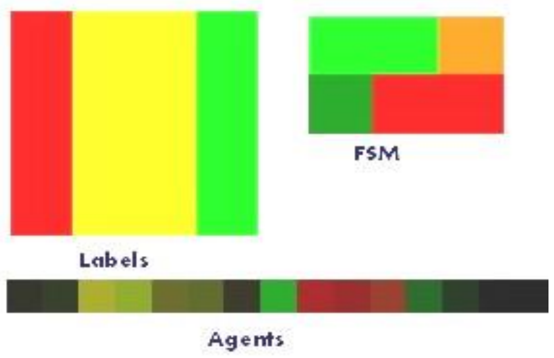


Fig. 2. In a simplified version of the method, three textures are used. A FSM texture, a world texture specifying labels and an agent texture where state and position are encoded

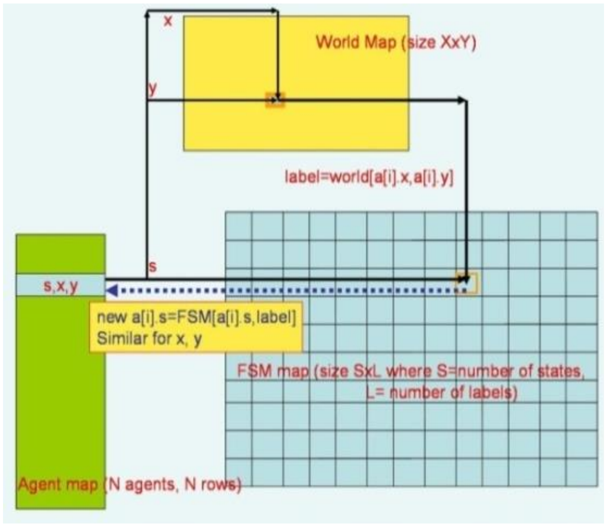
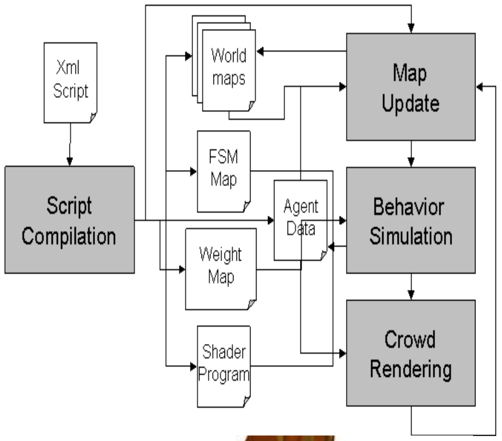

x
World Map (size XxY)
y
label=world[a[i].x,a[i].y]
s
s,x,y
new a[i].s=FSM[a[i].s,label]
Similar for x, y
FSM map (size SxL where S=number of states, L= number of labels)
Agent map (N agents, N rows)

Fig. 3. Simulation process consists of consulting the three textures from Fig. 1

2.1 Encoding Neighbor Information on World Maps

Another way to lower the complexity of simulations, however, is to use a world-space map to encode the information in the environment rather than the agents and thereby reduce the problem of determining neighbor contributions to $O(n)$. Some methods [7, 8] encode the environment with information guiding the agents,



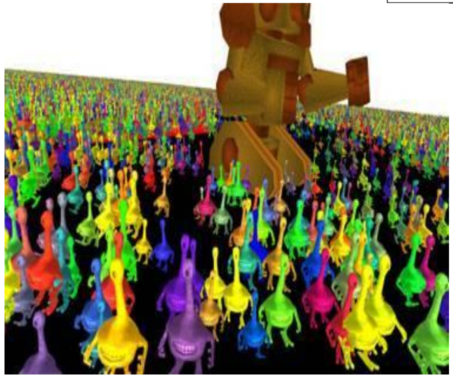


Fig. 4. XML scripts are translated to GLSL code for simulator and auxiliary maps; it is a more complex scenario

but then treat these agents as simple particles following the forces encoded in the environment.

We explored a more general method in [9]. Here the agents encode (paint) information in world space images representing the environment (as can be seen in Fig. 1). The agents, however, use this information exerting their own goals and perceptions since behaviors are specified as state machines (also stored as images). The latter and world and agent space images are used to determine collision avoidance and other steering behavior. In particular, attraction and repulsion forces in Fig. 1 are added to achieve behavior similar to that achieved by Reynolds and Helbing.

Other images in agent space codify agent state. In a simplified version (Fig. 2 and 3) we use a Label map as the single world space map. We codify state and position in an agent map. Then, given state and position, and by consulting color

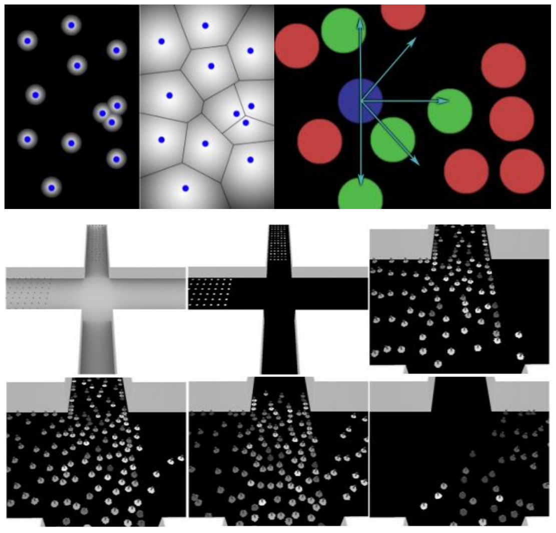


Fig. 5. Voronoi-like method for collision avoidance (top) and Voronoi + RVO example (bottom)

in the appropriate position in the label map, we determine new state and positions from the FSM map.

In general, we can use several auxiliary images in world space to codify obstacles, heights, gradients. This became rather cumbersome, so in latter work, and for easier authoring, maps and finite state machines were codified in XML and translated to GLSL shaders (Fig. 4 top). This allowed us to generate more complex scenarios (Fig. 4 bottom).

2.2 VL Path Planning

Realistic crowd simulations face several problems in order to achieve more accurate results; one problem is collision avoidance. We have studied a parallel technique using the graphics hardware to avoid all collisions between agents by efficiently finding the nearest neighbors in a crowd simulation and calculating the response.

It is basically an image-based technique similar to the one used in the previous section, but here, instead of painting and adding different forces, every agent paints an area of a given radius that codifies the distance to the agent. When these areas overlap, the distance to the closest agent is kept. As a result, this texture is a truncated Voronoi Diagram (Fig. 5 top left) which other agents can read to move through the environment (Fig. 5 top right). A truncated Voronoi Diagram can be described as follows. Given a set of *S* seed points, divide a plane into |*S*| tiles such that all points '*p*' inside a tile are closer to a particular seed than to any other seed and are within a predefined radius, called tile radius '*r*', per seed, using the seed coordinates as a center.

Like other similar techniques that use painting of influence areas, this technique can achieve simulations with thousands of agents in real-time (Fig. 5 bottom). But in contrast to methods that codify forces, our method in the previous section, agents do not get stuck.

To detect neighbors using the truncated Voronoi diagrams, we take samples in the local vicinity of the agents. To sample its neighborhood, an agent '*a*' performs a ray marching technique over the diagram to sense the viewing area in the direction of its movement. Fig. 5 (top right) shows a graphic representation of the ray marching process. The circles represent the agents in the simulation; the purple agent is the one currently looking for its nearest neighbors and the green agents are the nearest neighbors. The red agents are not considered by the purple agent because they are currently out of its viewing area.

This method is capable of simulating thousands of autonomous agents at interactive frame rates while performing accurate collision avoidance. It is important to note that the proximity queries are performed in the graphics hardware, which allows us to implement the collision avoidance also in the graphics card.

Following the previous idea, we have developed other two techniques for proximity queries that are suitable for simulating thousands of agents in real time. These techniques are gather-based and scatter-based approaches.
For the scatter technique, we follow the idea that an agent should be able to find its closest neighbors with only one texture fetch. To accomplish this, each agent paints in an environment map an area where it is visible. Each pixel in the environment map holds a list with the IDs of neighbor agents. We use a Layered frame buffer (LFB) to generate the nearest neighbor lists

and render the environment map for the scatter technique (Fig. 6a). LFBs are commonly used for Order Independent Transparency (OIT) and to our knowledge, have not been used for proximity queries or behaviors.

Contrary to the scatter technique, in the gather technique each agent only writes one pixel in the environment map. To find their neighbors, every agent scans over an area in the environment map. The purpose of scanning an area is for each agent to generate its own neighbor list. As in the previous technique, every agent paints in a texture its ID, but only on a single texture element. This eliminates the need to create a LFB. Then, every agent reads an area searching for its nearest neighbors. Fig. 6b shows the main difference between the environment maps created.

As the first method presented in this section, these two techniques are capable of simulating thousands of autonomous agents at interactive frame rates (Fig. 6c).

2.3 Data Driven Simulation

A recent trend is to try to validate models with real world observations, rather than by reproducing phenomena that seem to be right. Even Helbing *et al.* [10] have recently stated that force models, while successful, are not consistent with empirical observations and are hard to calibrate. They suggest a cognitive science approach based on behavioral heuristics. They develop a new model where guided by visual information, namely the distance of obstructions in candidate lines of sight, pedestrians apply two simple cognitive procedures to adapt their walking speeds and directions.

Others have done this perceptual or synthetic vision based steering [11], however, in most of these approaches each agent needs its own memory, and so these methods fail to scale up to several hundreds of thousands of agents and do it efficiently.

Our work in [9] using the world space maps performs efficient collision avoidance operations in the spirit of synthetic vision even for millions of agents, if instead of painting round areas, we paint areas with the shape of the appropriate perception cones. However, we have yet to

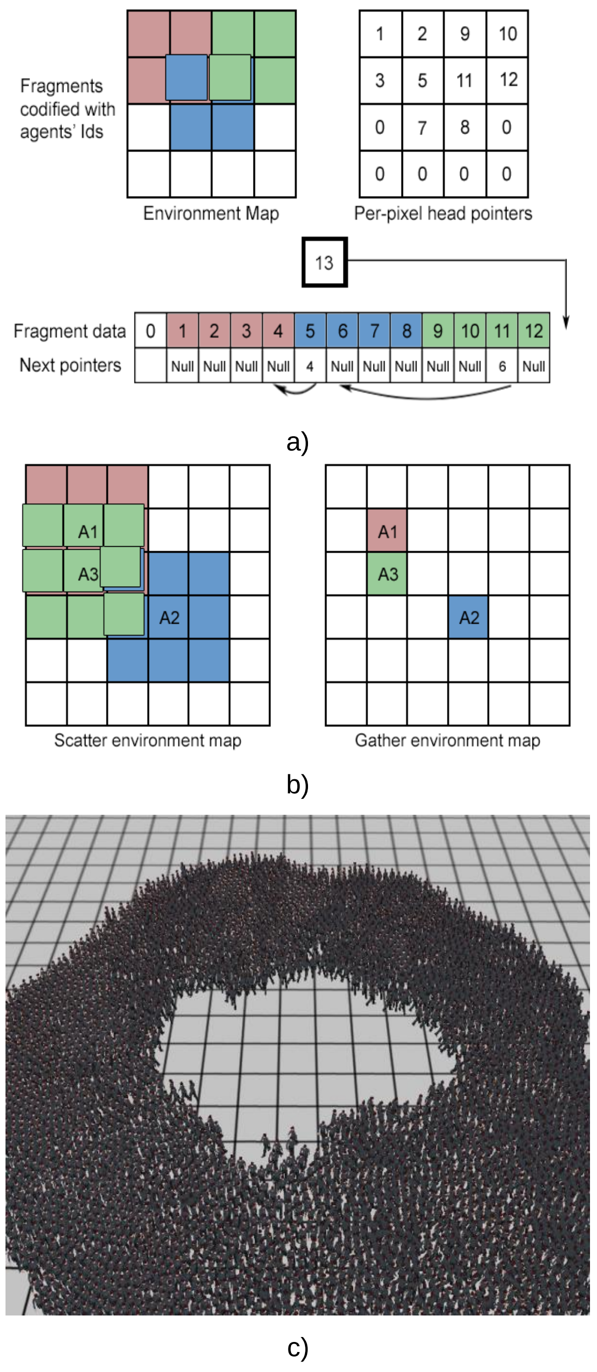


Fig. 6. a) Layered frame buffer used to generate a list of nearest neighbors. b) Comparison between the environment maps generated for scatter (left) and gather (right). c) Circle-4096 test. Agents initially in circle and have to get to their diametrically opposite position

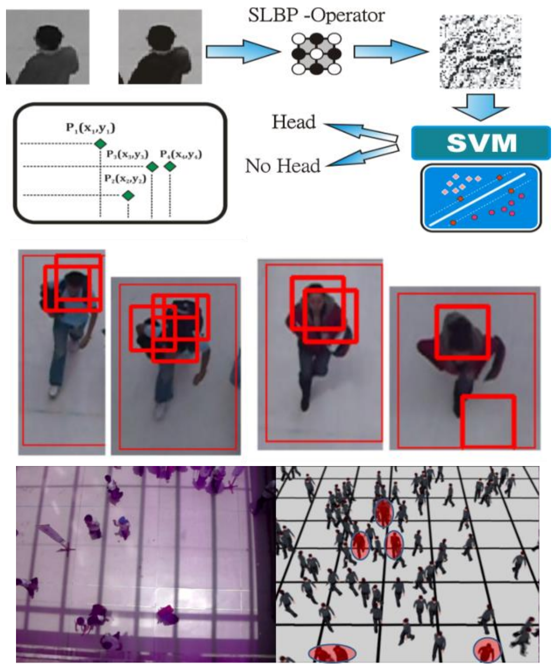


Fig. 7. Vision module combined with simulation (avatars in red, simulated agents in black)

validate this model with actual real-world behavior. Data driven simulation is fundamental for validating the behavior models of the agents. Currently, we are working in this. We have already coupled our simulation module with the vision module. The image involving the nucleus is first transformed from RGB color space to grayscale, then it is equalized and a multi-resolution histogram of SLBP [27] (semantic local binary patterns) is extracted to form a feature vector, then this vector is fed to a trained support vector machine (Fig. 7 top). The result is to label each image as having a head or not having a head (Fig. 7 center). Each detected head is tracked over every frame by a pyramidal version of the Lucas-Kanade tracker. Using these points and the simulation stage, we combine steering models with actual behavior (Fig. 7 bottom).

3 Navigation

Closely related to collision avoidance, navigation in a crowd simulation context refers to the ability of agents to effectively avoid collisions against objects within a scene while moving toward their goal. These obstacles may or may not move, but the criterion that classifies them as objects within a scene is the fact that obstacles do not avoid collisions: they do not present reactive behavior.

There is a phenomenon occurring when a crowd is observed; at near distances individual behavior arises while at far distances group human behavior can be noticed. This assumption may not imply that two types of simulations are required but describes human behavior when observed from different perspectives, as an accurate simulation would. Referring to Zhou *et al.* [12], this phenomenon can be interpreted on two levels:

– *Micro-level:* when experienced from a close distance, a crowd appears as a small-sized crowd regardless of its actual size, revealing fine entity traits.
– *Macro-level:* when viewed as a whole, from far distance, a crowd presents coarse, fluid-like traits.

Nevertheless, in practice, there are crowd navigation techniques which perform well at either micro- or macro-level. For example, classic path finding algorithms such as A* or Dijkstra's [13] has been used extensively at micro-scale level when the number of agents is low (in a number that a CPU can handle) and obstacles do not move. For a macro-scale level simulation, with higher agent numbers and the option of moving obstacles, several techniques model and solve the agent navigation problem, in the form of grid partitioning, formation and navigable areas or using video sequences as input.

Grid partitioning methods consist in dividing the navigable space into cells where according to different rules or mathematical models, agents are able to find their path to a given goal even if objects are present in the environment. An approach to grid partitioning is the use of cellular automata [14, 15, 16]. Cellular automata solve the collision avoidance and navigation problems with one algorithm, but it presents a lack of separation and control for individuals as they must follow the majority flow and the direction toward an exit. A navigation method similar to cellular automata –in the sense of grid partitioning of the navigable space– is the one based on vector fields; such method is able to produce a character motion flow for one agent which is responsive to user input [17].

Another option for navigation consists in defining navigable areas instead of a well-structured grid. Pettré *et al.* [18] opt for decomposing the navigable area into overlapping cylindrical cells with the aid of a Voronoi Diagram that will generate a Navigation Graph which provides path variety and batch processing for groups of pedestrians. It does not, however, support dynamic scenarios.

Video sequences also have been used as input to generate the motion of a virtual crowd [19]. It turns out that since video sequences capture reality, it is expected that the agents' navigation is shaped by the physical characteristics of the captured scenario.

3.1 MDP for Planning

As it was mentioned earlier, navigation techniques can simulate crowds at micro- or macro-level, so our approach, on which we are working now, will unify these levels of abstraction, thus agents will be able to take its own decisions which may affect the whole crowd, while group dynamics may affect the individual. Our first observation is that an agent, while moving through an environment, performs a sequential decision problem which when solved has to find its path from an origin to a destination following a set of additive rewards; this is usually called Markovian Decision Process (MDP). A layered-based method which solves a *semi*-MDP [20] may turn impractical, since it requires different layers encoding extra information of the navigable space. In cases where obstacles or other kind of objects are dynamically introduced, a new layer must be added which increments complexity to the whole solution. Another problem with the MDP formalism is that the state space grows exponentially with the number of domain variables, and its inference methods grow in the number of actions. Thus, in large problems, MDPs become impractical and inefficient.

In our case we have implemented a layer independent GPU-based approach. It uses MDPs, and an MDP is running as a preprocess to calculate multiple free-of-collision trajectories which agents in a crowd will follow (Fig. 8 *top*). Then on run-time, the MDP is adapted within a given radius, using a spatial kernel which encodes route alternatives inside a given radius, thus characters are able to avoid collisions against other agents or moving objects (Fig. 8 bottom).

Table 1. GPU- and CPU-based MDP comparison

System	Iterations	Optimal Policy
Nvidia Geforce GT445M	184	4.1s
Inter Core i7 1.87 GHz, single thread	10	1,500s After 10 iterations

Table 1 shows performance results obtained from calculating MDP in GPU and CPU for the test scenario shown in Fig. 8 top. In the case of the CPU-based MDP, it used only a single thread process and after 25 minutes the program calculated only 10 of 184 iterations needed to reach the optimal policy.

3.1 Lattice-Boltzmann Crowd Flow

For extreme densities of agents, intersection

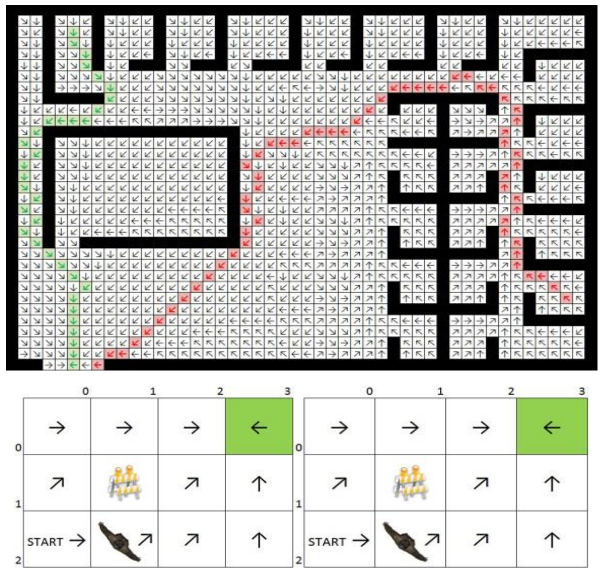


Fig. 8. MDP for crowd navigation and collision avoidance

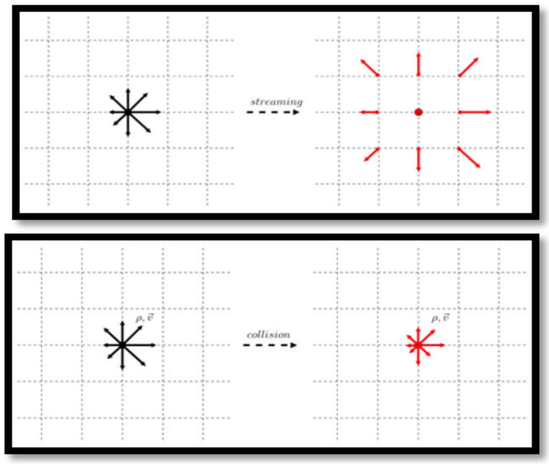


Fig. 9. Lattice-Boltzmann operations: stream (top), collision (bottom)

computation becomes expensive or our adaptive MDP approach might have unnecessary adaptations since different agents might be in the same cell. At this macro-simulation level, crowds mostly follow a general flow; only certain agents have to be simulated as such and most crowd movement can be simulated as fluids. We can use Lattice Boltzmann Method (LBM), which is a method for solving the Navier-Stokes equations using Lattice Gas Cellular Automata. It consists of discretizing the dominion into a set of connected sites (the lattice or mesh), with state variables defined at each site and update rules, based on local and neighbor information (*collision* and *streaming*). The generic LBM simulation algorithm follows the following steps:

– Each particle travels in a discrete direction. This is the discretization of velocity space.
– At each time step, the particles move along their assigned directions toward the next lattice point: they are streamed (Fig. 9 top).
– If more than one of these particles arrive simultaneously at the same lattice point, a collision rule is applied, redistributing the particles such that the conservation laws (Navier-Stokes for mass and momentum) are satisfied (Fig. 9 bottom).

We implemented LBM in CUDA as Single Component Single Phase (SCSP):

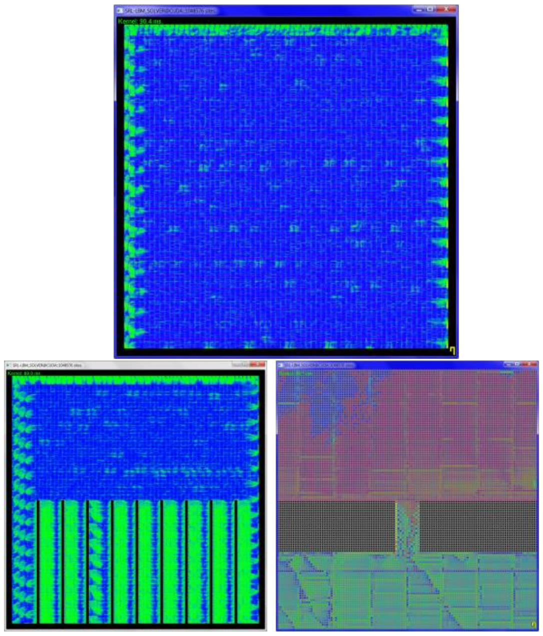


Fig. 10. LBM simulation without (top) and with (bottom) solid boundaries

```
foreach time step do
  CalculateMacroscopicVariables()
  DetermineNewEquilibriumDF()
  Collide()
  ProcessBoundaries()
  Stream()
End
```

Readers can find a detailed explanation of the algorithm in [21]. Implementation results of the algorithm for a Lattice of 1024x1024 cells the iteration time was 90.3 ms in a Nvidia Geforce GTX 560m GPU.

4 Level of Detail

Rendering many characters is a must for crowds. In what follows we explore several methods for level of detail that allow rendering crowds with many characters.

4.1 Uniform Crowd Impostor LOD

Impostor-based LOD in the GPU and instancing allows real time rendering of large crowds of

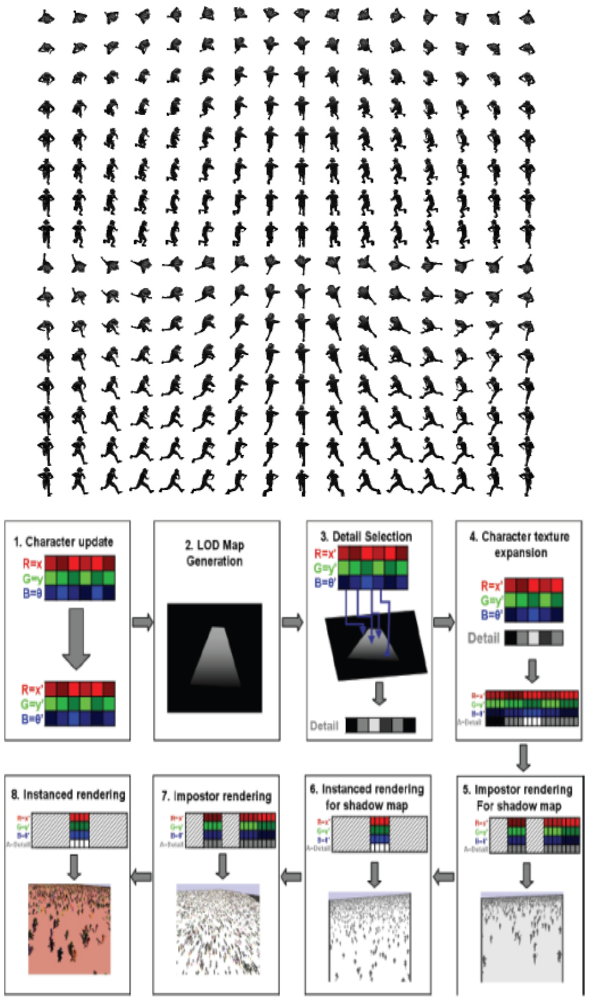


Fig. 11. Impostor images and process

similar characters. In Fig. 11 one can see a texture with images the same character in several poses and different camera angles. This is the impostor texture. When the character is far enough from the camera, instead of displaying the character's geometry, we display the image of the character closest in pose and camera angle.

4.2 Varied crowd discrete LOD

For varied crowd visualization, using impostors is too expensive, thus we do discrete LOD and View Frustum Culling in the GPU. In Fig. 12, this method is outlined. First, all necessary initializations are performed on the CPU. These

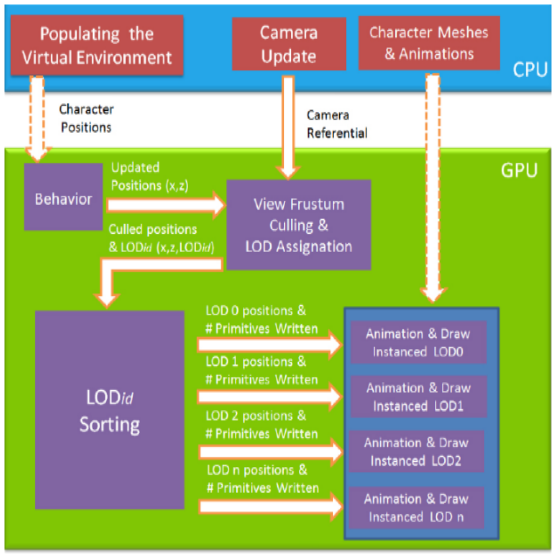

Populating the Virtual Environment
Camera Update
Character Meshes & Animations
CPU
Character Positions
Camera Referential
GPU
Behavior
Updated Positions (x,z)
View Frustum Culling & LOD Assignation
Culled positions & LODid (x,z,LODid)
LOD 0 positions & # Primitives Written
LOD 1 positions & # Primitives Written
LOD 2 positions & # Primitives Written
LOD n positions & # Primitives Written
LODid Sorting
Animation & Draw Instanced LOD0
Animation & Draw Instanced LOD1
Animation & Draw Instanced LOD2
Animation & Draw Instanced LOD n

Fig. 12. Discrete LOD in GPU

include loading information stored on disk (e.g., animation frames and polygonal meshes) and information generated as a preprocess (e.g., character positions) or in runtime (e.g., camera parameter updates).

This information is used on the GPU to calculate the characters' new positions, do view frustum culling, and assign a specific level of detail (LOD) for each character and for level of detail sorting and character rendering. The following is a brief description of each stage.

Populating the Virtual Environment and Behavior. In these stages we specify the initial positions of all the characters, how they will move through the virtual environment and how they will interact with each other. The result is a set of updated character positions.

View Frustum Culling and Level of Detail Assignation. In this stage we use the characters' positions to identify which characters will be culled. Additionally, we assign a proper LOD identifier to the characters' positions inside the view frustum according to their distance to the camera.

Level of Detail Sorting. The output of the View Frustum Culling and Level of Detail Assignation stage is a mixture of positions with different LODs. In this stage we sort each position according to its LOD identifier into appropriate buffers such that all the characters' positions in a buffer have the same level of detail.

Animation and Draw Instanced. In this stage we will use each sorted buffer to draw the appropriate LOD character mesh using instancing. Instancing allows us to translate the characters across the virtual environment and add visual and geometrical variety to the individuals that are part of the crowd.

4.3 Point-based LOD

Applying the GPU for discrete LOD and view frustum culling calculation results in a very small time penalty (~ 0.39 ms) [22]. Nevertheless, this approach is limited for the rendering step. Drawing the characters using polygons as their geometrical representation takes a considerable amount of resources in modern GPUs. It has been shown [23] that alternative rendering methods, such as hierarchical point-based methods (Fig. 13 top), improve rendering performance.

Our point-based LOD approach consists in using a point-based system combined with a hybrid hierarchical skeleton structure that allow us to create varied animated crowds, as well as to reduce rendering costs. The system takes as input any given model vertex, normal and UV parameterization to generate a hierarchical skeleton structure based on octrees (Fig. 13 bottom). To create that skeleton, containing volumes are calculated for every limb in the model. This way data from individual parts of the model are stored and can be used for rendering and animation. For rendering, each limb's data is reduced to create a point sample with fewer points; this was done on four levels, the last two levels have a single point that represents the whole limb and a single point for the whole character, respectively.

For animation, the process is the same, as the animation of the joints is applied to vertices or points on the appropriate limbs at any level of the associated octree, making it possible to animate any given character. For lower levels of detail, animations can be reduced by only the most important limbs, such as the whole legs or arms.

The system is divided into three stages. The first stage holds the highest level of detail: models are fully rendered using the complete geometry, as well as tessellation and displacement mappings for better rendering results. This is expensive and resource-consuming, so only very few models are selected within the crowd to be rendered at the first stage. The second stage is the result of using the octree structure: this process gives as a result different point samples for each model. In the last stage the world is divided in tiles and a quadtree is made with these tiles as leaves: agents in tiles or their quadtree parents that are far away from viewer are blended together to reduce the required rendering resources. Animation and collision avoidance computations can be skipped in these tiles.

5 Texture-Based Characters

Current character modeling and animation techniques require a great amount of work and time. If we want to model and animate each character of a crowd made of hundreds of characters, the artist must design or scan each model, rig each character, specify the source of the animation for each character (motion capture, forward kinematics, inverse kinematics, video sequences, among others) and render each character. We propose a better approach and have developed texture-based methods for modeling, rigging, skinning and animating varied crowds. These methods require the generation of a family of characters which has the same UV parameterization, and then a small set of textures is needed for rigging, skinning and animating a crowd made of thousands of characters.

5.1 Generation of Diversity

As crowds increase in complexity and size, more assets such as meshes and textures need to be designed and created; body templates alleviate the need to create more assets. Our approach consist in generation of crowds of characters visually and geometrically different using body-part templates which are combined together. These body-part templates or basis meshes represent a specific part of a body such as the head, torso, arms, or legs; these basis meshes and their textures were extracted from a character

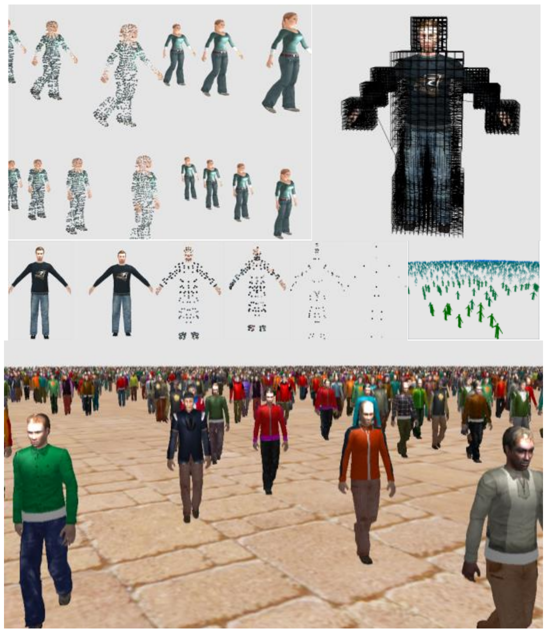


Fig. 13. Point rendering vs geometric models (top left). Hierarchical skeleton structure based on octrees (top right). Different Levels of Detail and coding (middle). Scene rendered using complete system (bottom)

data set. Then by combining the basis meshes and their textures, we generate families of novel characters according to specific features (sex, ethnicity, complexion or clothing) given by the user (Fig. 14 top).

Following Blanz' approach [24] where all facial models share the same parameterization space and thus form a subspace of morphable models, our character dataset and the character family generated from them also have this feature. The advantages of having this feature are explained in the next paragraph and following subsections.

To improve the visual variety of our characters we use a manual generated texture called anatomy image which is a gray scale image were brighter areas represent high levels of fat and darker areas represent low levels of fat, so we are able to generate different body complexities from skinny to obese characters (Fig. 14 bottom). The first advantage of having the family in full correspondence is that we can apply this anatomy image in different characters indistinctly thus eliminating the possibility of having one character for each character of the crowd.

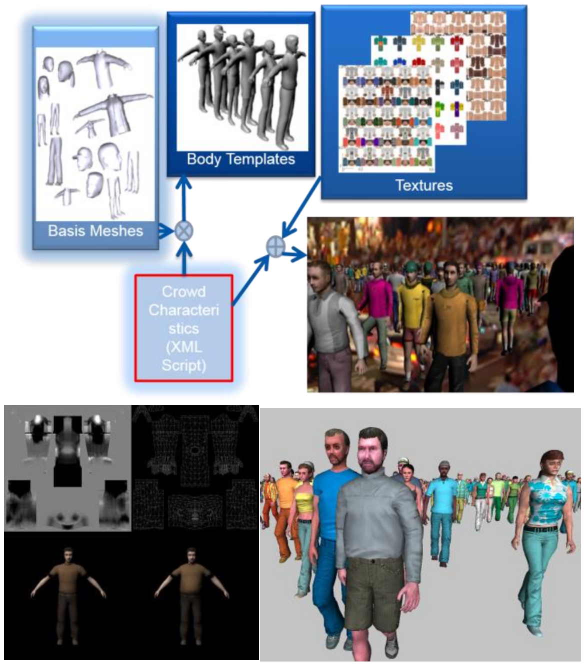


Fig. 14. Generation of Diversity (GOD): templates, fat/muscle displacement maps, results

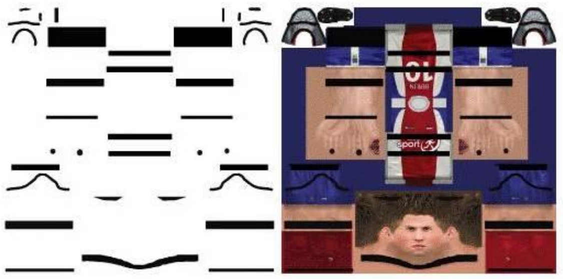


Fig. 15. Left. Joint texture. Right. Joint texture and its correspondence with the character texture parameterization

5.2 Texture Space Rigging

The animation of characters is usually done through the use of a skeleton, which is an articulated structure of segments and joints combined with information detailing how the surface geometry of the figure is bound to that structure. This is a time consuming process of animating a character, since an experienced artist

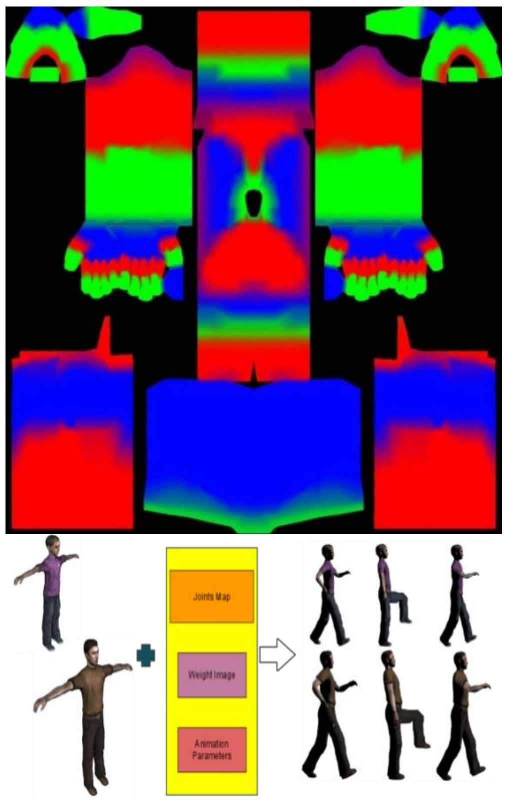


Fig. 17. Blending maps for skinning. Skinning two characters with aligned parameterizations: matching poses allow animation transfer

can spend several hours in this process. Therefore, conventional techniques can't be used to animate a crowd made of hundreds or thousands of different characters. Our alternative is to take advantage of the UV parameterization of the characters. If all of them have the same UV coordinates, and rigging and skinning information is stored in textures which correspond to that UV parameterization, we can reuse these textures in all the characters.

Fig. 15 (left) shows a texture used to calculate a skeleton of a character. Each joint area (dark zones) has a unique ID representing a body joint; notice that this texture must be in correspondence with the character's texture parameterization (Fig.15 right). To obtain the 3D pivot points for character's limb rotation, which when connected will conform the skeleton, the computation of the centroid at each joint area is performed by selecting and averaging all the vertices belonging to a given joint area.

As mentioned earlier, texture-based rigging can be transported from one model to another if texture parameterization is the same in both cases, i.e., between characters of the same family (Fig. 16).

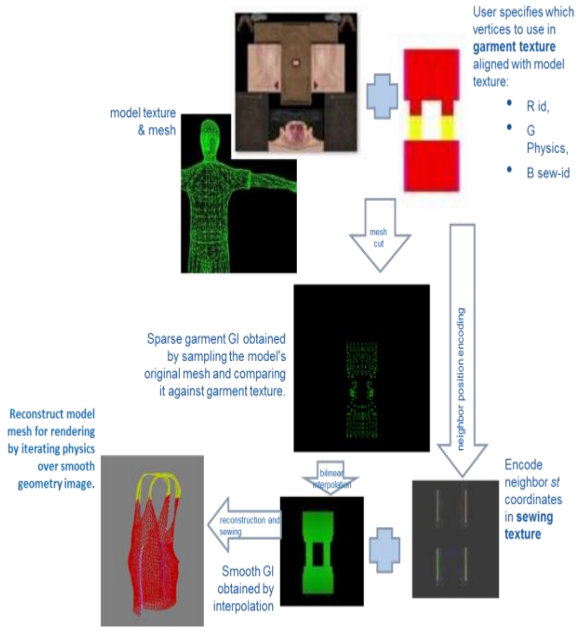

model texture & mesh
User specifies which vertices to use in garment texture aligned with model texture:
R id,
G Physics,
B sew-id
mesh cut
neighbor position encoding
Sparse garment GI obtained by sampling the model's original mesh and comparing it against garment texture.
Reconstruct model mesh for rendering by iterating physics over smooth geometry image.
bilinear interpolation
reconstruction and sewing
Smooth GI obtained by interpolation
Encode neighbor st coordinates in sewing texture

Fig. 18. Modeling clothing

5.3 Texture Space Skinning

The next step in character animation consists in specifying how the character's mesh will be attached to the skeleton, this process is called skinning. The idea behind this is to specify weight values used to modify the character mesh according to the skeleton pose with a reduced amount of artifacts that appear at joint sections. Thus, we encode these weights, which can be generated using authoring tools such as Maya, into a texture (Fig. 17 top).

Finally, the rigging is completed with animation key frames. Key frames can be generated via motion capture, forward kinematics, inverse kinematics, video sequences, among others. Our current implementation uses forward kinematics

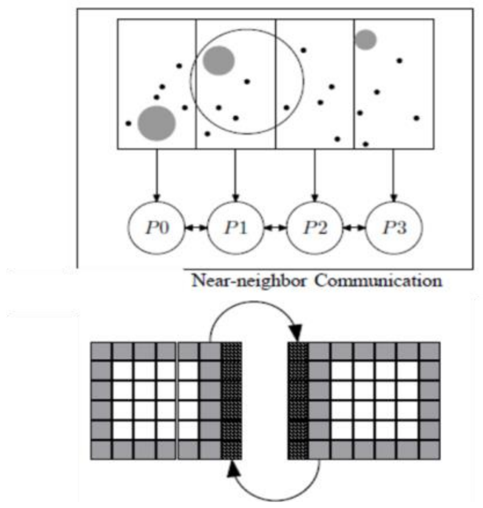


Fig. 19. MPI regions and interchange of information

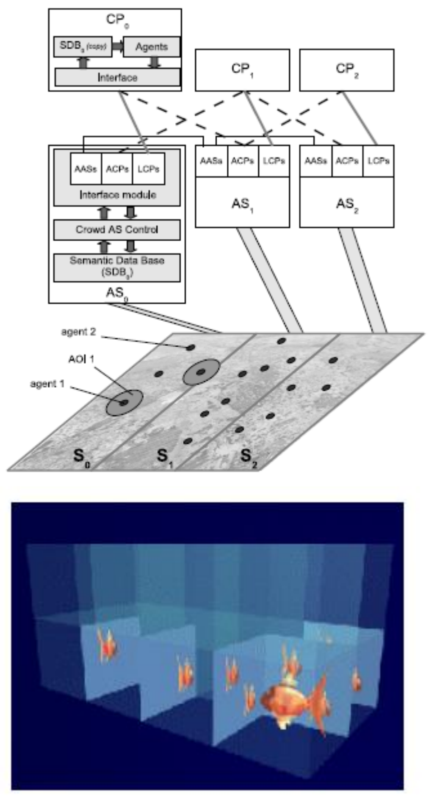


Fig. 20. MPI & Cameras, combining partial renders for parallel rendering

(computation of the position and orientation of character's end effector as a function of its joint angles) to generate a pose. Each pose is a set of local transformation matrices which specify rotations on each articulation, while an animation clip is a set of poses.

Using texture-based methods as described previously allows us to transfer animations straightforwardly between characters which share the same UV mapping (Fig. 17 bottom), thus material and computational resources are reduced.

6 Clothing

In a spirit similar to that of image space skinning, we can also add clothing. We can model cloths as a system of mass and springs and the user overlays the desired model on the texture parameterization, aligned with the major body features. After this, as shown in Fig. 18, a sparse geometry image is used, which is obtained by sampling the model's original mesh, then it is compared against the garment texture and interpolated to obtain a smooth geometry image, also using a sewing image. The system can reconstruct the complete model mesh by iterating physics simulation:

- Physical forces applied to all masses in the system,
- Integrate system to solve for positions,
- Satisfy spring constraints,
- Resolve collisions with character (ellipsoids can be used to make this efficient).

7 Conclusions, Ongoing Work

We have been doing GLSL/CUDA data parallelism one GPU/thread per agent on a single node. The idea is that one can apply similar methods on GPU clusters and HPC for larger scale problems. Such systems scale up almost linearly by using multiple nodes.

One could use MPI and subdivide the world into 2D or 3D areas, exchange data in the borders between neighbors, as it can be seen in Fig.19.

In [25] a better architecture is proposed: subdivide into (2D/3D) areas, with interchange of buffers at borders, areas manage their own agents, while manager distributes areas to each worker to maintain load balance. Other approaches add separate managers for cameras [26], as can be seen in Fig. 20 top.

For parallel rendering, one would need to run OpenGL on each node and do partial renderings, to be combined for achieving final visualization results. There are several ways to do this and each of them has pros and cons. An illustration can be seen in Fig. 20 bottom.

However, in view of the latest developments such as the new family of mobile multicore chipsets and GPU-based cloud gaming platforms, the pieces are almost there for this kind of architecture to work.

Acknowledgements

This work has been partially funded by CONACyT-BSC postdoctoral fellowship; SNI-54067; “Agentes virtuales y Robóticos en Ambientes de Realidad Dual”, Tecnológico de Monterrey research initiative. We would also like to thank Nvidia for hardware donations used in some of these algorithms.

References

1. **Reynolds, C.W. (1987).** Flocks, herds and schools: A distributed behavioral model. *14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH’87),* Anaheim, California, 25–34.
2. **Helbing, D. & Molnar, P. (1995).** Social force model for pedestrian dynamics. *Physical review E,* 51 (5), 4282–4286.
3. **Helbing, D., Farkas, I., & Vicsek, T. (2000).** Simulating dynamical features of escape panic. *Nature,* 407(6803), 487–490.
4. **Van Den Berg, J., Lin, M., & Manocha, D. (2008).** Reciprocal velocity obstacles for real-time multi-agent navigation. *2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA, 1928–1935.
5. **Guy, S.J., Chhugani, J., Curtis, S., Dubey, P., Lin, M., & Manocha, D. (2010).** PLEdestrians: a least-effort approach to crowd simulation. *2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA’10),* Madrid, Spain, 119–128.
6. **Jund, T., Kraemer, P., & Cazier, D. (2012).** A unified structure for crowd simulation. *Computer Animation and Virtual Worlds,* 23(3-4), 311–320.
7. **Tecchia, F., Loscos, C., Conroy, R., & Chrysanthou, Y. (2001).** Agent Behaviour Simulator (ABS): A Platform for Urban Behaviour Development. *First International Game Technology Conference*. Hong Kong, China.
8. **Treuille, A., Cooper, S., & Popović, Z. (2006).** Continuum crowds. *33rd International Conference and Exhibition on Computer Graphics and Interactive Techniques (ACM SIGGRAPH '06),* Boston, USA, 1160–1168.
9. **Millan, E., Hernandez, B., & Rudomin, I. (2007).** Large crowds of autonomous animated characters using fragment shaders and level of detail. *ShaderX5: Advanced Rendering Techniques (*501–510), Boston, MA: Charles River Media.
10. **Moussaïd, M., Helbing, D., & Theraulaz, G. (2011).** How simple rules determine pedestrian behavior and crowd disasters. *Proceedings of the National Academy of Sciences of the United States of America,* 108(17), 6884–6888.
11. **Ondřej, J., Pettré, J., Olivier, A.H., & Donikian, S. (2010).** A synthetic-vision based steering approach for crowd simulation. *Special Interest Group on Computer Graphics and Interactive Techniques Conference (SIGGRAPH 2010),* Los Angeles, CA., USA, Article No. 123.
12. **Zhou, S., Chen, D., Cai, W., Luo, L., Low, M.Y.H., Tian, F., Tay, V.S.H., Ong, D.W.S., & Hamilton, B.D. (2010).** Crowd modeling and simulation technologies. *ACM Transactions on Modeling and Computer Simulation*, 20(4), Article 20.
13. **Cormen, T.H., Leiserson, C.E., Rivest, R.L., &Stein, C. (2001).** *Introduction to Algorithms* (2nd ed.). Cambridge, Mass.: MIT Press.
14. **Blue, V.J., Embrechts, M.J., & Adler, J.L. (1997).** Cellular automata modeling of pedestrian movements. *IEEE International Conference on Systems, Man, and Cybernetics*, 3, Orlando, FL, 2320–2323.
15. **Zhang, S., Li, M., Li, F., Liu, A., & Cai, D. (2011).** A simulation model of pedestrian flow based on geographical cellular automata. *19th International Conference on Geoinformatics*, Shanghai, China, 1–5.
16. **Zhiqiang, K., Chongchong, Y., Li, T., & Jingyan, W. (2011).** Simulation of evacuation based on multi-agent and cellular automaton. *2011*

International Conference on Mechatronic Science, Electric Engineering and Computer, Jilin, China, 550–553.

17. **Bian, C., Chen, D., & Wang, S. (2010).** Velocity field based modelling and simulation of crowd in confrontation operations. *16th International Conference on Parallel and Distributed Systems*, Shanghai, China, 646–651.
18. **Pettré, J., Grillon, H., & Thalmann, D. (2008).** Crowds of moving objects: Navigation planning and simulation. *35th International Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH 2008),* Los Angeles, CA, USA, Article No. 54.
19. **Ju, E., Choi, M.G., Park, M., Lee, J., Lee, K.H., & Takahashi, S. (2010).** Morphable crowds. *ACM Transactions on Graphics*, 29(6), Article 40.
20. **Banerjee, B., Abukmail, A., & Kraemer, L. (2008).** Advancing the layered approach to agent-based crowd simulation. *22nd Workshop on Principles of Advanced and Distributed Simulation*, Roma, Italy, 185–192.
21. **Sukop, M.C. & Thorne Jr., D.T. (2006).** *Lattice Boltzmann Modeling: An Introduction for Geoscientists and Engineers*. Berlin; New York: Springer.
22. **Hernández, B. & Rudomin, I. (2011).** A rendering Pipeline for Crowds. *GPU Pro 2: Advanced Rendering Techniques* (369–384). Natick, Mass.: AK Peters.
23. **Rusinkiewicz, S. & Levoy, M. (2000).** QSplat: a multiresolution point rendering system for large meshes. 27th *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00),* New Orleans, Louisiana, USA, 343–352.
24. **Blanz, V. & Vetter, T. (1999).** A morphable model for the synthesis of 3D faces. *26th Annual Conference on Computer Graphics and Interactive Techniques*, Los Angeles, CA, USA, 187–194.
25. **Vigueras, G., Lozano, M., Perez, C., & Orduña, J.M. (2008).** A Scalable Architecture for Crowd Simulation: Implementing a Parallel Action Serve, *37th International Conference on Parallel Processing*, Portland, OR, 430–437.
26. **Vigueras, G., Lozano, M., & Orduña, J.M. (2011).** Workload balancing in distributed crowd simulations: the partitioning method*. The Journal of Supercomputing*, 58(2), 261–269.
27. **Mu, Y., Yan, S., Liu, Y., & Huang, T. (2008).** Discriminative local binary patterns for human detection in personal album*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2008),* Anchorage, AK, 1–8.

Isaac Rudomin recieved his Ph.D. from University of Pennsylvania. He is currently a senior researcher at the Barcelona Supercomputing Center, Spain. He is a member of the National System of Researchers of Mexico, Level I. He was a full-time professor at the Monterrey Institute of Technology, State of Mexico campus (Tecnológico de Monterrey, campus Estado de México) from 1991 to 2012. His research interests are human and crowd simulation, animation and visualization, human-computer Interaction and high performance computing.

Benjamín Hernández recieved his Ph.D. from the Monterrey Institute of Technology, State of Mexico campus (Tecnológico de Monterrey, campus Estado de México). He currently holds a post-doctoral research fellowship at the Barcelona Supercomputing Center, Spain, on leave from his position as a full-time professor at the Monterrey Institute of Technology, Mexico City campus. He is a member of National System of Researchers of Mexico, Candidate Level. His research interests are at the intersection of real-time simulation, animation and visualization of crowds, human computer interaction and parallel computing on GPUs. He has also advised postgraduate theses in these fields.

Oriam De Gyves is a Ph.D. student specializing in Computer Science and particularly in Computer Graphics, at the Monterrey Institute of Technology, State of Mexico campus (Tecnológico de Monterrey, campus Estado de México). He studies and does research of behaviors for crowd simulation using General Purpose Computation in Graphics Processing Units. His research interests include simulation, behavior and visualization of real-time crowds, as well as parallel computing on GPUs.

Leonel Toledo is a Ph.D. student at the Monterrey Institute of Technology, State of Mexico campus (Tecnológico de Monterrey, campus Estado de México). He made a research interchange visit working on crowd simulation at the Barcelona Supercomputing Center, Spain. He has been a half-time Professor at the Monterrey Institute of Technology, State of Mexico campus, since 2011 and his research interests include crowd simulation, animation, visualization, and rendering.

Ivan Rivalcoba is a doctoral student at the Monterrey Institute of Technology, State of Mexico campus (Tecnológico de Monterrey, campus Estado de México). His thesis focuses on computer vision algorithms for the study of crowd behavior. Besides, he works as an IT Professor at Gustavo A. Madero Institute of Technology. He made a research interchange visit working in computer vision at the Barcelona Supercomputing Center, Spain. His research interests also include computer vision and feature descriptors for human detection.

Sergio Ruiz is a software engineer at the Monterrey Institute of Technology, Mexico City campus (Tecnológico de Monterrey, campus Ciudad de México). Currently, he is a Ph.D. student at the same institution; his research area is path planning for simulated crowds proposing a thesis entitled “A hybrid method for macro and micro simulation of crowd behavior”. He is currently on a research interchange visit working on crowd simulation at the Barcelona Supercomputing Center, Spain.

Article received on 01/02/2013; accepted on 10/08/2013.