Eurographics Italian Chapter Conference (2006)
G. Gallo and S. Battiato and F. Stanco (Editors)

# An Architecture for Distributed Behavioral Models with GPUs

R. De Chiara, U. Erra, V. Scarano†

ISISLab - Dipartimento di Informatica ed Applicazioni "R.M. Capocelli" - Università di Salerno

**Abstract**

*We describe an architecture for massive simulation of a distributed behavioral model using graphics hardware. By leveraging on the recent programmable capabilities of GPUs we implemented the model capable of managing a large aggregate motion of birds in a virtual environment that can avoid both static and dynamic obstacles. We demonstrate the effectiveness of our GPU implementation by comparing the results to a CPU implementation and, emphasize the modularity of the proposed architecture that favors reusability in several contexts.*

Categories and Subject Descriptors (according to ACM CCS): I.6.8 [Simulation and modelling]: Animation

## 1. Introduction

One of the most beautiful and fascinating show that nature is capable to offer are the amazing evolutions of a flock of birds, a bank of fishes or a herd of animals. Although these groups consists of tens to hundreds of elements that stay close together at incredible speed, they never collide with each other or with an obstacle (e.g. trees or a wall) rather they create a perfectly synchronized aggregate motion.

A flock is composed by single entities which take decisions based only on their local perception of the group through their immediate neighbors. No centralized control is available but each member follows its own behavioral model composed of few, simple rules: keep distance from other birds, stay aligned, fly toward a goal, avoid obstacles and so on. Surprisingly, a global group behavior emerges from this limited set of simple rules.

Simulating the group behavior of a large number of moving members has received the attention of different researches in several fields such as computer animation [Rey87], virtual reality [UT02], artificial life [TTG94] or human related behavior [LMM03]. Today, the availability of computational power has begun to move attention from simulation of few elements to large groups in real time.

† {dechiara, ugoerr, vitsca}@dia.unisa.it

© The Eurographics Association 2006.

In this paper, we present a modular architecture for simulating behavioral models on a GPU. The architecture is designed so that each level can be easily replaced and adapted to different scenarios, as well as being used both for simulation and for real-time visualization. In Section 2 we, first, describe the behavioral models and some implementation techniques that are used on GPUs. Then, in Section 3, we describe the model of flock motion by Reynolds [Rey87] and, successively, in Section 4, use it as an example to show how our pseudo-algorithm is able to map behavioral models on the GPUs. Finally, in Section 5, we compare the simulation to a CPU-based solution as well as compare our architecture to other parallel solutions available in literature.

## 2. Behavioral models and GPUs

In 1983 William Reeves [Ree83] proposed for the first time a simple behavioral model for modeling water, fire, grass and atmospheric effects using particles system. In this model a particle can be defined as a point in 3d space with a set of associated attributes as for instance position, velocity, size, shape, lifetime, and so on. A simulation using this model does not take into account the interaction between particles but the motion is defined only by their internal state.

Since the publication of Reeves' paper, several extensions have been made to the initial particle system model. One of the most important was provided by Craig Reynolds in

1987 [Rey87]: he proposed a model for simulating decentralized behavioral as flocks of birds, bank of fishes, or a herd of land animals called autonomous characters. The main difference between the two models was that in Reynolds' each element is "intelligent" because the simulation takes into account not only the internal state but also external conditions as, for instance, the distance of a character from its neighbors. In fact, realistic animations of the entire group emerges from these simple local rules within the flock structure. In a certain sense it can be seen as a dynamic system like a cellular automata where the space is fixed.
Two relevant works has been done on passive particle system [KSW04, KLRS04] using programmable graphics hardware. In particular [KSW04] takes advantage of OpenGL memory objects for an efficient GPU realizations of a real-time animation and rendering of particle dynamics.

The model of Reynolds is the starting point for other decentralized behavioral where individuality is defined using simple local interactions between individuals. In recent works, see [BMdOB03], a physically based model of crowd is simulated using different individualities for agent and group behaviors in the particle systems. In [BCN97] the authors have used particle systems and dynamics respectively for modeling the motion of groups with significant physics. In [NT96] the authors have developed local rules for controlling collective behaviors.
In these works, a group is composed essentially of several *autonomous characters*. These characters can interact between them and between the environments through a set of steering behaviors which are defined independently of the character's means of locomotion [Rey99]. These simple behaviors are the building blocks that can be combined to create more complex patterns of behavior. Two are the important aspects of the computation that must be considered carefully: proximity queries and obstacles data structures. Proximity queries are necessary to know information about neighbors: potentially each characters can interact with each other. This problem, know as *n-body forces problem* has an asymptotic complexity of $O(n^2)$. Instead, obstacles data structure is a requirement necessary to implement obstacle avoidance behaviors.

Much of the work in literature about behaviors models addresses the problem of simulating correctly the motion of groups composed of few elements rather then visualizing or simulating interactive large groups (or crowds) in real time. Recently, graphics processor units have shown high performance computing at low cost. The reason behind this performance is the parallel architecture of the graphics processors which permits several data organized into *stream* to be processed in parallel using a same program called *kernel*. In particular, data parallelism is made possible ensuring that computation of one element of the stream is not affected by another element of the same stream. That is, data parallelism implies local computation during processing. Thereby, stream programming model naturally fits within the requirements of the decentralized behavioral models. We now show which resources of the programmable graphics hardware are suitable to tackle the problems and the techniques used in order to design a modular architecture for behaviors models.

### 2.1. GPU Resources

**Textures.** Implementing efficient parallel data structure on the GPU requires use of textures. In fact, textures are bi-dimensional arrays of 4-dimensional component of float values. For each character, state-preserving attributes like position, velocity, mass, size are stored compactly in place of pixels into 2D textures as shown in [DEST04, KLRS04, CRM04]. Textures are used not only for storing state-preserving information but also to store environment-related information as we will see below. During simulation, textures are used as rendering target to maintain output related to every single behavior. Since graphics hardware does not support read/write at the same time, we use a double buffering schema where state-preserving information appear as input and intermediate results computed by a single behavior appear as output.

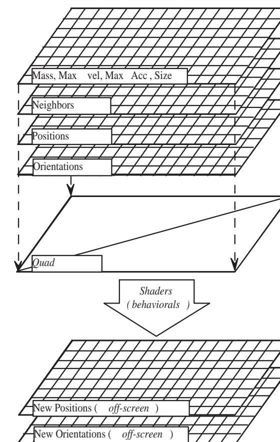


**Figure 1:**

The streaming model on GPU.

**Shaders.** The "brain" of autonomous characters are implemented using fragment shaders. Today, graphics hardware supports multiple pipeline which permits parallel execution of a fragment program. Then, multiple characters can be processed at the same time. Steering behaviors are implemented using fragment shaders, so that each shader takes a state-preserving information as input and has position and orientation related to current behavior as output. Combinations of steering behaviors can be done using a feature of stream programming: the *Data-Dependency Graph*. In this

© The Eurographics Association 2006.

way, more behaviors can be chained together allowing complex behaviors to be generated.

#### 2.1.1. Space and obstacles perception

**Proximity queries.** To support the local perception but also to manage a large set of characters we use a data structure that (1) allows to quickly obtain information about neighbors of every character and (2) has to be updated efficiently at every frame as the group moves.
The idea is to use a simple space partitioning data structure where characters are sorted in a regular cell grid and every cell keeps a list of characters that are available in it. For any given character in constant time we can calculate the cell it is located and, by exploring the adjacent cells, it is possible to gather information about characters around it.
Although sorting can be efficiently implemented on GPUs as shown in [KLRS04], by using the well-known parallel sorting algorithm “odd-even merge sort”, our choice was to maintain this part on CPU. In fact, the number of passes to sort the entire group can be too expensive for each frame as demonstrated in [KLRS04]; techniques such as distributing the load over several frames yields an approximate solution that is not suitable for our simulation. Then, during the simulation, updated information about new positions is sent back to the CPU. This strategy allows, also, to collect intermediate results (on the CPU) if one is interested. Furthermore, the increasing availability of PCI Express bus allows information to flow back from the video card as fast as it flows in. Since the AGP bus was designed for graphics, there was never a perceived need for rendered graphics to go anywhere other than out the back of video card. Today, PCI Express can be considered a further step toward tightly coupled CPU-GPU applications.
Then, the CPU sorts data into spatial data structure and, for each character, it finds the nearest neighbors. Up to four neighbors can be stored compactly into one texture for each character, but more than a texture can be used to store neighbors based on particular behavior model. Since this phase is known as a critical time-consuming phase (see [Rey00]) we present later a heuristic in order to avoid sorting under user-defined conditions.

**Obstacles avoidance.** In the simulation an important aspect is the interaction with objects in the environment. An expected behavior is that an autonomous characters avoids the obstacles it meets on its own path. A possible collision adds new information in the knowledge of every character that sees it and the model has to take it into account. A classical approach used in [EW96] is the *force fields*. In this method a discrete force field surrounds every object present in the environment, approaching to an obstacle the forward vector is summed with the vectors of force field and the character feels a growing opposing force on its path toward it. Our approach for the force field on the GPU takes into account the dot product between the force field vector $\vec{\mathbf{n}}$ and forward vector $\vec{\mathbf{v}}$ the steering vector $\vec{\mathbf{s}}$ can be guessed from these three cases:

$$\vec{\mathbf{s}} = \begin{cases} 0 & \text{if } (\vec{\mathbf{v}} \cdot \vec{\mathbf{n}}) = 0 \quad (1) \\ \vec{\mathbf{n}} & \text{if } (\vec{\mathbf{v}} \cdot \vec{\mathbf{n}}) > 0 \quad (2) \\ \vec{\mathbf{n}} + \vec{\mathbf{v}} & \text{if } (\vec{\mathbf{v}} \cdot \vec{\mathbf{n}}) < 0 \quad (3) \end{cases}$$

In (1) the character is going parallel to an obstacle. In (2) the character is going away from an obstacle. In (3) the character is going toward an obstacle.
A problem that appears by using the force field solution is what we called “lack of time”: it is the situation in which a character may be safe at time $i$ and at time $i+1$ it may be inside a wall because of the approximations due to the discrete time simulation. To resolve this problem we try to foresee future positions in order to verify if a character is on a collision route. In a sense, this is similar to the approach used by Reynolds in [Rey99] where each character is prolonged along the forward vector in order to anticipate possible collision routes. The force field is built for every object in the scene using its normal vectors (Figure 6). This field is built using a coarse geometry model because, in order to avoid the obstacles, it is important to have quick approximate information about the shape and no detail is needed. The force field is discretized along a regular grid and stored into a 3D texture. Intermediate values can be computed using high efficient hardware linear interpolation. Moreover, an additional null layer outside the 3D texture permits a linear interpolation from a position where the field has maximum influence to another position where has no influence. However, this solution appears as a technique to manage both static and dynamic obstacles.

## 3. A behavioral model for flock motion

In this section we will aim at simulating a flock of birds starting from Reynold’s work, but we emphasize here (and, later, describe with modere detail in Section 4) how our considerations can be applied to others scenarios too. This model simulates the complex aggregate motion of a flock of birds, a herd of land animals, or a bank of fishes. In [Rey87] Reynolds focused his study about a model that is apparently plausible without considerations about how to manage efficiently large aggregate motion of elements.

Every element in this group is called *boid* (the contraction of *birdoid*). Every boid has some limitations: it has a strictly local knowledge of the space it occupies. In fact, none of the creatures being part of a group has a full knowledge of the entire group. Hence, the decisions must be taken by every single element from local perceptions of the world as well as from information that is perceived from its neighbors. This distributed decisions mechanism is borrowed from nature and the flock takes its decisions in a totally distributed manner aiming in order to obtaining a synchronized movement. The keystone of the simulation of this model is the imitation of a base set of elementary behaviors which combined are usually deep enough to enable the flock to present

© The Eurographics Association 2006.

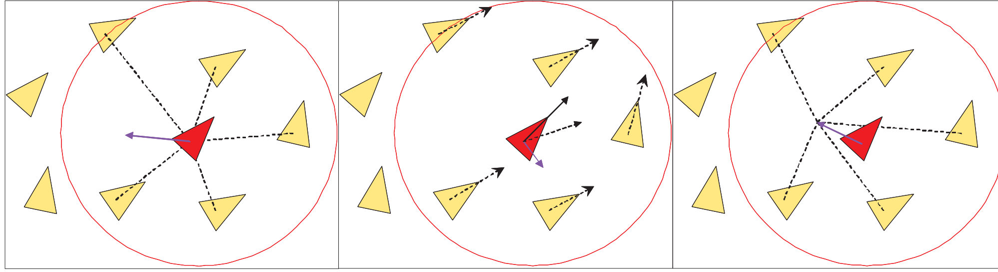

**Figure 2:** *Steering behaviors: (left-to-right) separation, alignment, cohesion.*

the complex aggregate motion that we can see in nature. This distributed behavioral mechanism jointly with a reasonable simulation of the physics of flight produces a very natural behavior.

### 3.1. Boid definition.

Every boid in the system is defined as follows: a set of state-preserving data used for the simulation of the flight as mass, maximum speed, maximum acceleration, global position, the current speed and a view reference system used to represent the point of view of the boid. Part of this information is constant and defined at the "birth" of the boid while another part is updated at every frame of the simulation.

The synchronized aggregated motion of the flock is achieved by fixing one or more spatial goals which the boids have to reach; these goals are the result of the sum of every boid's steering behaviors. The sum performed is weighted in order to give a characterization to every boid, a sort of *personality*. Every decision is taken considering a certain number of neighbors. This number is fixed to four in order to use only one texture but our simulation showed that this value was quite good. We have implemented three basic types of steering behavior presented by Reynolds in [Rey99] and called *flocking behavior* (an intuitive representation of them is shown in Figure 2):

- the *separation* behavior tends to keep distance from other neighbors. This behavior is necessary to prevent boids collision. A repulsive force is calculated as the difference vector between current boid position and every neighbors while the steering force is calculated as the average vectors between all the repulsive forces.
- the *alignment* behavior tends to align the boid with other neighbors computing the steering force as difference between the average of the forward vectors of the neighbors and the forward vector of the boid itself.
- the *cohesion* behavior tends to move the boid toward the center of his local neighborhood. This behavior is useful in order to give to the flock an aggregated aspect. The steering force is obtained computing the average position of neighbors.

Another behavior that has been implemented in the simulation is the *leader following* behavior. This behavior constrains every boid to follow a fixed leader inside the flock, this leader can follow a predefined path or a random path. The leader can change during time, it can be a random boid, the farthest boid, the mass center of a group of boids or can be a fixed goal to reach. Finally, since the flock can not move inside a scene infinitely large, we also implemented a *containment* behavior, it constrains every boid to remain inside a bounding box that surrounds the entire scene and hence constrains the flock to remain in the scene.

## 4. Mapping behavioral models on the GPU

We will show how to implement a general model of simulation for autonomous characters on GPU and how we specialized this general structure to simulate the Reynolds model.
The simulation process takes as input four textures: 1) a constant texture $T_c$ to store scalar information as mass, maximum velocity and maximum acceleration, 2) a texture $T_o$ for orientation (three scalar values), 3) a texture $T_p$ for position (three scalar values) and current velocity (one scalar value), 4) a texture $T_n$ to store the nearest autonomous characters which will be the "sight" of every single character.
A single simulation step is showed in Algorithm 1, every execution updates the *entire flock*, being it fully defined by a set of textures.

The for-loop on line 1 is used to apply the various behaviors that every autonomous character is intended to run. The for-loop on line 1 updates an accumulation texture $T_{s_a}$, this texture will keep track of the influences of every obstacle force field summed up. The behaviors and the obstacles create a set of steering textures that are summed up on line 1: the personality blend use a weighted average to compute the texture $T_s$ which contains the total force applied on every autonomous character. From the total force an acceleration texture, $T_a$, is computed to decide how the position, the velocity and the orientation of every autonomous character is altered. Once again all these modifications are computed on the entire flock in a single step, on lines 1-1.
In the special case where the autonomous characters are

© The Eurographics Association 2006.

**Algorithm 1** The simulation algorithm

**Require:** Given an environment with $m$ obstacles and $b$ different behaviors to apply to $n$ autonomous characters
**Ensure:** Performs one simulation step for all the $n$ autonomous characters
1: Prepare the input texture $T_c, T_o, T_p, T_n$
2: **for** each behavior $i$ **do**
3: $[T_{s_i}] \leftarrow \text{BEHAVIOR}_i[T_p, T_o]$
4: **end for**
5: $[T_{s_a}] \leftarrow 0$
6: **for** each obstacle $i$ with texture field $T_{fi}$ **do**
7: $[T_{s_a}] \leftarrow \text{OBSTACLEAVOIDANCE}[T_p, T_o, T_{s_a}, T_{fi}]$
8: **end for**
9: $[T_s] \leftarrow \text{PERSONALITYBLEND}[T_{s_a}, \langle T_{s_1}, \ldots, T_{s_b} \rangle]$
10: $[T_a] \leftarrow \text{ACCELERATION}[T_c, T_s]$
11: $[T_P] \leftarrow \text{POSITIONVELOCITY}[T_c, T_a]$
12: $[T_O] \leftarrow \text{ORIENTATION}[T_c, T_a]$

boids obeying to Reynolds' model the algorithm will manage 5 different behaviors, leader following, containment, cohesion, alignment and separation, as showed in previous section.
Our architecture is intended to be general with respect to the complexity of the model to simulate, that means it can be used to simulate complex behavioral model like in [CRM04] or even simpler models like [KLRS04] even if, for certain cases, more efficient, ad-hoc, implementations can be obtained. We intend to emphasize that all these models share a common internal structure, and the main differences between them are the complexity of behaviors that can be easily assembled in our architecture.

### 4.1. Neighbor searching heuristics

The neighbor searching is performed for every character and provides it a partial knowledge of relative positions of the other characters: for every character it calculates a prefixed length list of the nearest characters in the flock. In our architecture this phase is implemented on CPU and we are aware that it can be a heavy computational phase.
In order to avoid the calculation of this list at every frame, we developed an heuristic that limits the calculation of neighbors list just when these lists are not more reliable. The intuition is: when the flock moves uniformly, its lists of neighbors remain the same from frame to frame, on the contrary when the movements are rapid and unexpected the neighbors, for every character, rapidly change needing recalculating the lists. At this point it is useful to consider that the calculation of the neighbors list is performed in a grid-based flavor: every character considers as neighbors just the characters belonging to the cells around it. Our heuristic is the following: at the beginning of every step of the simulation every character knows which cell of the grid it belongs to, at the end of the step, after performing all the calculation regarding its behavior, the character knows in which cell of the grid it will belong on the next step (both this information can be obtained at the cost of a floating point division). Every character can express the relative variation of cell position with a triple of values, one for every spatial dimension, taken from the set $[-1, 0, +1]$. At the beginning of every step a 3-dimensional matrix of 27 ($3 \times 3 \times 3$) values, called *scattering matrix*, is cleared and it is used to keep track of *the amount of characters that, at the end of each frame, have performed a certain change of cell*, the intent of the matrix is to measure the *scattering* of the flock. What we expect is that, as long as the flock moves freely in a wide empty space, it will keep a quite stable shape also keeping constant the neighbors lists, and this information is reflected in the scattering matrix through a large value in one cell and almost 0 in other cells. On the other hand, whenever the flock reaches the bounds of the space, or intercepts an obstacle to be avoided, the matrix will present a certain number of cells containing small values, meaning that boids are changing direction with an high variability. A large value in a cell of matrix means that a lot of boids are changing cell (in the spatial subdivision matrix) in the same direction. A sparse scattering matrix means that boids are changing cell (once again in spatial subdivision matrix) in various directions and this means that the neighbors lists have to be recalculated.
Tests showed us that, even in presence of a small variations of direction, the use of the scattering matrix heuristic allowed us to avoid the neighbor searching in almost 20% of the frames.

## 5. Results and conclusions

We have experimented on a PC with 3.2 GHz Intel Pentium IV CPU and a graphics board NVIDIA GeForce 6800GT with NV40 GPU, core speed of 450Mhz and memory speed of 700Mhz. The performance of CPU-based algorithm was measured using Microsoft Visual C++ .NET 2003 compiler. The test scene used to render the flock consists of a statically tessellated terrain, and six avoidable objects, each one with its vector field: four columns, a crossbeam and a moving sphere (Figure 5). The scene is composed of about 10000 polygons and each boid has 72 polygon.
Figure 3 shows the computational times on the GPU

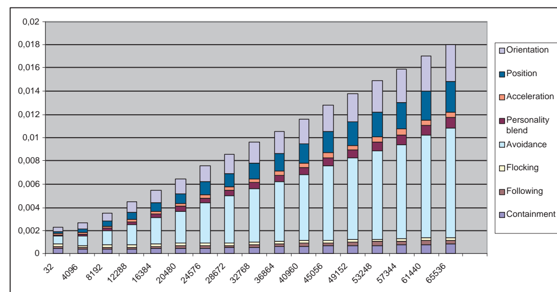


**Figure 3:** *Computational times on the GPU broken down into different passes.*

© The Eurographics Association 2006.

for each pass. It is clearly shown how behaviors like containment and following are almost independent from the number of birds present in the flock excepts avoidance. This is a time-consuming phase due to numerous read/write access into textures.
On the up side of Figure 4 we show the average computation time for a single frame without rendering the scene, i.e., the simulation time of the scene. On the down side of the same figure, we show the performances if one is interested also in the real-time rendering of the scene that is measured in terms of (average) frame-per-second. In both cases, we considered the average values on 1000 frames.

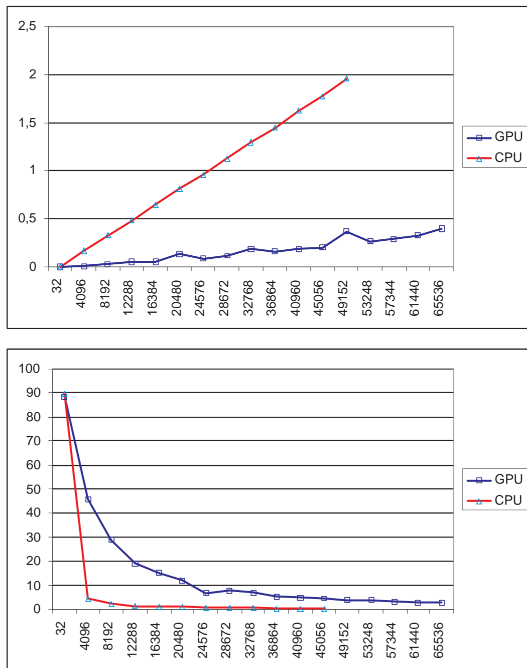


**Figure 4:** *Performances of the two different implementation(CPU Δ vs GPU ◇) with the number of boids on the x-axis. (up) Simulation time on the y-axis is expressed in seconds. (down) Visualization performances on y-axis is shown in frame-per-seconds.*

A first interesting comparison of our results can be conducted with results reported by Reynolds in [Rey00] where a (constant) performance of 60 fps was obtained with 600 boids. Our system animates more than 6000 boids with obstacle avoidance at 60 fps, but interactive frame rates, usually about 20 fps, can yet be obtained with more than 13000 boids.
In [ZZ04], the authors implemented Reynold's model (with # of boids up to 512) on a cluster of up to 16 PCs with high-bandwidth network (Myrinet). With the highest number of boids they analyze, our results are comparable with theirs (with 16 processors) since we require around 10 s to process 1000 frames for 512 boids (with visualization) and 4 s without visualization. It must be said that their scene contains 24 static obstacles but ours contains one moving obstacle beside the 5 static ones. Of course, we are obtaining the same results on an ordinary PC and our approach seems to scale much better, since no increase in communication overhead must be payed when the number of boids increases.
In this paper, we propose an architecture that can be used in several different contexts. The performances obtained witness the efficiency of the approach and we believe that our architecture represents an ideal framework to implement more elaborated behavioral models.

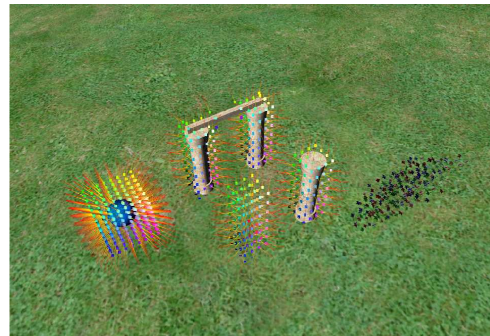

**Figure 6:** *The vector field.*

## References

[BCN97] Bouvier E., Cohen E., Najman L.: From crowd simulation to airbag deployment: Particle systems, a new paradigm of simulation. *Journal of Electronic Imaging 6*, 1 (January 1997), 94–107. Special issue on Random Model in Imaging.

[BMdOB03] Braun A., Musse S. R., de Oliveira L. P. L., Bodmann B. E. J.: Modeling individual behaviors in crowd simulation. In *CASA* (2003), pp. 143–148.

[CRM04] Courty N., Raupp Musse S.: Fastcrowd: Real-time simulation and interaction with large crowds based on graphics hardware. In *Eurographics/ACM Siggraph Symp. on Computer Animation (Poster session), SCA 04* (Grenoble, France, August 2004).

[DEST04] De Chiara R., Erra U., Scarano V., Tatafiore M.: Massive simulation using gpu of a distributed behavioral model of a flock with obstacle avoidance. In *Proc. of Vision, Modeling and Visualization, Nov. 16-18, 2004, Stanford (California, USA)* (2004), pp. 233–240.

[EW96] Egbert P. K., Winkler S. H.: Collision-free object movement using vector fields. *IEEE Comput. Graph. Appl. 16*, 4 (1996), 18–24.

© The Eurographics Association 2006.

**Figure 5:** *The tests scene: a statically tessellated terrain, four columns, a crossbeam and a moving sphere.*


[KLRS04] KOLB A., LATTA L., REZK-SALAMA C.: Hardware-based simulation and collision detection for large particle systems. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (New York, NY, USA, 2004), ACM Press, pp. 123–131.

[KSW04] KIPFER P., SEGAL M., WESTERMANN R.: Uberflow: a gpu-based particle engine. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (New York, NY, USA, 2004), ACM Press, pp. 115–122.

[LMM03] LOSCOS C., MARCHAL D., MEYER A.: Intuitive crowd behaviour in dense urban environments using local laws. In *TPCG* (2003), pp. 122–129.

[NT96] NOSER H., THALMANN D.: The animation of autonomous actors based on production rules. In *CA '96: Proceedings of the Computer Animation* (Washington, DC, USA, 1996), IEEE Computer Society, p. 47.

[Ree83] REEVES W. T.: Particle systemsŮa technique for modeling a class of fuzzy objects. *ACM Trans. Graph. 2*, 2 (1983), 91–108.

[Rey87] REYNOLDS C. W.: Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM Press, pp. 25–34.

[Rey99] REYNOLDS C. W.: Steering behaviours for autonomous characters. In *Game Developers Conference* (1999).

[Rey00] REYNOLDS C. W.: Interaction with groups of autonomous characters. In *Game Developers Conference 2000* (2000), CMP Game Media Group (formerly: Miller Freeman Game Group) San Francisco C., (Ed.), pp. 449–460.

[TTG94] TERZOPOULOS D., TU X., GRZESZCZUK R.: Artificial fishes: autonomous locomotion, perception, behavior, and learning in a simulated physical world. *Artif. Life 1*, 4 (1994), 327–351.

[UT02] ULICNY B., THALMANN D.: Towards interactive real-time crowd behavior simulation. 767–775.

[ZZ04] ZHOU B., ZHOU S.: Parallel simulation of group behaviors. In *Proceedings of the 2004 Winter Simulation Conference* (2004).


© The Eurographics Association 2006.