J. Parallel Distrib. Comput. 75 (2015) 20–28

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

# Neighborhood grid: A novel data structure for fluids animation with GPU computing

Mark Joselli [a,*], José Ricardo da S. Junior [b], Esteban W. Clua [b], Anselmo Montenegro [b], Marcos Lage [b], Paulo Pagliosa [c]

[a] PUC-PR, Brazil
[b] Media Lab – UFF, Brazil
[c] FACOM—Federal University of Mato Grosso do Sul, Brazil

## HIGHLIGHTS

- We present a new data structure for the neighborhood gathering on fluid simulation, called neighborhood grid.
- The neighborhood grid has an expressive speedup against the uniform grid on GPUs.
- The neighborhood grid uses less memory when compared with the uniform grid.

## ARTICLE INFO

*Article history:*
Received 8 January 2014
Received in revised form
16 July 2014
Accepted 20 October 2014
Available online 24 October 2014

*Keywords:*
Fluid animation
Real-time simulation
GPU computing
GPGPU
Data structure
Fluid simulation

## ABSTRACT

This paper introduces a novel and efficient data structure, called neighborhood grid, capable of supporting large number of particle based elements on GPUs (graphics processing units), and is used for optimizing fluid animation with the use of GPU computing. The presented fluid simulation approach is based on SPH (smoothed particle hydrodynamics) and uses a unique algorithm for the neighborhood gathering. The brute force approach to neighborhood gathering of $n$ particles has complexity $O(n^2)$, since it involves proximity queries of all pairs of fluid particles in order to compute the relevant mutual interactions. Usually, the algorithm is optimized by using spatial data structures which subdivide the environment in cells and then classify the particles among the cells based on their position, which is not efficient when a large number of particles are grouped in the same cell. Instead of using such approach, this work presents a novel and efficient data structure that maintains the particles into another form of proximity data structure, called neighborhood grid. In this structure, each cell contains only one particle and does not directly represent a discrete spatial subdivision. The neighborhood grid does process an approximate spatial neighborhood of the particles, yielding promising results for real time fluid animation, with results that goes up to 9 times speedup, when compared to traditional GPU approaches, and up to 100 times when compared against CPU implementations.



## 1. Introduction

Realism in real-time graphical simulations includes the search for real behaviors and physics. Due to the graphics technology improvements, simulation of natural phenomena, such as water flows or smoke, has become possible to be performed in real time and interactive environments, like scientific applications and digital games. In fact, this kind of simulation is now quite popular in computer games, digital movies and animations. It is important to state that in real-time simulations, there is a trade-off between realism and interactivity, and most of the times the realistic behavior is put aside in order to have interactive frame rates.

Many non-graphics algorithms traditionally executed on the CPU are often suitable for parallel execution, which makes them appropriate to be implemented on GPUs (graphics processing units). Fluid animation that explores this programming model on the GPU is a promising line of research, since it can drastically speedup the computation cost of the fluids' behavior [4]. This work presents a new data structure for speeding up the neighborhood

* Corresponding author.
*E-mail addresses:* mark.joselli@pucpr.br, mjoselli@ic.uff.br (M. Joselli), jricardo@ic.uff.br, josericardo.jr@gmail.com (J.R.d.S. Junior), esteban@ic.uff.br (E.W. Clua), anselmo@ic.uff.br (A. Montenegro), mlage@ic.uff.br (M. Lage), pagliosa@facom.ufms.br (P. Pagliosa).

http://dx.doi.org/10.1016/j.jpdc.2014.10.009
0743-7315/

gathering which can be applied in fluid simulation. The concepts of the proposed architecture could also be applied to other multi-core processors like multi-core CPUs, the Playstation 4, Xbox One, and clusters.

GPUs are a collection of SIMD (single instruction, multiple data) processors designed to run a streamed graphics pipeline. It is a computational model where the processing of each pixel is independent of the others and usually requires localized memory reads (texture fetching). There are rules of thumb to create efficient streamed applications, where the most important one is to organize the data streams in order to maximize memory read performance based on locality. These rules tend to result in a more efficient usage of available cache memory and read ahead mechanisms of these devices. The proposed simulation system makes use of such strategies.

The SPH (smoothed particle hydrodynamics) method simulates the fluid as a set of discrete elements, referred as particles. In order to process the interaction between the particles using the Lagrangian approach, a neighborhood-gathering algorithm is needed in order to process the forces among the particles. The naive approach of such algorithm has complexity of $O(n^2)$, since it has to process each particle against all the other $n - 1$ particles. Most of the research on parallel and distributed SPH fluid simulation tries to avoid the high complexity of proximity queries by applying some form of spatial subdivision to the environment and by classifying particles among the cells based on their position. To accelerate data fetching in a parallel hardware (such as GPUs) the particles listed must be sorted in a way that all particles on the same cells are grouped together. This approach helps decrease the number of proximity queries, but it is very sensible to the maximum number of particles that can fit in a single cell. In this work, instead of using a similar approach, it is proposed a novel data structure which maintains particles into another kind of proximity based data structure. In this data structure, called neighborhood grid, each cell stores only one particle and does not directly represent a discrete spatial subdivision. The neighborhood grid is an approximated representation of the system of neighbors of the environment that maps the $N$-dimensional environment to a discrete map (lattice) with $N$ dimensions. Hence, particles that are close in a neighborhood sense appear close to each other in the map. Another approach is to think of it as a multi-dimensional compression of the simulated environment that still stores the original position information of all particles.

**Contributions**: This work presents a novel parallel data structure for graphical real-time fluid animation based on the SPH method. This solution uses a novel data structure, called neighborhood grid, in order to gather the neighborhoods of the particles in an optimized manner. By doing so, this proposal fills the lack of GPU bounded architectures that usually process all elements in the scene at each frame, thus being an efficient solution for other parallel and distributed solutions of fluid simulations.

**Paper summary**: This work is organized as follows. After referring to the SPH method in Section 2 and the related works of fluid simulation in Section 3, the neighborhood grid data structure is introduced in Section 4. In Section 5 the architecture environment on GPUs and the acceleration data structures employed in the simulations are described. In Section 6 some results are shown and, in Section 7, the conclusions are presented.

## 2. SPH model

In order to perform fluid simulations, the following equations need to be solved:

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}\right) = -\nabla p + \rho\mathbf{g} + \mu\nabla^2\mathbf{v}, \tag{1}$$

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = 0, \tag{2}$$

which are known as Navier–Stokes equations for modeling the flow of incompressible Newtonian fluids. In these equations, $\rho$ represents the density of the fluid, $\mathbf{v}$ is the velocity field, $p$ is the pressure field, $\mu$ represents the viscosity of the fluid and $\mathbf{g}$ corresponds to the resultant of external forces.

Eq. (1) is known as the equation of motion and states that the changes in the linear momentum must be equal to all of the forces that act in the system. Eq. (2) is known as the *mass conservation equation* and states that in the absence of sinks and drains the mass in the system must be constant.

In this paper, the Navier–Stokes equations are solved using the SPH meshless Lagrangian method, which was first introduced by [22,11] to perform simulations of astrophysical problems, and later extended for high weakly compressible flows [24]. Although there are some improvements over this method [31,10], the original one is employed in this paper.

In SPH, a compact support, radial and symmetrical smoothing kernel function is used to evaluate, for any point in the space, the field quantities which are only defined for a discrete set of particles [23]. The evaluation of a continuous scalar field $A(\mathbf{x})$ is achieved by calculating a weighted summation of the contributions of all the particles $i \in 1\ldots n$, with position $\mathbf{x}_i$, mass $m_i$ and attributes $A_i$ using: $A(\mathbf{x}) = \sum_{i=1}^{n} m_i \frac{A_i}{\rho_i} W(\mathbf{r}, h)$, where $\rho_i$ is the density of particle $i$, $\mathbf{r} = \mathbf{x} - \mathbf{x}_i$, $h$ represents the smoothing length and $W(\mathbf{r}, h)$ is the smoothing kernel.

The pressure field is computed using the state equation $p_i = k_p(\rho_i - \rho_0)$ [5], where $k_p$ is the stiffness constant of the fluid and $\rho_0$ is its rest density. In [25], the authors take $k_p = c^2$, where $c$ is the sound velocity. More details about the SPH method can be found in [26].

## 3. Related work

Physical simulations using Eulerian grid-based approach can be seen in Foster and Metaxas [9], who were the first to solve the full 3D Navier–Stokes equations in order to re-create visual properties of fluid dynamics. Stam [35] simulated gases employing a semi-Lagrangian integration scheme that achieves stability using artificial viscosity and rotational damping. Foster and Fedkiw [8] extended this technique to liquids using a level-set method. Latter, Enright et al. [6] added free surface tracking to the Fedkiw's approach.

The first graphical work using the meshless method called SPH was proposed by Desbrun and Gascuel [5] with the aim at simulating deformable objects. In [36], the method was extended to lava simulation through the coupling between viscosity and temperature. Müller et al. [26] used the SPH method to solve the Navier–Stokes equations in real time fluid simulations. In [27], the authors extended their work by allowing fluid interaction with rigid bodies to simulate virtual surgery using a Gaussian quadrature method to distribute ghost particles on rigid bodies surfaces, which are responsible for generating repulsive forces.

In this work, a new data structure is presented for fluid animation based on SPH, using GPU computing. There are some works that use GPU computing for doing fluid animation. Kipfer and Westermann [19] employed SPH to simulate fluid flows over deformable terrains on GPU using shader programming paradigm. Kurose and Takahashi [21] simulated fluids and rigid bodies with two-way interaction between them using SPH on GPU. For interaction, the authors discretized rigid body's polygons into a set of particles and solve a linear complementary problem (LCP) to calculate the collision forces among them. Also, Green [13] presents the implementation of the SPH on a GPU using CUDA.

When computing fluid animation based on SPH method, some form of neighborhood gathering becomes necessary for the calculation of the movement of the fluid particles, which has direct

influence in the performance of the simulation and can be the bottleneck of the SPH fluid simulation [12]. The most common approach in the literature for neighborhood gathering is the use of uniform grids. This method is fast to construct, but it suffers low cache-hit rate when applied in fluid animation, since the fluid particles fill the domain in a non-uniform way. Kurose and Takahashi [21] use this method to optimize the neighborhood gathering, but it also grows bigger in size as the fluid discretization grows.

One way of reducing this problem is using an index sort with the uniform grid. This index sort first sorts all the fluid particles according to their cell indices, and then stores them into an array. These indices of the sorted array are stored in each cell of the grid. With that, the cells just store one reference to the first particle with the corresponding cell index. This optimization is described in [30], and was implemented in the GPU by [13] for fluid simulation. The uniform grid with SPH on GPU can also be seen in many others works, like [14,39,2]. This method is also used in this work in order to compare the results with our structure.

In [15], the authors show two optimizations of a parallel CPU implementation for the uniform grid with index sort and the uniform grid, with great performance. They also present a comparison between this optimization and the other related methods. Although it is hard to compare to other's work, for the lack of the code and the tested architecture, the simulation with the neighborhood grid can achieve a 174 fps (frames per second) rate (which comprises the processing of the neighborhood gathering, behavior processing and render the fluid) of 262k particles, while the work [15] has a performance of 16.5 ups (updates per second, which consists of the processing of the neighborhood gathering and behavior processing without rendering and with a fixed time step) of 170k particles.

Goswami et al. [12] show an implementation of SPH with CUDA using a $z$-indexing optimization for the traditional uniform grid, implemented with spatial hash, which appears to be the fastest real-time SPH method in the literature. Based on the $z$-index hash map, the approach can achieve better performance since it reduces the memory overhead for organizing the data in the memory to avoid bank conflicts and optimize memory coalescence. Our proposal also includes memory coalescence and naturally organizes its memory to avoid bank conflicts. Goswami's work achieved a performance of 10 fps for the physical calculations (without the render) of 255k particles running on a Geforce GTX 280. Even though it is not fare to compare without having the same simulation parameters and hardware environment, our approach achieved 16, 8 fps for the same steps, but also including the rendering stage, with 262k on the same GPU model.

In [40] the authors combine in the same method the density and pressure force computation, which is updated with one frame delay. Even though it doubles the performance, it still has the uniform grid cost, which is higher than the neighborhood grid method. In addition, with that technique the simulation can become unstable [34].

There are also some other structures that can be used, like spatial hashing [37], Verlet list [38], kd-tree [1] and hierarchical subdivision [20], but as far as the authors of this work know, there is no performance implementation of fluid animation on GPU that uses them for neighborhood gathering.

By using some sort of spatial subdivision to classify the particles into a grid, the proximity query algorithm can be performed against a reduced number of pairs. For each particle, only those inside the same grid cell and at adjacent ones, depending on its position, were considered. This strategy leads to a sequential complexity that is closer to $O(n)$. This complexity, however, is highly dependent on the maximum density of each grid cell, which can be very high if the simulated environment is large and dense. We remark that the complexity of the neighborhood grid is not affected by the size of the environment or the distribution of the particles over it. An earlier version of this method has already been applied in a flocking simulation scenario with some promising results [29,17] (yielding considerable speedups over the traditional uniform grid).

## 4. Neighborhood grid

The computation of the particles' physical attributes in a meshless fluid simulation depends on observing the surrounding neighbors of each particle. It is known that the naive straightforward implementation of the neighborhood gathering algorithm has complexity $O(n^2)$, where $n$ is the number of particles, since it performs at least one proximity query for each particle pair in the simulation. Moreover, since particles are autonomous and can move each frame, the neighborhood maintenance is a very computationally intensive task.

Spatial subdivision techniques have been used to group and sort particles in order to accelerate the neighborhood search, especially for parallel and distributed solutions. Most of the implementations are based on spatial subdivisions techniques, such as a uniform grid over the considered environment. At each time step, all particles have their grid cell index calculated based on their locations. On GPU and other parallel-based solutions, sorting algorithms may be used to reorder the particles according to their cell index (this technique is usually called index sort), so it can benefit from the read-ahead and caching mechanisms of such hardware with parallel architecture. Using this approach, geometric neighbors are stored near each other in the data structure. With that, the data structure is very suitable for parallel processing, avoiding bank conflicts and taking advantage of data coalescent by maximizing memory read performance based on locality. However, static subdivisions schemes still have some limitations, for example, when they are used on simulations over large domains. If the size of each grid cell fits a large number of particle entities, the neighborhood search problem becomes limited by the $O(n^2)$ complexity factor in the worst case scenario.

Our approach uses a grid data structure, which the authors called neighborhood grid (NGrid), that stores information about all the particles of the simulation. In the neighborhood grid, each particle is mapped to an individual cell (1:1 mapping) according to its spatial location. Particles that are close in a geometric neighborhood sense are mapped to be close in the grid sense. In order to keep the neighborhood grid property, a sorting mechanism is necessary. The following subsections describe the neighborhood data gathering using the neighborhood grid method.

### 4.1. 3D proximity data structure: the neighborhood grid

The proposed data structure was mainly developed to be used with parallel devices like GPUs. All the information about the particles are stored in 3D arrays (the neighborhood grids), where each position holds the entire data for an individual particle. In this data structure, each cell fits only one particle.

The neighborhood grid structure is based on the Extended Moore Neighborhood gathering algorithm [32] using a 3D grid or a 2D matrix to hold the information of the particles (in this work a 3D grid is used in all tests). To reduce the cost of proximity queries, each particle only gathers the information about the particles surrounding its cell, based on a constant search radius. Fig. 1 illustrates the neighborhood grid with gathering radius 1.

This kind of spatial data structure with extremely regular information gathering enables a good prediction of the performance, since the number of proximity queries will always be constant over the simulation. This happens because instead of making the proximity queries over all particles inside a coarse grid bucket/cell

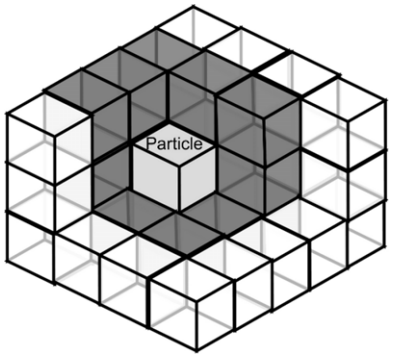



**Fig. 1.** Example of the neighborhood grid with radius equal to 1.

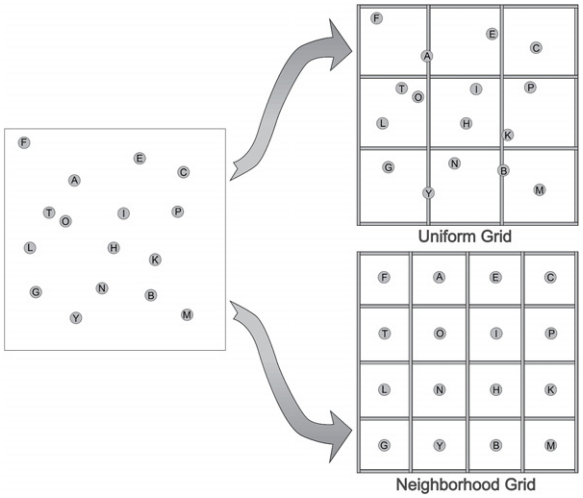



**Fig. 2.** Example of the organization of a set of particles.

(variable quantity), such as in traditional implementations, each particle would query only a fixed number of surrounding individual neighbors. However, this grid has to be sorted continually in such a way that those particles, which are neighbors in the geometric space, are stored in individual cells that are close in the neighborhood grid, so that each particle should gather information only about its closest neighbors. In order to better illustrate how the particles would be organized by the Neighborhood Grid and by the Uniform Grid, Fig. 2 illustrates a set of particles and how they would be organized in both structures.

The neighborhood grid starts, at the beginning of the simulation, with a full sort on the three dimensions, but during the simulation (and depending on the sorting step), some misalignment may occur over the data structure because the particles move during the simulation. This can make the gathering step to miss some of the neighbor particles. However, since the algorithm needs a sorting mechanism, this misalignment is very small—less than 1% of all particles are misaligned in the test case (when compared with the full sorted grid), and in the next step this misalignment is fixed by the sorting mechanism. This type of structure cannot guarantee that the processed set of particles' neighbors are really the closest set of neighbors in the Euclidean way, since the neighborhood grid can be misaligned or the chosen radius can be small. Based on tests, we observed that a particle usually have a mean of 17–40 particles. With this amount of particles and with a radius of 4, a particle could visit all its right neighbors, achieving a small error. But with the correct sorting algorithm and precise radius this kind of algorithm can yield visually believable simulations. We also think that this approximated neighborhood gathering is probably an interesting model for optimizing real-time visual fluid animation, since it resembles the movement of fluids.

### 4.2. Maintenance of the neighborhood grid: sorting stage

Since the particles move at each frame, the neighborhood grid becomes misaligned. In order to maintain the neighborhood grid in such a way that neighbors in the geometric space are stored in cells close to each other, it has to be sorted continually.

The position information of each particle is used to perform a lexicographical sort based on its three-dimensional coordinates. The goal is to store the particle with the smaller values for $Z$, $Y$ and $X$ in the closer-bottom-left cell of the grid, and the particle with highest values of $Z$, $Y$ and $X$ in the far-top-right cell, respectively. Using these three values to sort the grid, the furthest lines will be filled with the particles with the higher values of $Z$ while the top lines will be filled with the particles with higher values of $Y$ and the right columns will store those with higher values for $X$.

When performing a sorting over a one dimension array of floating point values, the goal is that given an array $\mathbf{A}$, the following rule must apply at the end: $\forall A[i] \in \mathbf{A},\ i > 0 \Rightarrow A[i-1] \leq A[i]$. Extending this rule to a grid $\mathbf{G}$, each cell has three floating point values $X$, $Y$ and $Z$, so the following rules are defined:

1. $\forall G[i][j][k] \in \mathbf{G},\ k > 0, G[i][j][k-1].Z \leq G[i][j][k].Z$;
2. $\forall G[i][j][k] \in \mathbf{G},\ k > 0, G[i][j][k-1].Z = G[i][j][k].Z \Rightarrow G[i][j][k].X \leq G[i][j][k-1].X$;
3. $\forall G[i][j][k] \in \mathbf{G},\ k > 0, G[i][j][k-1].Z = G[i][j][k].Z$ and $G[i][j][k].X = G[i][j][k-1].X \Rightarrow G[i][j][k].Y \leq G[i][j][k-1].Y$;
4. $\forall G[i][j][k] \in \mathbf{G},\ j > 0, G[i][j-1][k].Y \leq G[i][j][k].Y$;
5. $\forall G[i][j][k] \in \mathbf{G},\ j > 0, G[i][j-1][k].Y = G[i][j][k].Y \Rightarrow G[i][j][k].Z \leq G[i][j-1][k].Z$;
6. $\forall G[i][j][k] \in \mathbf{G},\ j > 0, G[i][j-1][k].Y = G[i][j][k].Y$ and $G[i][j-1][k].Z \leq G[i][j][k].Z \Rightarrow G[i-1][j][k].X \leq G[i][j][k].X$;
7. $\forall G[i][j][k] \in \mathbf{G},\ i > 0, G[i][j][k].X \leq G[i-1][j][k].X$;
8. $\forall G[i][j][k] \in \mathbf{G},\ i > 0, G[i-1][j][k].X = G[i][j][k].X \Rightarrow G[i][j][k].Y \leq G[i-1][j][k].Y$;
9. $\forall G[i][j][k] \in \mathbf{G},\ i > 0, G[i-1][j][k].X = G[i][j][k].X$ and $G[i][j][k].Y = G[i-1][j][k].Y \Rightarrow G[i][j][k].Z \leq G[i-1][j][k].Z$.

Our proposed data structure is independent of the sorting algorithm used, as long as the rules above are always, eventually or even partially achieved during simulation, depending on the desired neighborhood precision.

## 5. Architecture environment

The proposed architecture implements fluid simulations using a novel GPU computing solution, based on the neighborhood grid data structure. Our solution allows a high performance increase during simulation. This architecture environment is explained in the following subsections.

### 5.1. Execution workflow

Our proposed workflow can be described as follows: at the beginning, the architecture sorts all particles based on their positions. The sorted particles gather their neighborhoods according to the radius and the system calculates their pressure and density. Based on these results, the internal forces are computed for each particle, and all external forces acting on it are added, such as gravity and users' input values. Finally the system calculates the new velocities and positions, integrating the whole system. This process is shown in Fig. 3, where fluid simulations tasks are separated for better understanding.

### 5.2. Neighborhood gathering

Fluids are represented using a set of particles that interact with each other. This is always true for fluids' particles, as each of them

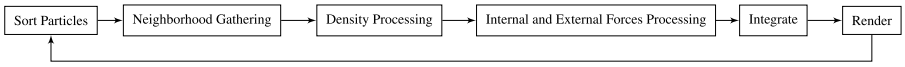



**Fig. 3.** The main loop workflow of the proposed architecture.

needs to find its neighborhood particles for calculating variables such as pressure and density, according to the SPH method. In order to keep the processing entirely in the GPU, the position of the particles is mapped as textures for the vertex shader, which is responsible for rendering. This information is stored in 3D arrays (the neighborhood grids), where each position holds the entire data for an individual particle.

The CUDA implementations of the neighborhood grid have been applied with some memory optimizations in order to speedup the data structure [7]. In the CUDA architecture there are many kind of memories, and different types of optimizations can be done in order to speedup the processing. Data alignment feature is important, so that structures can benefit from the read processing. For the main memory, called global memory, it is necessary to model data in a coalesced manner in order to optimize the memory access. We also use the texture memory, which is a read only memory and that can be accessed using fewer instructions, and shared memory which is a memory close to the register and can be read and written by any thread in the same block.

One of the techniques used in our approach is the data alignment. Since the device can read 4-byte, 8-byte, or 16-byte words from global memory into registers in a single instruction, all data has size multiple of these values. So, for our Neighborhood grid, we have a float4 (i.e., an array with four float numbers) indicating the position with the pressure, and another float4 for the velocity with the density of the particle. Also all this data is double buffered, so that it can benefit from the texture memory, using a ping–pong technique [16]. This technique works as follows: we read the old values from texture arrays while we write the updated values into the ping–pong array. These arrays reverse their roles every update as readable textures and writable arrays in the global memory.

The global memory bandwidth is more efficiently explored when the simultaneous memory accesses made by the threads in a half-warp (in CUDA, a warp is a sequence of 32 threads in a thread block; a half-warp is the first or the last 16 threads of a warp) can be modeled as coalesced into one or two memory transactions. This happens if the following rules are true: all the threads access have 32/64/128-bit words, resulting in one 32/64/128-byte memory transaction, and all the memory accessed lies in the same memory segment, which must be equal to the transaction size. The neighborhood grid is naturally organized to use coalesced memory access, since the neighborhood grid sorts the data according to the access of the neighbors. This organization also allows an effective usage of the GPUs shared memory, because most of the neighbors memory can be used inside the same block, minimizing the processing time, since it requires fewer warps to access [16]. Doing so each thread copies its data to the shared memory and each copied particle data at the shared memory can be used by all other threads in that block. Also the access of this memory is organized in a way to avoid bank conflicts, which are multiple simultaneous access on the same bank memory. This way the Neighborhood Grid can benefit from the shared memory, without suffer from the bank conflicts, which can slow the processing.

Different state settings that rule the fluid simulation must be updated at every frame, according to the simulation behavior. During fluid simulation, each particle has its state composed of a float4 for the position, a float4 for the force resultant, a float2 for density/pressure and a float4 for its velocity. These data are stored at GPU's global memory. The fluid simulation is performed entirely on the GPU, avoiding data transfers between CPU–GPU, since it is typically the bottleneck in most of the CPU–GPU systems.

In this work, the simulation of each particle is mapped to one GPU thread for both the sorting and simulation steps, so it is important to mention that the grids are double buffered; consequently each of these tasks does not write data over the input structures that can still be read by the other parallel GPU threads. This work could also use atomic operations for the grid operations, but these kinds of operations are still very costly for massive simulations.

#### 5.2.1. Sorting mechanism

We use a bitonic sort [28], which makes a full sort in each dimension. The bitonic sort is a simple parallel sorting algorithm that is very efficient when sorting a small number of elements [3], which is our case since the sort strategy is applied to each dimension separately. The used implementation is an optimized and adapted version based on a previous work of NVidia [28]. This sort is divided in 3 passes, one for each dimension ($X$, $Y$ and $Z$).

The complexity of this algorithm is $O(n \log(n)^2)$ where $n$ is the number of elements to sort in sequential time. These comparisons are performed by $m$ CUDA threads making such parallel implementation of the algorithm to perform with a complexity of $O(\log(n)^2)$, if there were $m = n$ stream processors on the GPU.

This sorting stage does not make a full sort on the neighborhood grid but only a full sort on each dimension ($X$, $Y$ and $Z$) of the grid. If a change is made, for instance, in one particle position on the $Y$ pass, another pass for the $X$ would be needed in order to keep the neighborhood grid with a full sort. In tests for consistency the authors have seen that this misalignment is very small, usually less than 1% of the particles changes place in one step of the simulation. In the next step this error will be fixed, and the use of a full sort on the neighborhood grid would impose some loss in performance without visible gain in the simulation.

## 6. Results

This section presents the results obtained from the proposed data structure. A series of screenshots of a fluid animation with 65k particles, built with the presented architecture, can be seen in Fig. 4.

In order to evaluate our proposed data structure for SPH particle simulation, this work has implemented the traditional uniform grid with an index sort method in the GPU with the use of CUDA, which is the same implementation proposed at [33,18]. This implementation uses a uniform grid to subdivide the bounding box of the world to be simulated in cells. In order to optimize this structure, especially for GPUs, the particles are sorted into an index array according to the cells' indexes.

### 6.1. Simulation configuration

For the tests, a PC with an Ubuntu 10.10 Linux distribution equipped with an Intel i7 3.07 GHz using 8 GB of RAM and a GPU NVidia GeForce GTX 580 with 512 cores was used. Simulations with different configurations were performed. Fluid rendering is done in two ways: by applying ray-tracing in the particles using POV-Ray[1] (for Fig. 3), and by using simple primitives for performance tests, which consist in simple spheres rendered using a vertex shader. To assure that results are consistent, each test was

[1] POV-Ray—The Persistence of Vision Raytracer, www.povray.org.

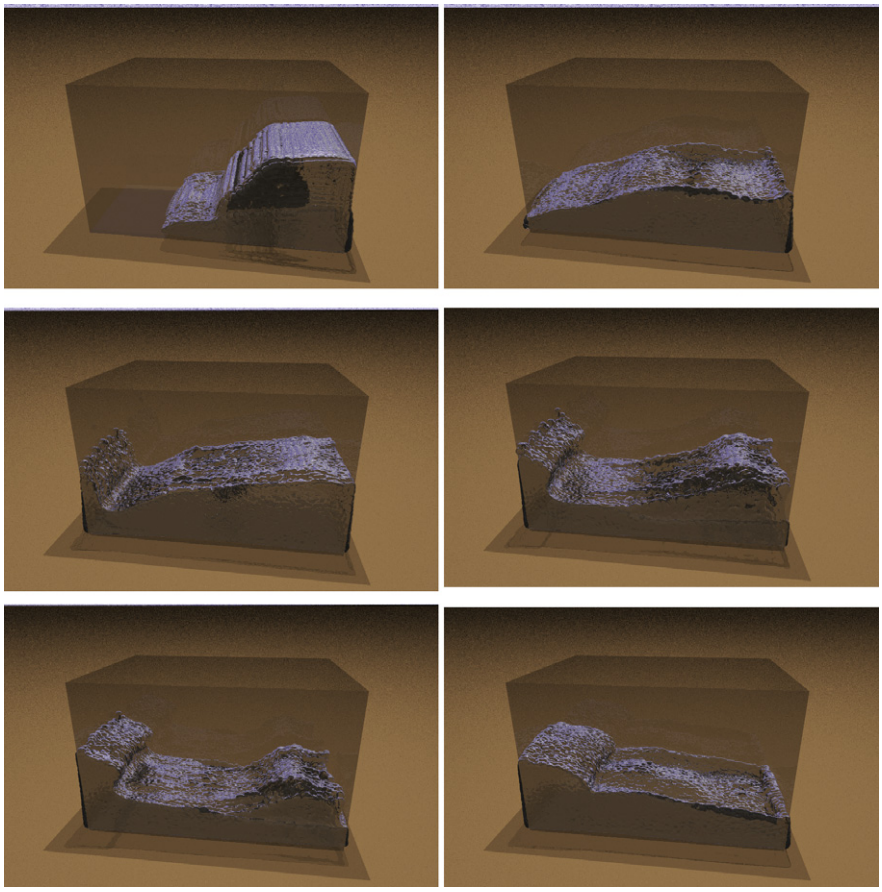


**Fig. 4.** Screenshots of the simulation.

repeated 10 times and the standard deviation of the average times was confirmed to be within 5%.

The test scene consists of a fluid with different number of particles and a box with fixed size, where the fluid will be dropped. The interaction with the box is done apart from the fluid calculation, during the integration step. When a particle collides with this box, it reflects its velocity with a damping factor $d$. The fluid particles are updated at a fixed speed of 30 interactions per second and the rendering is processed at every frame. The fluid is dropped in different configurations to test different initial conditions, such as the classical dam-break, where the fluids are concentrated in a dam and the dam is released, and drop, where the fluids are dropped from a high spot into a receptacle. To evaluate the scalability of the data structure, the number of particles being simulated varied from thousands to a million. The Moore neighborhood radius was set to 4 (which was the radius with less mean error). The cell size of the traditional uniform grid influences the total number of visited neighbors in each interaction; in our tests we have used a size of $h = 1$ for this (which was the size with better performance). The fluid particle radius was also set to 1. The grid size varied from $16 \times 16 \times 16$ to $128 \times 128 \times 128$. The fluid was also influenced by the gravity acceleration, which was set to 9.8 $m/s^2$. The damping factor $d$ was set to 0.9.

### 6.2. Test results

Table 1 presents the results for the simulations using the neighborhood grid method and the uniform grid with index sort considering a varying number of particles. In both methods, all processing is done in the GPU. *Maintenance* represents the time elapsed for the maintenance of the neighborhood gathering algorithm (sorting the neighborhood grid or building the uniform grid). *Simulation* is the time spent in the simulation processing (density, force and integration calculations). Here, the time for rendering the particles is not taken into account. Also, fps represents the *frames per second* which measures the time necessary to update and render the simulation (of simple particles spheres). *Speedup* is defined by the relation $S = \frac{X_1}{Y_2}$, being $X_1$ the fps for the Neighborhood Grid and $Y_2$ the fps for the Uniform Grid. As expected, the fluid simulation using the neighborhood grid method presents better results than the simulation using the uniform grid with index sort. Also, it can be seen that the time spent with the maintenance of the neighborhood grid is lower than the time spent for building a uniform grid and maintaining it. Also the simulation time is lower, since the neighborhood grid has a much better memory organization (by keeping neighbors close together in memory it avoids memory bank conflicts and maintains a coalesced memory), achieving an expressive speedup.

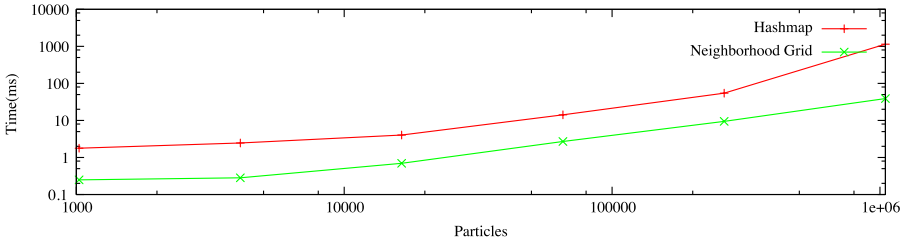
Table 2 shows the memory usage for the presented data structure. From these results, it is possible to see that our data structure uses much less memory space, since it does not need extra memory to keep the data in a linear form. This data is required for the extra arrays for the index sort and the uniform grid in the GPU. Table 2 also shows the mean error and variance as a function of the number of particles keeping the radius equal to 4 (which produced the smaller error with greater performance). This mean error is calculated by processing the particles with the neighborhood grid and comparing their positions with the ones calculated by the uniform grid method (by calculating the distance between both). From these results, it can be seen that the error is small (lesser than 0.05, while the radius size is 1.0) as well as the variance. The mean error from density calculation and force steps were also measured, being not included in this work, since they follow the same evolution as the mean error included in Table 2, absent 10% for the density calculus error.

**Table 1**
Time in milliseconds spent with tasks by each gathering method.

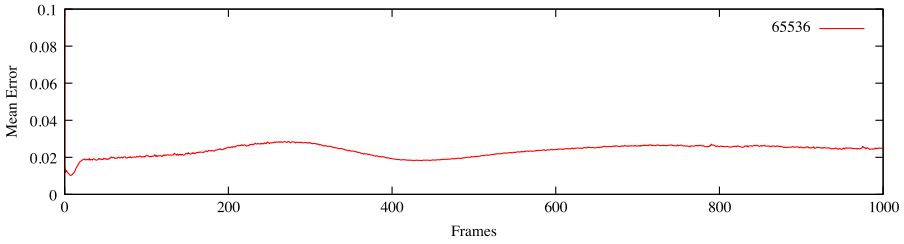
| # particles | Uniform grid with index sort | | | Neighborhood grid | | | Speedup |
|---|---|---|---|---|---|---|---|
| | Maintenance | Simulation | fps | Maintenance | Simulation | fps | |
| 1,024 | 0.0429 | 0.71 | 703 | 0.015 | 0.23 | 1388 | 1.97 |
| 4,096 | 0.1175 | 1.68 | 571 | 0.015 | 0.23 | 1201 | 2.10 |
| 16,384 | 0.1281 | 6.55 | 377 | 0.016 | 0.79 | 826 | 2.19 |
| 65,536 | 0.1728 | 60.49 | 131 | 0.017 | 2.94 | 384 | 2.95 |
| 262,144 | 0.1891 | 127.33 | 30 | 0.018 | 10.84 | 109 | 3.63 |
| 1,048,576 | 0.3166 | 401.88 | 3 | 0.026 | 25.27 | 27 | 9 |

**Table 2**
Memory use when using the uniform grid with index sort and the neighborhood grid.

| # particles | Memory use | | Error | |
|---|---|---|---|---|
| | Uniform grid with index sort | Neighborhood grid | Mean error | Variance |
| 1,024 | 65.9 kB | 5.6 kB | 0.000642 | 0.000004 |
| 4,096 | 264 kB | 22.4 kB | 0.002437 | 0.000108 |
| 16,384 | 1.23 MB | 89.6 kB | 0.011948 | 0.000393 |
| 65,536 | 1.92 MB | 360 kB | 0.020246 | 0.000689 |
| 262,144 | 3.5 MB | 1.4 MB | 0.025120 | 0.001765 |
| 1,048,576 | 7.7 MB | 5.6 MB | 0.035663 | 0.001940 |




(a) Graph of the performance versus number of particles in a log scale.




(b) Graph of the mean error versus time.

**Fig. 5.** Evolution of the simulation in milliseconds (a) and in error (b).

Fig. 5(a) shows the evolution in a log scale of the performance of the simulation in milliseconds with both gathering methods varying the number of particles. This plot shows that the neighborhood grid data structure is faster for fluid simulation and that it scales better than the uniform grid with index sort. Fig. 5(b) shows the mean error of the particles position during 1000 frames of the simulation with 65k particles in the scene. From these results it can be seen that the error does not increase considerable during the simulation, maintaining a low variance of the error. The mean accumulated error from these 1000 frames is 3.61 with a 0.97 variance. This error was calculated by processing 1000 frames with each structure and taking the differences positions of the particles. This error is small considering that the radius of the particle is 1.0.

The number of particle neighbors varies for each particle. In our simulation tests, we have noticed that the uniform grid with index sort visits from 100 up to 275 particles in order to use from 17 up to 35 particle neighbors in its calculation. In turn, the neighborhood grid always visits 102 particles in order to use from 17 to 35 neighbors. Finally, a test comparing the presented architecture implemented in the GPU and the uniform grid implemented in the CPU shows a speedup of 100 times (with 100k particles).

## 7. Conclusion and future work

We have presented a novel data structure (neighborhood grid) which can be employed for real-time SPH fluid simulations on GPUs. With the proposed data structure we are able to simulate and render up to 1 million fluid particles in real-time frame rate, while the uniform grid with index sort method and most works in the literature barely maintains interactivity in the simulation. Moreover, with the neighborhood grid and bitonic sort, configured with a radius of 4, a visual simulation similar to the uniform grid with index sort method can be experienced with significant speedup. The authors of this work suggest using this configuration to achieve the best visual and performance at the fluid simulation.

The results shows that performing fluid simulation using the neighborhood grid method increases simulation performance, obtaining a speedup of more than 9 times over the traditional spatial hashing simulation on GPU.

Even though this structure does not fulfill all the steps of a SPH fluid animation, it is still good enough for animation of fluids, since it has a small error and the behavior of the fluids appears to the human eye similar to the SPH implementation.

## References


[1] B. Adams, M. Pauly, R. Keiser, L.J. Guibas, Adaptively sampled particle fluids, ACM Trans. Graph. 26 (2007).
[2] S. Bayraktar, U. Güdükbay, B. Özgüç, GPU-based neighbor-search algorithm for particle simulations, J. Graph. GPU Game Tool. 14 (1) (2009). http://dx.doi.org/10.1080/2151237X.2009.10129272.
[3] G.E. Blelloch, C.G. Plaxton, C.E. Leiserson, S.J. Smith, B.M. Maggs, M. Zagha, An experimental analysis of parallel sorting algorithms, Theory Comput. Syst. (1998).
[4] J.M. Cohen, S. Tariq, S. Green, Interactive fluid-particle simulation using translating Eulerian grids, in: I3D'10: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, ACM, New York, NY, USA, 2010, pp. 15–22.
[5] M. Desbrun, M. Paule Gascuel, Smoothed particles: a new paradigm for animating highly deformable bodies, in: Computer Animation and Simulation 96 (Proceedings of EG Workshop on Animation and Simulation), Springer-Verlag, 1996, pp. 61–76.
[6] D. Enright, S. Marschner, R. Fedkiw, Animation and rendering of complex water surfaces, ACM Trans. Graph. 21 (2002) 736–744.
[7] G. Falcao, V. Silva, L. Sousa, How GPUs can outperform ASICs for fast LDPC decoding, in: Proceedings of the 23rd International Conference on Supercomputing, ICS'09, ACM, New York, NY, USA, 2009, pp. 390–399.
[8] N. Foster, R. Fedkiw, Practical animation of liquids, in: SIGGRAPH'01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, ACM, New York, NY, USA, 2001, pp. 23–30.
[9] N. Foster, D. Metaxas, Realistic animation of liquids, Graph. Models Image Process. 58 (1996) 471–483.
[10] D. Gao, J.A. Herbst, Alternative ways of coupling particle behaviour with fluid dynamics in mineral processing, Int. J. Comput. Fluid Dyn. 23 (2009) 109–118.
[11] R.A. Gingold, J.J. Monaghan, Smoothed particle hydrodynamics-theory and application to non-spherical stars, Mon. Not. R. Astron. Soc. 181 (1977) 375–389.
[12] P. Goswami, P. Schlegel, B. Solenthaler, R. Pajarola, Interactive SPH simulation and rendering on the GPU, in: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA'10, Eurographics Association, Aire-la-Ville, Switzerland, 2010, pp. 55–64.
[13] S. Green, Particle-based fluid simulation, 2008. http://developer.download.nvidia.com/presentations/2008/GDC/GDC08_ParticleFluids.pdf (17/10/2011).
[14] T. Harada, S. Koshizuka, Y. Kawaguchi, Smoothed particle hydrodynamics on GPUs, in: Proceedings of XXV Computer Graphics International, CGI'07, 2007, pp. 1–8.
[15] M. Ihmsen, N. Akinci, M. Becker, M. Teschner, A parallel SPH implementation on multi-core cpus, Comput. Graph. Forum 30 (2011) 99–112.
[16] M. Joselli, J. Ricardo da Silva, M. Zamith, E. Clua, M. Pelegrino, E. Mendonca, Techniques for designing GPGPU games, in: Games Innovation Conference (IGIC), 2012 IEEE International, IEEE, pp. 1–5.
[17] M. Joselli, E.B. Passos, M. Zamith, E. Clua, A. Montenegro, B. Feijó, A neighborhood grid data structure for massive 3D crowd simulation on GPU, in: SBGAMES, IEEE, 2009, pp. 121–131.
[18] J.R.d.S. Junior, M. Joselli, M. Zamith, M. Lage, E. Clua, E. Soluri, N. Tecnologia, An architecture for real time fluid simulation using multiple GPUS, in: SBGames 2012, SBC.
[19] P. Kipfer, R. Westermann, Realistic and interactive simulation of rivers, in: GI'06: Proceedings of Graphics Interface 2006, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 2006, pp. 41–48.
[20] P. Kipfer, R. Westermann, Realistic and interactive simulation of rivers, Proc. Graph. Interface 2006 (2006) 41–48.
[21] S. Kurose, S. Takahashi, Constraint-based simulation of interactions between fluids and unconstrained rigid bodies, in: Proceedings of Spring Conference on Computer Graphics, 2009, pp. 197–204.
[22] L.B. Lucy, A numerical approach to the testing of the fission hypothesis, Astron. J. 82 (1977) 1013–1024.
[23] J.J. Monaghan, Smoothed particle hydrodynamics, Annu. Rev. Astron. Astrophys. 30 (1992) 543–574.
[24] J.J. Monaghan, Simulating free surface flows with SPH, J. Comput. Phys. 110 (1994) 399–406.
[25] J.P. Morris, P.J. Fox, Y. Zhu, Modeling low Reynolds number incompressible flows using SPH, J. Comput. Phys. 136 (1997) 214–226.
[26] M. Müller, D. Charypar, M. Gross, Particle-based fluid simulation for interactive applications, in: SCA'03: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Eurographics Association, Aire-la-Ville, Switzerland, 2003, pp. 154–159.
[27] M. Müller, S. Schirm, M. Teschner, B. Heidelberger, M. Gross, Interaction of fluids with deformable solids, Comput. Anim. Virtual Worlds 15 (2004) 159–171.
[28] nVidia, Bitonic Sort Demo, Technical Report, 2007, Avalible at: http://www.nvidia.com/content/cudazone/cuda_sdk/Data-Parallel_Algorithms.html#bitonic.
[29] E.B. Passos, M. Joselli, M. Zamith, E.W.G. Clua, A. Montenegro, A. Conci, B. Feijo, A bidimensional data structure and spatial optimization for supermassive crowd simulation on GPU, Comput. Entertain. 7 (2010) 60:1–60:15.
[30] T.J. Purcell, C. Donner, M. Cammarano, H.W. Jensen, P. Hanrahan, Photon mapping on programmable graphics hardware, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, 2003, pp. 41–50.
[31] J. Ren, J. Ouyang, B. Yang, T. Jiang, H. Mai, Simulation of container filling process with two inlets by improved smoothed particle hydrodynamics (SPH) method, Int. J. Comput. Fluid Dyn. 25 (2011) 365–386.
[32] P. Sarkar, A brief history of cellular automata, ACM Comput. Surv. 32 (2000) 80–107.
[33] J. Ricardo da Silva Junior, E.W. Gonzalez Clua, A. Montenegro, M. Lage, M.d.A. Dreux, M. Joselli, P.A. Pagliosa, C.L. Kuryla, A heterogeneous system based on GPU and multi-core CPU for real-time fluid and rigid body simulation, Int. J. Comput. Fluid Dyn. 26 (2012) 193–204.
[34] B. Solenthaler, R. Pajarola, Density contrast SPH interfaces, in: Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA'08, Eurographics Association, Aire-la-Ville, Switzerland, 2008, pp. 211–218.
[35] J. Stam, Stable fluids, in: SIGGRAPH'99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999, pp. 121–128.
[36] D. Stora, P.-O. Agliati, M.-P. Cani, F. Neyret, J.-D. Gascuel, Animating lava flows, in: Proceedings of the Conference on Graphics Interface'99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 203–210.
[37] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, M.H. Gross, Optimized spatial hashing for collision detection of deformable objects, VMV (2003) 47–54.
[38] L. Verlet, Computer "experiments" on classical fluids. II. Equilibrium correlation functions, Phys. Rev. 165 (1968) 201–214.
[39] Y. Zhang, B. Solenthaler, R. Pajarola, Adaptive sampling and rendering of fluids on the GPU, in: Eurographics/IEEE VGTC Symposium on Point-Based Graphics, New York, NY, USA, pp. 137–146.
[40] Y. Zhang, B. Solenthaler, R. Pajarola, Adaptive sampling and rendering of fluids on the GPU, in: Proceedings of the Fifth Eurographics/IEEE VGTC Conference on Point-Based Graphics, SPBG'08, Eurographics Association, Aire-la-Ville, Switzerland, 2008, pp. 137–146.


**Mark Joselli** is the Chief-Developer-Officer of Nullpointer, a Brazilian Software Development Company. He teaches game development and mobile development at PUC-PR. He is also a Game Developer, mobile developer and a Researcher in GPGPU. He also created the first Framework for GPGPU games.

He is a Doctor (2013) and a Master (2007) in Computer Science from Federal Fluminense University. He is an Industrial Electronic Engineer by the Federal Center for Technological Education Celso Suckow da Fonseca (CEFET-RJ—2005) and also a bachelor in philosophy by Unisul (2013).

**José Ricardo da S. Junior** is B.Sc. in Systems Analysis at Estacio de Sa University (2005), M.Sc. in Computer Science at Fluminense Federal University (2010), master-vocational Game Development and 3D applications by the Veiga de Almeida University (2007) and teaching-medio-second-degree flame by the College (1998).

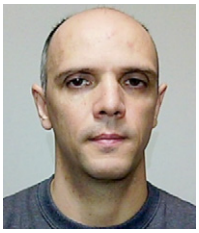
Acting on the following topics: Computational Fluid Dynamics, Fluid Simulation, Load Balancing GPU/CPU, Smoothed Particle Hydrodynamics.

**Esteban W. Clua** has M.Sc. and Ph.D. in Computer Science. He is currently a professor at the Computing Institute at Federal Fluminense University (UFF) and general coordinator of MediaLab—UFF. He is one of the founders of Brazilian Simposium in Games and Digital Entertainment (SBGames) by Brazilian Computer Society (SBC), council member of the special committee of the SBC games and entertainment and a pioneer in the area scientific research in games and digital entertainment. In 2007 he received the prize as the largest contributor ABRAGAMES academy for digital games industry in Brazil.

**Anselmo Montenegro** has B.Sc. degree in Computer Science at Federal Fluminense University (UFF—1995), M.Sc. (1997) and Ph.D. in Computer Science at PUC-Rio (Catholic University of Rio de Janeiro—2003). He is currently Associate Professor of UFF.

He has experience in Computer Science, with emphasis on graphics processing, acting on the following topics: models of global illumination in GPU, visualization, image-based, mathematical morphology, space carving, photography and 3d information visualization.

**Marcos Lage** has B.Sc. in Mathematics at Universidade do Estado do Rio de Janeiro (2004), M.Sc. (2006) and Ph.D. (2009) in Applied Mathematics at PUC-Rio de Janeiro. He is currently professor of at Universidade Federal Fluminense. His main research area is the numerical simulation of fluids. He is also interested in topological data structures, vector field's reconstruction and geometric modeling.



**Paulo Pagliosa** has Ph.D. in Structural Engineering from the School of Engineering of São Carlos at Universidade de São Paulo (1998). He is currently an associate professor of Computer Science in the College of Computing at Universidade Federal de Mato Grosso do Sul, and has experience in visualization and simulation, acting on the following topics: GPGPU, computational mechanics, geometric processing, and physics for games.