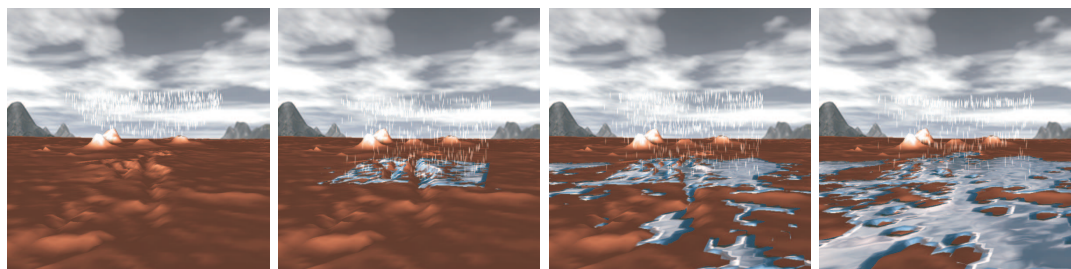


A Cellular Automata Framework for Real Time Fluid Animation

Sicilia F. Judice and Bruno Barcellos S. Coutinho and Gilson A. Giraldi
National Laboratory for Scientific Computing



Abstract

In this work, we focus on fluid modeling and animation through Lattice Gas Cellular Automata (LGCA) for computer games. LGCA are discrete models based on point particles that move on a lattice, according to suitable and simple rules in order to mimic a fully molecular dynamics. In this paper, we combine the advantage of the low computational cost of LGCA and its ability to mimic the realistic fluid dynamics. The new animating framework is composed by two LGCA models: (a) A 3D fluid animation technique; (b) A GPU surface flow animation over terrain models. In this work we highlight the advantages of our proposal for computer games applications. In the experimental results we emphasize the simplicity and power of the proposed models when combined with efficient techniques for rendering.

Keywords:: Real-time Simulation, Cellular Automata, Fluid Animation, Terrain Models

Author's Contact:

{sicilia,lsdelphi,gilson}@lncc.br

1 Introduction

Physically-based techniques for the animation of natural elements like fluids (gas or liquids), elastic, plastic and melting objects, among others, have taken the attention of the computer graphics community. The motivation for such interest rely in the potential applications of these methods and in the complexity and beauty of the natural phenomena that are involved. In particular, techniques in the field of Computational Fluid Dynamics (CFD) have been applied for fluid animation in applications such as virtual surgery simulators and visual effects [Müller et al. 2004a; Müller et al. 2004b; Müller et al. 2003].

The traditional fluid animation methods in computer graphics rely on a top down viewpoint that uses 2D/3D mesh based approaches motivated by the Eulerian methods of Finite Element (FE) and Finite Difference (FD), in conjunction with Navier-Stokes equations of fluids [Stam 2003b; Foster and Metaxas 1997; Hirsch 1988].

Animation of fluids is important for computer games applications in order to immerse players into plausible virtual worlds [Gam 2004]. However, real time is a fundamental requirement in this area which is the main challenge for traditional CFD methods due to the cost of the computational simulation [Stam 2003a].

Recently, we demonstrated the advantages of changing the viewpoint to the bottom up models of the Lattice Gas Cellular Automata (LGCA) [Xavier et al. 2005; Barcellos et al. 2007]. These are discrete models based on point particles that move on a regular grid structure (**lattice**), according to suitable and simple rules in order to mimic a fully molecular dynamics [Frisch et al. 1987a]. Particles can only move along the edges of the lattice and their interactions are based on simple collision rules. Such framework needs

low computational resources for both the memory allocation and the computation itself. Such models have been applied for scientific application in two-phase flows description (gas-liquid systems, for example), numerical simulation of bubble flows [Inamuro et al. 2004], among others.

Besides, a multiscale technique was applied to demonstrated that a particular type of LGCA, the so called **FHP** model, can reproduce Navier-Stokes behaviors for 2D fluids [Frisch et al. 1986; Wolfram 1996]. However, there is no need to solve Partial Differential Equations (PDEs) to obtain a high level of description. Besides being fast and realistic from the physical viewpoint, it is based on binary variables, and so, there are no numerical instabilities at all [Doolen 1990]. In [Giraldi et al. 2005] we show the FHP capabilities for animating 2D fluid-fluid interactions. In [Barcellos et al. 2007] we demonstrated the advantages of a technique inspired in the FHP model for surface water flow simulation in digital terrain models.

In this paper we propose a framework for real time fluid animation based on two elements: (a) A three dimensional fluid simulation model based on the FHP and interpolation techniques; (b) The surface flow simulation in Digital Terrain Models (DTMs) proposed in [Barcellos et al. 2007]. In the former, our proposal consists of regularly distribute independent FHP planes along the x and z directions and perform simple interpolations to generate a 3D macroscopic flow. The latter is a LGCA which is based on (water) particles which can move over the terrain surface but the projections of their displacements are constrained to the lattice directions. The DTM is a piecewise linear approximation of the terrain surface built as a digital elevation model (**DEM**); that means, there is a lattice that keeps the elevation of the terrain at each grid node. We can apply the 3D fluid simulation method to animate rainfall. On the other hand, we can simulate flood and watershed rainfall by using simple interaction model between the fluid and the terrain.

The main contributions of this work is the development of a particle based framework, combining LGCA techniques and DTMs to create realistic animations of systems that involve 3D fluid and fluid-terrain interaction for computer graphics and computer games applications.

The paper is organized as follows. Section 2 gives a review of related works. Section 3 describes the FHP model and our extension for 3D. In sections 4 and 5 we describe our technique for simulating surface water flow over terrains and its GPU implementation. The interaction between models is discussed on section 6. Section 7 presents experimental results. In section 8 we present the conclusions and future works.

2 Related Work

This work focuses on the animation of three dimensional fluids and fluid-terrain interaction. The former involves numerous works that can be coarsely classified in non-physically and physically based models [Iglesias 2004; Deussen et al. 2004]. Our work belongs to the latter class, which can be subdivided in PDEs and Lattice based techniques [Frisch et al. 1987b; Iglesias 2004]. PDEs methods in-

cludes continuous fluid equation, like the Navier-Stokes ones, and numerical techniques based on discretization approaches that can be Lagrangian Smoothed Particle Hydrodynamics (SPH) [Liu and Liu 2003], method of characteristics [Stam 1999], Moving-Particle Semi-Implicit [Premoze et al. 2003] or Eulerian (Finite Element) ones [Foster and Metaxas 1997].

Lattice based techniques, like HPP, FHP and Lattice Boltzmann methods, work following a different viewpoint [Frisch et al. 1987b; Benzi et al. 1992]. For instance, in the case of HPP and FHP, instead of applying continuous mechanics (and, consequently, PDEs) principles, they model the system as a set of point particles, that move on a lattice, interacting according to suitable and simple rules in order to mimics a fully dynamics [Frisch et al. 1987b]. These are *bottom up* approaches in which the macroscopic behavior of the fluid can be recovered by interpolation techniques [Frisch et al. 1987b].

Lattice models have a number of advantages over more traditional numerical methods, particularly when fluids mixing and phase transitions occur [Rothman and Zaleski 1994]. The simulation is always performed on a regular grid and can be efficiently implemented on a massively parallel computer. Solid boundaries and multiple fluids can be introduced in a straightforward manner and the simulation is performed equally efficiently, regardless of the complexity of the boundary or interface [Buick et al. 1998]. In addition there are not numerical stability issues because the evolution follows integer arithmetic. However, system parametrization (viscosity, for example) is a difficult task in such lattice models and they are less realistic than PDE based models.

For the purpose of game application, real time is a fundamental requirement. So, the trad-off between realism and frame rate becomes the main challenge. Therefore, Navier-Stokes equations solvers, based on finite difference approaches, with special care about stability and speed were proposed [Stam 2003a] as well as Lattice Boltzmann methods (LBM) on graphics hardware [Zhao et al. 2007] and real time techniques for simulating atmospheric effects [Wenzel 2006].

SPH has been also applied for fluid simulation and interaction of the fluid with rigid bodies through penalty techniques, contact (pressure) forces, and a discrete angular velocity representation ([Gam 2004], Vol. 6, page 189). Besides, Linearized Bernoulli's equations and convolution techniques for discretization have been applied for real time computer-generated ocean surface ([Gam 2004], Vol. 4, page 256).

Interaction between fluids models and digital terrains is a subclass of fluid-surface interaction [Ye et al. 1996]. It can be addressed by hybrid methods in which the fluid is a continuum medium, simulated by Navier-Stokes plus SPH or grid based techniques, and the surface is represented as a discrete one [GENEVAUX et al. 2003; Solenthaler et al. 2007; Batty et al. 2007]. These approaches deal with the specific problem of preventing the leaking of fluid across the polygonal surface [Guendelman et al. 2005; Bridson et al. 2002].

In addition, fluid flows can be simulated on 2D manifolds represented by (continuous) subdivision surfaces that have a natural quad patch parametrization [Stam 2003b]. Besides, a hybrid particle and implicit surface approach to simulating water was proposed in [Foster and Fedkiw 2001], which led to the particle level set method [Enright et al. 2002].

Finally, rendering techniques must be applied to ensure the desired level of realism but keeping the frame rate suitable for game applications. Realistic rendering can be properly addressed through several algorithms incorporating usual refraction, reflection and Fresnel terms ([Gam 2004], Vol. 1, page 583), refraction approaches based on perturbing the texture coordinates used in a texture lookup of an image of the nonrefractive objects in the scene ([GPU 2005], Chap. 19) and accelerated volume rendering on GPU [Kriger and Westermann 2005].

3 FHP

The FHP was introduced by Frisch, Hasslacher and Pomeau [Frisch et al. 1986] in 1986 and is a model of a two-dimensional fluid. It can be seen as an abstraction, at a microscopic scale, of a fluid. The FHP model describes the motion of particles traveling in a discrete space and colliding with each other. The space is discretized in a hexagonal lattice.

The FHP particles move in discrete time steps, with a velocity of constant modulus, pointing along one of the six directions of the lattice. The dynamics is such that no more than one particle enters the same node at the same time with the same velocity. This restriction is the *exclusion principle*; it ensures that six Boolean variables at each lattice node are always enough to represent the microdynamics.

The velocity modulus is such that, in a time step, each particle travels one lattice spacing and reaches a nearest-neighbor node. When exactly two particles enter the same node with opposite velocities, both of them are deflected by 60 degrees so that the output of the collision is still a zero momentum configuration with two particles. The deflection can occur to the right or to the left, indifferently, as shown in Figure 1. For symmetry reasons, the two possibilities are chosen randomly, with equal probability.

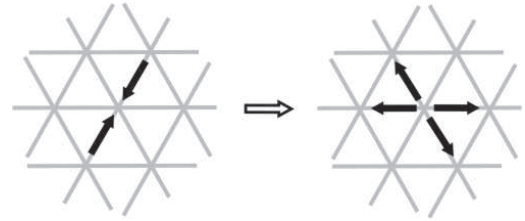


Figure 1: The two-body collision in the FHP Source.

When exactly three particles collide with an angle of 120 degrees between each other, they bounce back to where they come from (so that the momentum after the collision is zero, as it was before the collision). Both two- and three-body collisions are necessary to avoid extra conservation laws. For all other configurations no collision occurs and the particles go through as if they were transparent to each other.

The full microdynamics of the FHP model can be expressed by evolution equations for the occupation numbers defined as the number, $n_i(\mathbf{r}, t)$, of particle entering node $\mathbf{r}$ at time t with a velocity pointing along direction $\vec{c}_i$

$$\vec{c}_i = \left(\cos \frac{2\pi i}{6}, \sin \frac{2\pi i}{6} \right), \quad (1)$$

where $i = 1, 2, \dots, 6$ labels the six lattice directions, as shown in Figure 2.

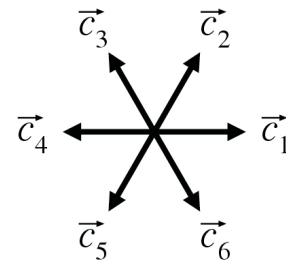


Figure 2: The six lattice directions.

The numbers n_i can be 0 or 1. We also define the time step as Δ_t and the lattice spacing as Δ_r . Thus, the six possible velocities $\vec{v}_i$ of the particles are related to their directions of motion by

$$\vec{v}_i = \frac{\Delta_r}{\Delta_t} \vec{c}_i. \quad (2)$$

The microdynamics of a LGCA is written as

$$n_i(\mathbf{r} + \Delta_r \vec{c}_i, t + \Delta_t) = n_i(\mathbf{r}, t) + \Omega_i(n(\mathbf{r}, t)) \quad (3)$$

where Ω_i is called the collision term [Chopard and Droz 1998].

Now we propose an extension to 3D, using the 2D FHP model explained above. In practice, the system of a given cellular automata rule cannot deal with an infinite lattice, it must be finite and have boundaries [Chopard and Droz 1998]. So, first we must define a domain in the three dimensional space, where the microscopic particles can evolve. Then, our proposal consists of regularly distribute planes along the x and z axis, like in Figure 3. Each of these planes is a system whose cellular automata rule is the 2D FHP.

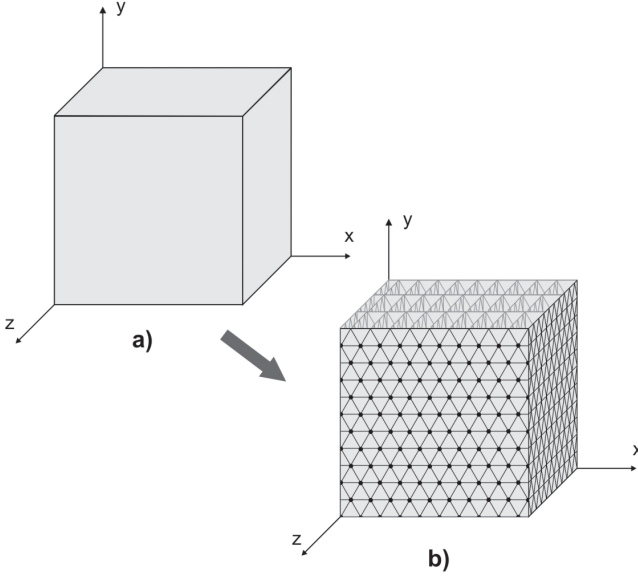


Figure 3: (a) Domain in 3D Space and (b) the distribution of 2D FHP planes.

Once simulated the two dimensional FHP in each plane independently, we perform a simple interpolation to generate a 2D macroscopic flow in each plane. In this step, we add new nodes to the FHP grid in order to complete a rectangular grid in each plane, as pictured in Figure 4.

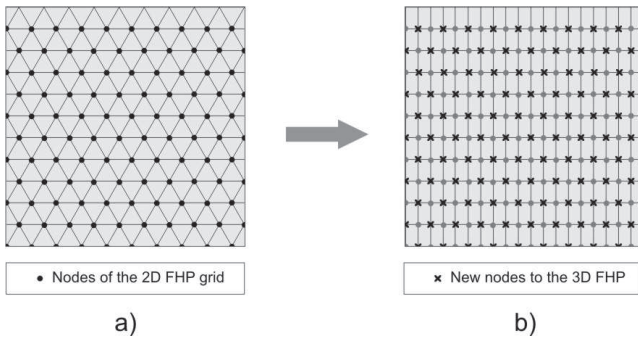


Figure 4: Extend the (a) FHP grid to generate a (b) rectangular mesh.

Following the usual definition of statistical mechanics, we compute the macroscopic density in each node (x_i, y_i, z_i) of the plane $x = x_i$ through the expression:

$$\rho_x(x_i, y_i, z_i, t) = \sum_{j \in V} \sum_{k=1}^6 n_k(x_j, y_j, z_j, t), \quad (4)$$

where V is a neighborhood of point (x_j, y_j, z_j) .

An analogous expression can be used for the plane $z = z_i$. Now, we must render a 3D macroscopic flow. We shall observe that each

node (x_i, y_i, z_i) in Figure 3 belongs to the planes $x = x_i$ and $z = z_i$. So, the 3D density can be finally obtained through a simple mean of the corresponding values in the FHP planes, that means:

$$\rho(x_i, y_i, z_i, t) = \frac{\rho_x(x_i, y_i, z_i, t) + \rho_z(x_i, y_i, z_i, t)}{2}. \quad (5)$$

We also calculate the velocity field through macroscopic samples of the FHP behavior. In order to do that, we use a set of test particles whose initial positions in 3D domain are random. For each macroscopic particle p_{mac} , we take a neighborhood $V(p_{mac}, radius)$, where $radius$ is an radius of influence, and we calculate the sum of the velocities $\vec{c}_i$ (equation (1)) of the microscopic particles inside the neighborhood:

$$\vec{v}(p_{mac}) = \sum_{\vec{r} \in V(p_{mac}, radius)} \vec{c}_i(\vec{r}, t). \quad (6)$$

Once we have calculated the velocity field, given a test particle p , in $\vec{x}(p, t)$ position at instant t , the position of the particle p at next time is:

$$\vec{x}(p, t + \Delta t) = \vec{x}(p, t) + \sigma \vec{v}(p), \quad (7)$$

where σ is a constant for the macroscopic step in time.

4 Surface Flow Simulation Model

The model is composed by two basic elements: (1) A DEM that approximate the terrain; (2) A LGCA for water flow simulation [Barcellos et al. 2008]. The DEM is composed by a regular lattice and an elevation function $\varphi(i, j)$ that gives the elevation of the terrain in the lattice node (i, j) . The LGCA is a particle based model that uses the 2D regular lattice of the DEM. The Figure 5 highlights a (i, j) node of the lattice and its 4 neighbors given by $(i-1, j)$, $(i, j-1)$, $(i+1, j)$ and $(i, j+1)$. These nodes are numbered V_1, V_2, V_3 and V_4 , respectively. We also can compute the terrain slope at point $s(i, j)$. It is defined as the projection of the surface normal over the domain D .

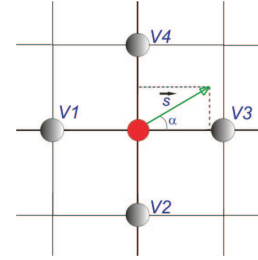


Figure 5: Neighborhood of a lattice node.

For each grid node (i, j) a particle counter is associated, which mimics the water held at terrain point $(i, j, \varphi(i, j))$. The particle system is used in order to update the field of counters of the lattice. The elevation and the slope define the stationary fields of the model. The water particles evolution updates the counter field which will be a non-stationary one.

The particles movement are discrete; that means, each particle moves from one lattice node to another one in the neighborhood, in the time step Δt . Therefore, given a particle at the position (i, j) and its slope $\vec{s}(i, j)$, we can define its direction of movement through the Algorithm 1.

Specifically, the particle moves from a node (i, j) to a nearest neighbor P , if the corresponding lattice direction $P - (i, j)$ is closer the slope direction at (i, j) and the high of the water free surface at (i, j) is greater than in P . This is verified in the 4th line in the Algorithm 1. A four bit string $n_1(i, j), n_2(i, j), \dots, n_4(i, j)$ is related to each node (i, j) of the lattice. We set $n_k(i, j) = 1$ if the node (i, j) has one particle to send to the neighbor k , as we will see in the Algorithm 2, in the 14th line, explained below.

Algorithm 1: findVelocityDirection

```

1:  $P \leftarrow findVelocityDirection(i, j, \vec{s})$ ;
2:  $V \leftarrow \{(k, l) \mid (i, j) - (k, l) \leq 1\}$ ;
3: while  $V \neq \text{NULL}$  do
4:    $P \leftarrow \arg \min \{arc(\vec{s}, (k, l) - (i, j)), (k, l) \in V\}$ ;
5:   Delete  $P$  from  $V$ ;
6:    $P \leftarrow \text{NULL}$ ;
7: end while
8: return  $P$ ;

```

The Algorithm 2 summarizes the whole method. The inputs are the elevation function φ and the precipitation $P(t)$. In the initialization step, we must initialize the field $n_k(i, j)$, the particle counter $counter(i, j)$ and the slope field $s(i, j)$. In the first loop of the simulation, given a particle at the position (i, j) and its slope $\vec{s}(i, j)$, the direction of movement is obtained through Algorithm 1, and, consequently, the four bits string n are updated. The second loop of the simulation step updates the particle counters of each node (i, j) and its neighbors.

Algorithm 2: calculateWaterSurface()

```

1: Input:
2:   Elevation function  $\varphi$ .
3:   Precipitation  $P(t)$ .
4: Initialization:
5:   Define the lattice  $L$ ;
6:    $n_k(i, j) \leftarrow 0$ ,  $counter(i, j) \leftarrow 0$ ,  $(i, j) \in L$ ;
7:   Compute the slope  $\vec{s}(i, j)$ ,  $(i, j) \in L$ ;
8: Simulation:
9: for each  $(i, j) \in L$  do
10:   $P \leftarrow findVelocityDirection(i, j, \vec{s})$ ;
11:   $n_l(i, j) = 0$ ,  $l = 1, 2, 3, 4$ ;
12:  for  $l \in \{1..4\}$  do
13:    if  $P = V_l$  then
14:      set  $n_l(i, j) = 1$ ;
15:    end if
16:  end for
17: end for
18: for each  $(i, j) \in L$  do
19:   $counter(i, j) \leftarrow counter(i, j) - n_1(i, j) - n_2(i, j) -$ 
     $n_3(i, j) - n_4(i, j) + P(t)$ ;
20:   $counter(i-1, j) \leftarrow counter(i-1, j) + n_1(i, j) + P(t)$ ;
21:   $counter(i, j+1) \leftarrow counter(i, j+1) + n_2(i, j) + P(t)$ ;
22:   $counter(i+1, j) \leftarrow counter(i+1, j) + n_3(i, j) + P(t)$ ;
23:   $counter(i, j-1) \leftarrow counter(i, j-1) + n_4(i, j) + P(t)$ ;
24: end for

```

The result is a function

$$f(i, j, t) = \varphi(i, j) + \beta \cdot counter(i, j, t), \quad (8)$$

which gives the high of the free surface flow f , at the point (i, j) in the simulation time t (β is a scale parameter). We shall emphasize that the algorithm does not suffer from numerical stability issues because the floating point fields are stationary and the update of the counter field is based on simple comparisons and integer arithmetic. Besides, volume conservation, an important issue to accurate simulates the flood distribution, is straightforward verified because all operations are conservative with respect to the number of particles. This algorithm can be generalized for DTMs that approximate the terrain surface for triangulations, as demonstrated in [Barcellos et al. 2008; Barcellos et al. 2007].

5 GPU-Based Simulation Model

The algorithm described in section 4 for surface flow simulation in DTMs is based on a cellular automaton approach, with local and simple rules. Therefore it may be very suitable for a GPU implementation. The main idea is to encode the information needed in the simulation as textures into the video memory (see [Barcellos et al. 2008] for more details).

The Figure 6 gives an overview of the whole GPU processing. At each interaction of the main loop a 2D texture, called **Property Texture**, encoding the automaton configuration is generated as follows. To each texture point (i, j) it is associated the index k , such that each color channel represents a different data: the four bit string $n_k(i, j) = 1$; $k = 1, 2, 3, 4$, the terrain elevation $\varphi(i, j)$ and a particle counter, respectively channels R , B and G . Based on the **Property Texture** a **RGBA texture** is generated, named **Incident Directions Texture**, with each channel been a flag that is set to 1 if the corresponding neighbor of a node has one particle to send to its position and set to 0 otherwise.

By using the **Incident Directions Texture** and the particles counter at time interaction t , two new fields are generated at time $t + 1$. One with the updated particle counters and a second field with the elevation of the free surface flow $f(i, j)$ at each (i, j) node of the lattice. These fields will be composed to form the new **Property Texture** and the **Free Surface Texture**, at time $t + 1$, represented on Figure 6. Then, the visualization pipeline starts, producing a new texture to encode the normal field of the free surface, called **Normal Vector Texture** in Figure 6.

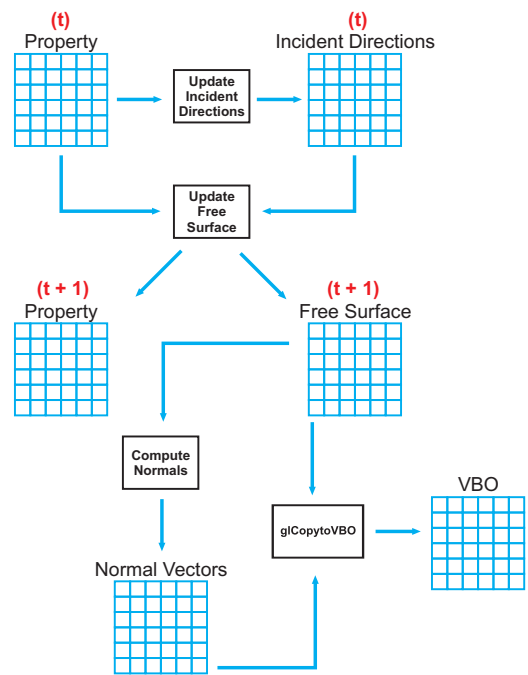


Figure 6: GPU processing for the surface flow animation.

The **Normal Vector Texture** and the **Free Surface Texture** are used as input to the **Vertex Buffer Object (VBO)**, as Figure 6 shows. The **VBO** will be managed as a **Vertex Array** to render the final visualization of the free surface of the fluid. Therefore, the whole computation (simulation and visualization) is performed in the GPU. This scheme allow us to minimize the information flow between the CPU and the GPU and improve the performance of the application. The GPU implementation is summarized by the algorithm 3. The corresponding shader are described in [Barcellos et al. 2008].

Algorithm 3: GPU Simulation()

```

1: Input:
2:   Property Texture (Velocity, Terrain Elevation, Particles Counter);
3: Simulation:
4:   Update Incident Directions;
5:   Update Free Surface;
6:   Compute Normals;
7:   Send Results to Vertex Array;
8: Visualization:
9:   Draw Vertex Array;

```

6 Interaction between Models

Our proposed framework for real time fluid animation is based on two elements: (a) A 3D fluid simulation model based on the FHP and interpolation techniques, that is implemented in CPU; (b) The surface flow simulation in DTMs, that is implemented in GPU. To integrate this two elements we have to implement a communication between them through a suitable boundary condition.

We use the 3D FHP to simulate a rainfall over a random region of the terrain, chosen by the user. After setting the initial configuration and parameters, we can start the evolution of the system. The Figure 7 shows the interaction between the models in each time step.

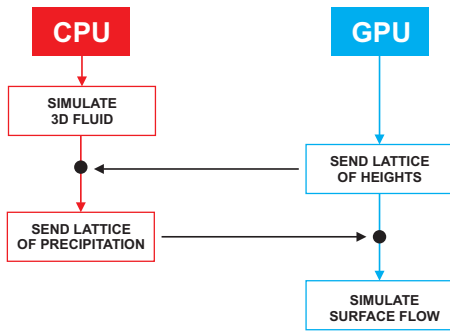


Figure 7: Interaction between models in each time step.

Basically, the rainfall has to evolve to reach the terrain. This is what happens in the **Simulate 3D Fluid** box in CPU (Figure 7). In each time step the microscopic particles of the 3D FHP evolve following the rules explained in section 3. After that, the system has to know if some of these particles have overtaken the limits of the three dimensional fluid (FHP). To perform this task, the GPU implementation has to send to the CPU implementation a lattice with the information about the field computed by the expression (8) at each node (i, j) . With this data, the system can check all the new position of the 3D FHP particles: if the particle has reached the limit it dies in the FHP grid and feed the lattice of the LGCA for water flow simulation. So, the CPU implementation has to send to the GPU implementation the updated particles counter field. The corresponding precipitation is used in the algorithm 2, updating the $P(t)$ variable. Then, the GPU implementation is able to simulate the fluid behavior over the terrain, what is represented in Figure 7 by the **Simulate Surface Flow** box in GPU. The GPU implementation has to update the free surface elevation, given by expression (8).

7 Experimental Results

In this section we describe details of the implementation and some experiments with the proposed framework. In these experiments we show the behavior of the fluid in different digital terrain models and highlight aspects that can be useful for computer graphics and real time applications: simple to initialize the animation, simulation over complex topography and computational efficiency.

The framework was developed in C/C++ language, with GLUI [Rademacher 1999; Stewart 2006] library for the graphic interface. The visualization was implemented in OpenGL [Wright et al. 2007]. To increase the scene realism we applied the environment mapping technic, the point sprite method to render the rainfall and Fresnel effects (based on the physical laws of reflection and refraction) to render the water free surface. The shaders was implemented in OpenGL Shading Language.

The experiments were performed in a Intel Core 2 Duo 2.66 GHz, with 4 GB of RAM and a Video Card NVidia GeForce 8800 GTX, running Windows XP. The pictures included in this paper are snapshots obtained from the framework. The corresponding videos can be found in <http://virtual01.lncc.br/~barcellos/videosSBGames2008.zip>

7.1 Particles Visualization

The microscopic particles of the FHP are initialized in random positions at the top of the bounding box. As the system evolve, new microscopic particles are born to feed the rainfall flow. As explained in section 3, with simple interpolation techniques we can fill the 3D macroscopic fields (density and velocity) over a regular grid. In this way we can estimate the particle density (equation (5)) at each node of the grid, and visualize then using the Point Sprite method. Meanwhile, we can observe the patterns of a regular grid, as shown in Figure 8.(a).

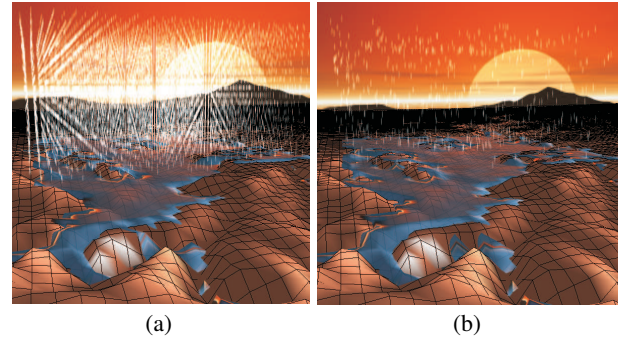


Figure 8: Particles visualization: (a) Microscopic particles; (b) Macroscopic particles.

To avoid this patterns we use a set of macroscopic particles at random positions, inside the bounding box of the rainfall. This macroscopic particles move according to the macroscopic velocity field, following expressions (6) and (7). In this way we have a more realistic visualization of the rainfall, as shown in Figure 8.(b). That is the way in the following examples, we render macroscopic particles to visualize the rainfall.

7.2 Fluid Behaviors

Before initialize the framework the user has to decide which digital terrain model he wants to use in the simulation. Then, the graphical user interface implemented allows to interactively place the rainfall domain over the terrain. Figure 9 shows some examples of digital terrain models and a bounding box that represents the location of the 3D fluid model. With the environment mapping technique we are able to generate different environments, like a sunny or cloudy day.

The Figure 10 shows some snapshots of the fluid motion with the downhill path over the terrain surface and affected regions. As expected, the water flows directly downhill and it takes the easiest downhill path that is available.

The lake formation can be better observed in Figure 11. We can notice, along time evolution, the enlargement of the flooded area, generating a watershed. A watershed is an area of land that drains downhill to a body of water, such as a river. It includes both the waterway and the land that drains to it.

The Figure 12 shows a concave region in the terrain digital model and the the water accumulation along time evolution.

7.3 Computational Efficiency

The examples of section 7.2 where performed with $20 \times 30 \times 30$ (height $\times$ width $\times$ depth) nodes in the 3D fluid simulation. We use a 256×256 lattice resolution for the surface flow simulation. Both initialization as particle generation methods for the particles of the FHP and for the set of macroscopic particles are random. The microscopic particles iterate following the rules explained in section 3. In our implementation the FHP particle are restricted to the bounding boxes pictured in Figure 9. If a microscopic particle goes outside this volume it is discarded by the simulation. The set of macroscopic particles iterate following the FHP velocity field. We use $radius = 5$ in expression (6) and $\sigma = 0.0015$ in expression 7. All the examples were simulated until 3000 frames.

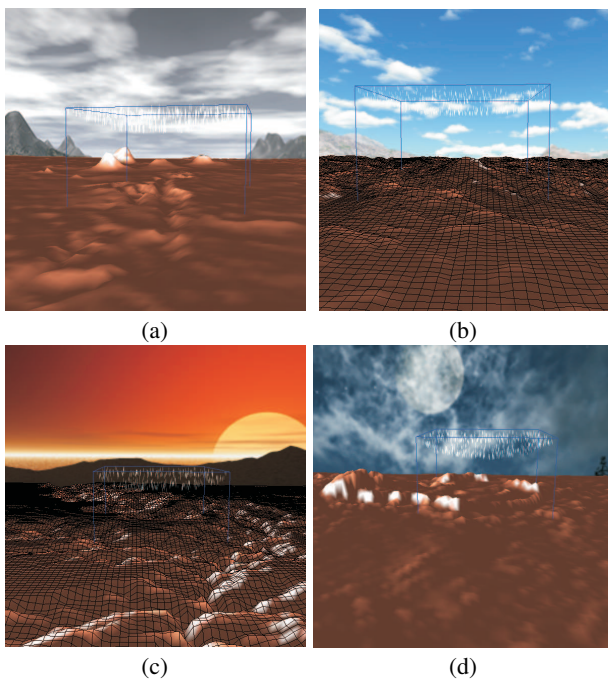


Figure 9: Examples of FHP bounding box over some DTMs with different environments: (a) Cloudy day at Mars terrain; (b) Sunny day at Puget Sound terrain; (c) Canyon terrain with sunrise; (d) Starry night at Venus terrain.

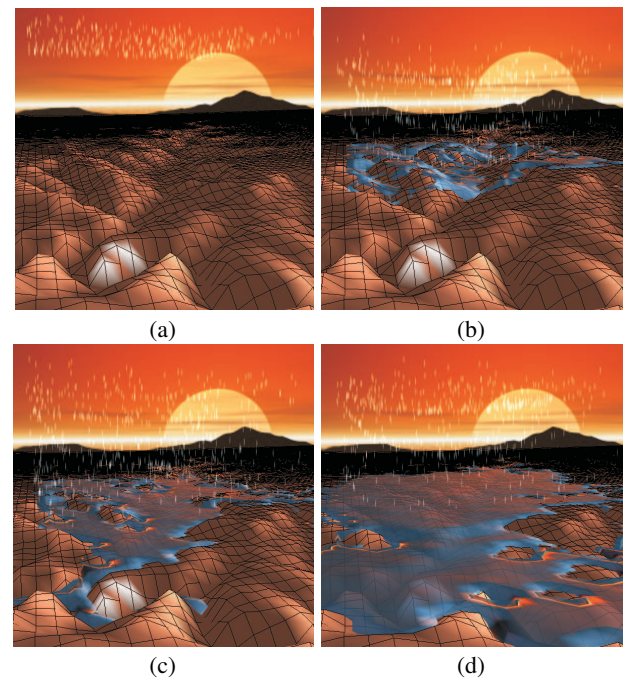


Figure 11: The lake formation: (a) At initial configuration; (b) After 60 steps; (c) After 900 steps; (d) After 3000 steps.

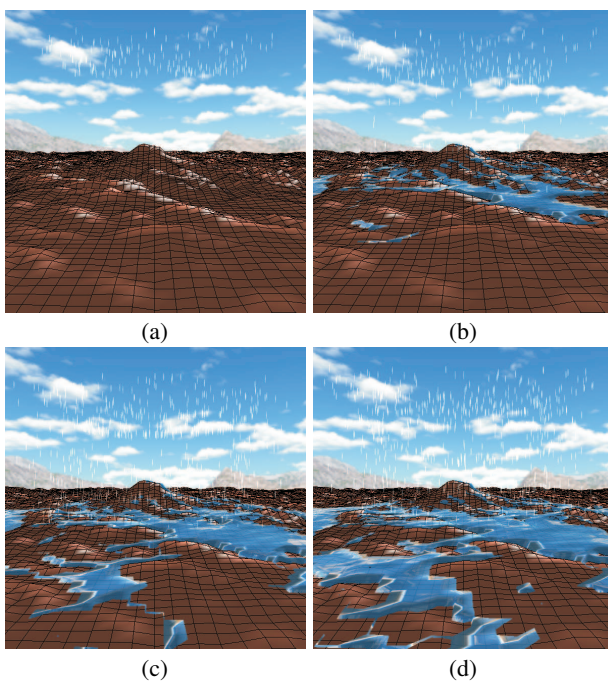


Figure 10: Fluid motion: (a) At initial time steps; (b) After 400 steps; (c) Configuration after 1200 time steps; (d) After 3000 steps.

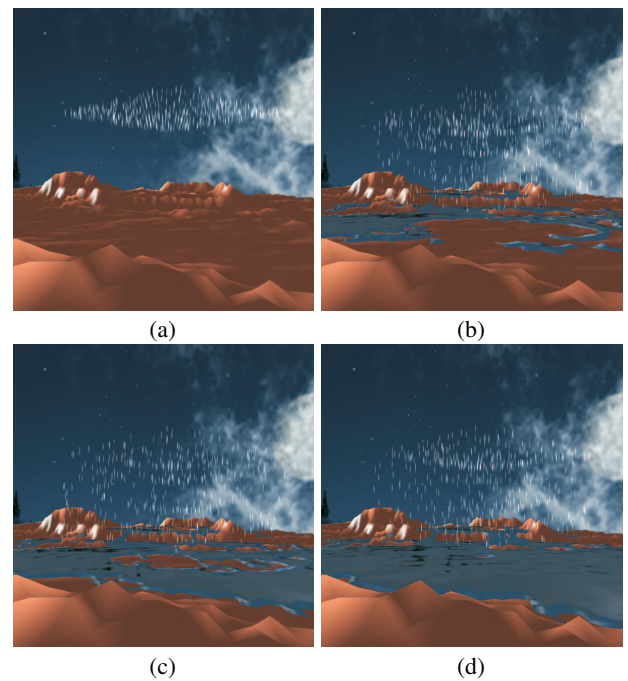


Figure 12: The water accumulation: (a) At initial configuration; (b) After 940 steps; (c) After 1500 steps; (d) After 3000 steps.

The time measurements presented in Table 1 were performed in order to compare the increase of the FPS when using different 3D FHP grid dimensions, as to observe what happens with the number of the FHP particles per frame and the accumulated particles in the water flow over the terrain. We maintain the configuration explained above and the Puget Sound as the terrain model.

Table 1: Table listing the 3D FHP grid dimensions (height $\times$ width $\times$ depth), the average number of FHP particles per frame, the accumulated number of particles in the water flow over the terrain, the average number of Macro particles per frame and finally the frames per second rates(FPS).

FHP Grid	FHP	Water	Macro	FPS
1) $10 \times 10 \times 10$	772	479	61	60
2) $10 \times 20 \times 20$	4333	26033	178	44.78
3) $10 \times 30 \times 30$	10653	64549	383	31.91
4) $10 \times 40 \times 40$	19343	121747	651	22.72
5) $20 \times 10 \times 10$	6541	14958	35	60
6) $20 \times 20 \times 20$	5686	1029	274	37.50
7) $20 \times 30 \times 30$	16464	41086	553	25
8) $20 \times 40 \times 40$	32219	104146	937	16.04
9) $30 \times 10 \times 10$	11620	16343	44	56.60
10) $30 \times 20 \times 20$	36535	54453	165	35.29
11) $30 \times 30 \times 30$	18397	1381	765	19.35
12) $30 \times 40 \times 40$	40038	56094	1230	12.24
13) $40 \times 10 \times 10$	14380	14853	53	53.57
14) $40 \times 20 \times 20$	52295	54265	191	30.61
15) $40 \times 30 \times 30$	104865	118215	454	17.65
16) $40 \times 40 \times 40$	42559	2039	1572	10.27

The 7th line shows the 3D FHP grid configuration used to generate the examples of section 7.2. We can observe that it returns a 25 FPS, which is closer a real time one. Other configurations, like the $30 \times 20 \times 20$ (line 10th), gives a FPS that goes beyond the real time frame rate.

The frame rate is very dependent from the number of FHP nodes and particles. However, once the FHP particles that goes outside the bounding box is simply discarded, it is possible that two similar configurations show different frame rates. That is why the experiments reported in the 2nd and 13th lines have the same number of nodes (4000) but the FPS is larger in the latter one.

We can observe that the experiment reported in the 13th line has a highest bounding box (height = 40) than the one in the 2nd line (height = 10), which ensures a large average number of FHP particles per frame in the first case (14380 against 4333). However, due to be higher, the experiment reported in the 13th line has a larger number of discarded particles (the particles that goes outside the bounding box), which means that it contributes less to the precipitation over the terrain (14853 against 26033).

Once the surface flow simulation is performed in GPU, an important concern is the accuracy of the numerical simulation. The graphics hardware used supports 4 bytes per color channel which fits the requirement on the accuracy of the computation.

Table 2: Table listing the lattice dimensions, the average number of FHP particles per frame, the accumulated number of Water Particles in the surface flow and the frame rate.

Lattice	FHP	Water	FPS
128×128	15884	53911	32.9
256×256	19163	71345	24.8
512×512	19219	77390	13.6

In the Table 2 we report the FPS evolution against the lattice resolution. If we keep parameters for the rainfall simulation and visualization unchanged, we get lower frame rates when increasing the lattice resolution because the computational cost of the surface flow simulation gets larger. Figure 13 pictures the effect of the lattice resolution in the final scene visualization. According to Table 2 we have a real time frame rate in this case. We observe also that the visual quality of the generated scene was preserved quite well.

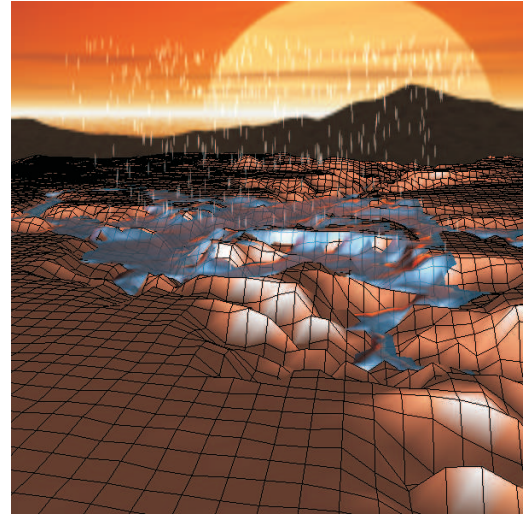


Figure 13: 128×128 digital terrain model.

8 Conclusion and Future Works

In this work, we propose a framework for fluid animation based on Cellular Automata models for computer games. The new animating framework is composed by a 3D fluid animation technique and a GPU surface flow simulation over terrain models. In the experimental results we emphasize the simplicity and power of the proposed models when combined with efficient techniques for rendering.

Future directions in this work are to perform a GPU implementation of the fluid model in order to improve performance and to compare the obtained results with the Lattice Boltzmann model to quantify the level of physical realism obtained.

Acknowledgements

To João Vicente P. Reis Filho, for all the discussions.

References

- BARCELLOS, B., GIRALDI, G. A., SILVA, R. L., APOLINARIO, A. L., AND RODRIGUES, P. S. S. 2007. Surface flow animation in digital terrain models. In *SVR 2007 - IX Symposium on Virtual an Augmented Reality*, 123–132.
- BARCELLOS, B., GIRALDI, G. A., APOLINARIO, A. L., AND RODRIGUES, P. S. S. 2008. Gpu surface flow simulation and multiresolution animation in digital terrain models. Tech. rep., National Laboratory of Scientific Computing.
- BATTY, C., BERTAILS, F., AND BRIDSON, R. 2007. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph.*
- BENZI, R., SUCCI, S., AND VERGASSOLA, M. 1992. The lattice boltzmann equation: Theory and applications. *Physics Reports*.
- BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. In *SIGGRAPH '02*.
- BUICK, J. M., EASSON, W. J., AND GREATER, C. A. 1998. Numerical simulation of internal gravity waves using a lattice gas model. *Int. J. Numer. Meth. fluids*.
- CHOPARD, B., AND DROZ, M. 1998. *Cellular Automata Modeling of Physical Systems*. Cambridge University Press.
- DEUSEN, O., EBERT, D. S., FEDKIW, R., MUSGRAVE, F. K., PRUSINKIEWICZ, P., ROBLE, D., STAM, J., AND TESSENDORF, J. 2004. The elements of nature: Interactive

- and realistic techniques. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*.
- DOOLEN, G. 1990. *Lattice Gas Method for Partial Differential Equations*. Addison-Wesley.
- ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. *ACM Trans. Graph.*
- FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *SIGGRAPH '01*.
- FOSTER, N., AND METAXAS, D. 1997. Modeling the motion of a hot, turbulent gas. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*.
- FRISCH, U., HASSLACHER, B., AND POMEAU, Y. 1986. Lattice-gas automata for the navier-stokes equation. *Phys. Rev.*
- FRISCH, U., D'HUMIÈRES, D., HASSLACHER, B., LALLEMAND, P., POMEAU, Y., AND RIVET, J.-P. 1987. Lattice gas hydrodynamics in two and three dimension. *Complex Systems*, 649–707.
- FRISCH, U., D'HUMIÈRES, D., HASSLACHER, B., LALLEMAND, P., POMEAU, Y., AND RIVET, J.-P. 1987. Lattice gas hydrodynamics in two and three dimension. *Complex Systems*.
2004. In *Game Programming Gems*, K. Pallister, Ed. Charles River Media, Cambridge, MA.
- GENEVAUX, O., HABIBI, A., AND DISCHLER, J.-M. 2003. Simulating fluid-solid interaction. In *Graphics Interface*.
- GIRALDI, G., XAVIER, A., JR, A. A., NETO, A., AND RODRIGUES, P. 2005. Animation of gas-liquid systems through lattice gas cellular automata and smoothed particle hydrodynamics. In *Proceedings of the CNMAC 2005*.
2005. In *GPU Gems 2*, M. Pharr, Ed. Addison Wesley.
- GUENDELMAN, E., SELLE, A., LOSASSO, F., AND FEDKIW, R. 2005. Coupling water and smoke to thin deformable and rigid shells. In *SIGGRAPH '05*.
- HIRSCH, C. 1988. *Numerical Computation of Internal and External Flows: Fundamentals of Numerical Discretization*. John Wiley & Sons.
- IGLESIAS, A. 2004. Computer graphics for water modeling and rendering: a survey. *Future Gener. Comput. Syst.*
- INAMURO, T., OGATA, T., AND OGINO, F. 2004. Numerical simulation of bubble flows by the lattice boltzmann method. *FUTURE GENERATION COMPUTER SYSTEMS* 20, 6, 959–964.
- KRGER, J., AND WESTERMANN, R. 2005. Gpu simulation and rendering of volumetric effects for computer games and virtual environments. In *In Proceedings, Eurographics*, 685–693.
- LIU, G. R., AND LIU, M. B. 2003. *Smoothed Particle Hydrodynamics : a Meshfree Particle Method*. World Scientific.
- MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of ACM SIGGRAPH symposium on Computer animation*.
- MÜLLER, M., KEISER, R., NEALEN, A., PAULY, M., GROSS, M., AND ALEXA, M. 2004. Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*.
- MÜLLER, M., SCHIRM, S., AND TESCHNER, M. 2004. Interactive blood simulation for virtual surgery based on smoothed particle hydrodynamics. *Technol. Health Care* 12, 1, 25–31.
- PREMOZE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A., AND WHITAKER, R. 2003. Particle-based simulation of fluids. *Computer Graphics Forum* 22, 3.
- RADEMACHER, P. 1999. *GLUI: A GLUT-Based User Interface Library*. Último acesso em 25/05/2006, June.
- ROTHMAN, D. H., AND ZALESKI, S. 1994. Lattice-gas models of phase separation: Interface, phase transition and multiphase flows. *Rev. Mod. Phys.*
- SOLENTHALER, B., SCHLAFLI, J., AND PAJAROLA, R. 2007. A unified particle model for fluid-solid interactions. *Comput. Animat. Virtual Worlds*.
- STAM, J. 1999. Stable fluids. In *Siggraph 1999*.
- STAM, J. 2003. Real-time fluid dynamics for games. In *Proceedings of the Game Developer Conference*.
- STAM, J. 2003. Flows on surfaces of arbitrary topology. In *SIGGRAPH '03*.
- STEWART, N. 2006. *GLUI User Interface Library*. Último acesso em 25/05/2006.
- WENZEL, C. 2006. Real-time atmospheric effects in games. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, 113–128.
- WOLFRAM, S. 1996. *Cellular automata and complexity*. Addison-Wesley, <http://www.stephenwolfram.com/publications/articles/ca/86-fluids/index.html>.
- WRIGHT, R. S., LIPCHAK, B., AND HAEMEL, N. 2007. *OpenGL SuperBible (4rd Edition)*. Addison-Wesley Professional.
- XAVIER, A. V., GIRALDI, G. A., RODRIGUES, P. S., JR, A. L. A., AND NETO, A. A. S. 2005. Lattice gas cellular automata for computational fluid animation. In *SIBGRAPI 2005 - Proceedings of the 4th Workshop of Theses and Dissertations*, 1–6.
- YE, Z., MAIDMENT, D., AND MCKINNEY, D. 1996. Map-based surface and subsurface flow simulation models: An object-oriented and gis approach. Tech. rep., Center for Research in Water Resources, University of Texas at Austin.
- ZHAO, Y., QIU, F., FAN, Z., AND KAUFMAN, A. E. 2007. Flow simulation with locally-refined lbm. In *SI3D*, ACM, 181–188.