A simple surface tracking method for physically-based 3D water simulations

G. Amador[1] and A. Gomes[1]

[1]Instituto de Telecomunicações,
Departamento de Informática, Universidade da Beira Interior,
6201-001 Covilhã, Portugal

ABSTRACT

Water simulation, and more generically fluid simulation, is an important research topic in computer graphics. In 3D Eulerian Navier-Stokes-based water simulations, surface tracking and rendering are two delicate problems. The existing solutions to these problems (i.e., implicit surfaces-based approaches, height-fields, ray-tracing), are either to computationally intensive for real-time scenarios, or present bulge water surfaces (i.e., blobby water surfaces). In this paper, we propose a novel tracking algorithm for rendering water surfaces. Instead of tracking the flow of water using either level sets or height-fields, each cell of an 3D grid density value is directly *measured in order to determine if it is either water, air, or water-air contact surface. The information in each cell is later used for the water surface splat rendering, using OpenGL vertex buffer objects.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Boundary representations I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

There are two major families of fluid simulation methods, those that are based on principles of physics and those that are not [Igl04, TY09]. The latter are known as procedural methods, and are not dealt in this paper. The former are usually based on the Navier-Stokes (NS) equations. In the family of physics-based methods we find three types of methods, namely: Lagrangian methods, Eulerian methods, and Lattice Boltzman Methods (LBM). All these water simulation methods require techniques for tracking, reconstruction, and rendering of water surfaces. It is true that several techniques addressing these problems already exist [EF02, EMF02, Fle07, MÖ9], but these require computation that penalize the overall frame rate of simulations. In some cases this problem is so severe that real-time simulations are impossible.

In Lagrangian methods, we use either the Moving Particle Semi-Implicit (MPS) [PTB*03, Yam09] or the Smoothed Particle Hydrodynamics (SPH) [MCG03, HCW*10, MLTOA10] methods to track and reconstruct the surface of water. Both methods are used to simulate water effects such as, for example, splashes, spray, or puddles. These methods employ implicitly-defined representations to reconstruct the water surface.

Procedural methods and some of the Eulerian methods utilize height-fields to render water mantles (e.g., ocean surface near shore, rivers, and pounds) [Mik04, BD06]. Eulerian methods and LBM, i.e., methods where space is discretized over a 1D, 2D, or 3D grid of cells, make usage of level sets [Sus03, TR04, KIS07], marker level sets [FF01, BM07], particle level sets [LSSF06], or even hybrids approaches [LGF04, CCE05, KLL*07] for tracking and reconstruction of water surfaces.

Implicit surface-based approaches (i.e., level sets-based, SPH, and MPS), for water surface tracking and reconstruction, are either to much computationally time expensive or present blobby water surfaces (i.e., non-smooth surfaces). The height-fields alternative is limited to the representation of water surface mantles. In order to reduce the computational time required in the physical-based simulation, thus allowing faster, in some cases real-time, surface tracking and reconstruction, in Eulerian simulations usually simplified versions of the NS equations are employed, namely: Shallow Water Equations (SWE) [Mik04, dlAMC10], Euler equations [Fed02, ELD08], or 2D NS [Sta02, Sta03a, Sta03b, ETK*07]. For real-time (i.e., games or virtual reality) water simulations, the most used techniques are SPH or height-fields combined with either procedural methods, SWE, 2D

NS equations, or Euler equations. Also hybrid approaches, i.e., approaches that resort to two or more methods for water simulation, are often used [LGF04, CCE05, BM07, MSJT08, N08, Gje09].

The main contribution of this paper is a simple, and efficient (i.e., without penalty on the simulation frame rate), modified semi-Lagrangian advection [Sta99] algorithm, to allow 3D coupled water-air physically-based simulations surface tracking. Therefore, if the simulation runs at interactive rates (i.e., real-time) the method presented may be used. The presented method, non-existent as far as the authors are aware, tracks altogether water, air, and water-air surface cells, without using any implicit surface-based approach. Also, the presented method, which resembles the one in [FF01], is designed in such way, that it may be extended to store useful information for an water surface rendering algorithm, i.e., for each grid cell which of its 26 neighbour cells are either water, air, surface, or boundary cells. Thus, in the presented method, comparatively to implicit surface-based techniques, less-computations are required. However, the presented method requires more memory read/write accesses.

The main contribution of this paper is a simple, and efficient (i.e., without penalty on the simulation frame rate), modified semi-Lagrangian advection [Sta99] algorithm, to allow 3D coupled water-air physically-based simulations surface tracking. Therefore, if the simulation runs at interactive rates (i.e., real-time) the method here presented may be used to render water surfaces. Our novel method is able to track altogether water, air, and water-air surface cells, without using any implicit surface-based approach. Also, the presented method, which resembles the one in [FF01], is designed in such way that it may be extended to keep track useful information for an water surface rendering algorithm, i.e., each grid cell carries information of its 26 neighbour cells so that it is possible to know whether they are either surface cells or not. Thus, in our method, comparatively to implicit surface-based techniques, less-computations are required. However, our method requires more memory read/write accesses.

This paper is organized as follows. In Section 2 the physically-based fluid simulator based on stable fluids is briefly explained. Section 3 addresses the implementation of the surface tracking algorithm. Section 4 deals with the analysis of the performance of the water simulation. Finally Section 5 draws relevant conclusions and points out new directions for future work.

2. Fluid Simulator

As previously mentioned, our surface tracking algorithm is a modified stable fluids method in the sense that it extends the stable fluids method to include water surface tracking. Our algorithm was applied to the 3D version port, addressed in [Ash05], of the original 2D stable fluids [Sta03b]. Our method is a Eulerian method in such a manner that it divides a boxed domain into a 3D grid, which is then mapped to an 1D array, as illustrated in Fig. 1). We have modified the advection step of the stable fluids in order to allow tracking and rendering water surfaces (see Section 3). But, before proceeding any further, let us briefly describe the stable fluids, in particular the specific step of advection.

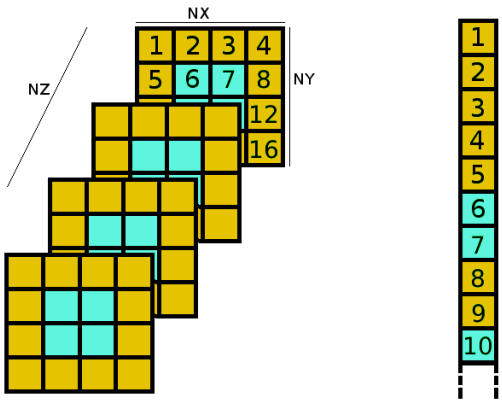

Figure 1: *3D grid (left) represented by a 1Darray (right).*

The motion of fluids (e.g., water or air) can be described by a set of partial differential equations, known as the Navier-Stokes equations (Eqs. 1, 2, and 3).

$$\frac{\partial \vec{u}}{\partial t} = -(\vec{u} \cdot \nabla)\vec{u} + \nu \nabla^2 \vec{u} + \vec{f} \tag{1}$$

$$\frac{\partial \rho}{\partial t} = -(\vec{u} \cdot \nabla)\rho + k\nabla^2 \rho + S \tag{2}$$

$$\nabla \vec{u} = 0 \tag{3}$$

where $\vec{u}$ represents the velocity field, ν is a scalar describing the kinematic viscosity of the fluid, $\vec{f}$ are the external forces added to the velocity field (e.g., gravity), ρ is the density of the field, k is a scalar that describes the rate at which density diffuses, S is the external source added to the density field, $\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right)$ is the gradient operator, and $\nabla^2 = \nabla \cdot \nabla = \left(\frac{\partial^2}{\partial x^2}, \frac{\partial^2}{\partial y^2}, \frac{\partial^2}{\partial z^2}\right)$ is the Laplacian operator.

Eqs. 1 and 2 describe the evolution of velocity and density over time. Eq. 3 states that the velocity field, where the fluid flows, must be mass/energy conserving. To solve these three equations, first we have to solve each of the terms in the velocity field equation (Eq. 1) and mass/energy conservation equation (Eq. 3). To solve each of the terms three steps are required, namely: add external accelerations $\left(\vec{f}\right)$, diffusion $\left(\frac{\partial \vec{u}}{\partial t} = \nu \nabla^2 \vec{u}\right)$, and move $\left(-(\vec{u} \cdot \nabla)\vec{u} \text{ and } \nabla \vec{u} = 0\right)$. Afterwards, each of the terms of the density (i.e., fluid quantities) equation (Eq. 2) are solved. These terms are solved in three steps, namely: add external sources (S), diffusion $\left(\frac{\partial \rho}{\partial t} = k\nabla^2 \rho\right)$, and advection $(-(\vec{u} \cdot \nabla)\rho)$. Notice that, if the kinematic viscosity ν in Eq. 1 is discarded the fluid is classified as inviscid, and the obtained equations are referred as the Euler equations.

To implement our surface tracking algorithm, we only needed to modify the advection step of stable fluids (term $(-(\vec{u} \cdot \nabla)\rho)$ in Eq. 2). The remaining steps of stable fluids, aside from being ported from 2D to 3D, required no other changes. Therefore, we refer the reader to [Sta03b, Ash05, CCE05] for further explanations on the discretization of Eqs. 1, 2, and 3.

Let us now focus on the modified advection step. When

a fluid flows, solid objects, densities, the fluid itself (i.e., self-advection), and other quantities (i.e., other fluids) are advected, convected or transported, as a result of the fluid's velocity when it's moving. To better understand advection let us consider that each cell of an 3D grid represents a fluid particle, and that each particle travels along a velocity field. The stable fluids advection is performed using an implicit formulation, to overcome the instability for large time steps of explicit methods. Stam's approach, referred as semi-Lagrangian advection, as illustrated in Fig. 2, consists in tracing each particle movement along an uniform velocity field (blue arrows) back in time (green arrow), to its former position (black circle), and then swap its value with the one of the starting grid cell.

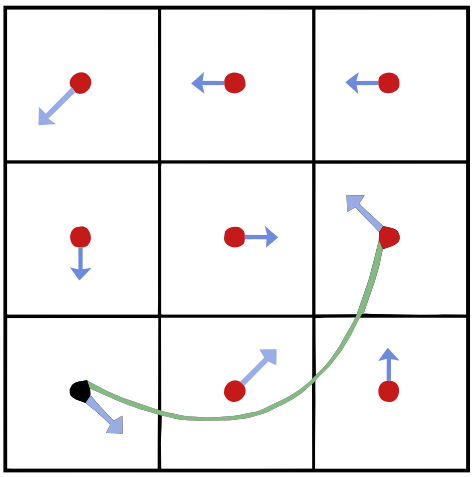

Figure 2: *Semi-Lagrangian Advection.*

3. Surface Tracking and Rendering

In computer graphics, we are primarily interested on rendering the surface of water. Underwater effects are dealt using underwater caustics rendering techniques, which are not addressed in this paper.

To render the water surface in 3D Eulerian-based simulations, two steps are required. First the surface of water must be tracked. Water surfaces can either be in contact with other fluids, or solids. Other previously mentioned methods, trace cells in Eulerian approaches and particles in Lagrangian approaches, that are air or water. Secondly the rendering/reconstruction of the water surface is addressed. Our algorithm only addresses the first step.

Our approach is to change the density semi-Lagrangian advection step in order to determine, if a cell is either a air cell, a water cell, water surface cell, or bounding cell (i.e., moving or static obstacles). In order to further address water surface rendering, the previous information for each cell is stored in the first 6 bits, of an 8 digits hexadecimal number (i.e., 32 bits number), as illustrated in Fig. 3. The remaining 26 bits of this hexadecimal number, currently not being updated by our algorithm, are designed to store which of the cell's 26 neighbours are either water surface cells or not. With the information of the first 6 bits splat rendering of the water surface is currently performed. However, if stored, information about each cell's 26 neighbours could be further used in a water surface rendering/reconstruction algorithm. Therefore, we already used hexadecimal flags instead of strings or characters (e.g., WATER, WATER-AIR, etc.) to assign each cell as either water, surface, air or bound cell. The hexadecimal representation chosen was the less memory consuming solution found, to store information about each cell and its respective 26 neighbours.

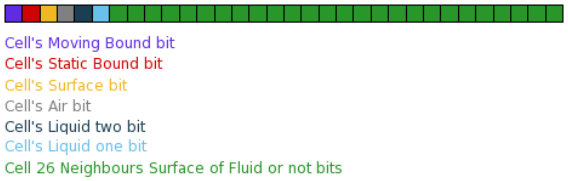

Figure 3: *Cell configuration.*

The simulation grid data is stored in a data structure (`GRID`), where information on the total number of grid cells (`GRID_cells`), the total number of water surface cells (`S_cells`), the total number of water (surface or not) grid cells (`W_cells`), the grid cells size per axis (`sizeX,sizeY,sizeZ`), the previous and current velocity and density values for each grid cell (`vx,vy,vz,vx0,vy0,vz0,d,d0`), and which grid cells are marked as either water, air, water surface, or internal and external boundaries (`w_a_s_bds`):

```
typedef struct {
int GRID_cells;
int S_cells;
int W_cells;
float sizeX,sizeY,sizeZ;
float *vx,*vy,*vz,*vx0,*vy0,*vz0,*d,*d0;
unsigned long *w_a_s_bds;} GRID;
```

The array (`w_a_s_bds`) could be made of 1 byte elements for each grid cell. However, other information (i.e., which of a cell's 26 neighbours are also water surface cells), for each grid cell, will be stored, in future work. This information will be used for two purposes, namely, a surface reconstruction and rendering algorithm to replace current splat rendering, and to reduce the number cell's to process in the move step for velocity and the diffusion step for velocity and density.

As observed in Fig. 3, for each grid cell, four bytes of data are used to store information. The bits, of each of these bytes. that are set to `'1'` are the properties of the corresponding cell (e.g., if the cell is either moving or static obstacle). The cells that are marked as static boundaries (i.e., with the hexadecimal value `'0x40000000'` in `w_a_s_bds`) are read from a file. The water, air, water surface, and moving boundary data are stored in `w_a_s_bds`. This information is updated at each full update of the physical component update of the simulation. Thus, no calculation is required to determine an implicit function that defines the water surface as in the level set approach.

To access a specific cells data of the grid an macro is used (`IX`). The simulation grid number of elements in the x,y,z directions are also a set of macros (`NX,NY,NZ`). The remaining macros are only to reduce code redundancy in the `advect` and `draw_density` functions.

```
#define NX 32
#define NY 32
#define NZ 32
#define IX(x,y,z) (x+(NX+2)*(y+z*(NY+2)))
#define IF1(i,j,k,hex) (((g->w_a_s_bds[IX(i↩
    ,j,k)])&(hex))!=(hex))
#define IF2(i,j,k,hex) (((g->w_a_s_bds[IX(i↩
    ,j,k)])&(hex))==(hex))
#define FOR_EACH_CELL1\
for(k=1;k<=NZ;k++){\
for(j=1;j<=NY;j++){\
for(i=1;i<=NX;i++){
#define FOR_EACH_CELL2\
for(k=1;k<=NZ;k++){\
z=((k-0.5f)*g->sizeZ)/NZ;\
for(j=1;j<=NY;j++){\
y=((j-0.5f)*g->sizeY)/NY;\
for(i=1;i<=NX;i++){\
x=((i-0.5f)*g->sizeX)/NX;
#define END_FOR_EACH_CELL }}}
```

After the update of the velocity field, at the density advection step our tracking algorithm is applied. The proposed tracking algorithm is divided in two parts. In the first part, for each advected cell, if the cell is not either a moving (i.e., with the hexadecimal value `'0x80000000'` in `w_a_s_bds`) or static (i.e., with the hexadecimal value `'0x40000000'` in `w_a_s_bds`) boundary, and if the cells density value is higher than 0.2, the cell is marked as water (i.e., with the hexadecimal value `'0x08000000'` in `w_a_s_bds`) otherwise the cell is marked as air (i.e., with the hexadecimal value `'0x10000000'` in `w_a_s_bds`).

After performing advection for all grid cells, before enforcing the boundary conditions, the second part of the algorithm is initiated. Before starting the second part, the total number of water (surface or not) and water surface cells counters (`W_cells` and `S_cells`) are set to zero. The second part consists in identifying which water cells are the water surface cells. To achieve this, for each cell that is marked as water (i.e., with the hexadecimal value `'0x08000000'` in `w_a_s_bds`) the total of water cells (surface or not) counter (i.e., `W_cells`) increments. Afterwards, if at least one of that cell's 26^{th} neighbours is air the cell is marked as water surface cell (i.e., with the hexadecimal value `'0x20000000'` in `w_a_s_bds`), and we increment one to the the total of surface water cells counter (i.e., `W_cells`). The modified advection source code follows:

```
void advect(GRID *g,int b,float *d,float *↩
    d0,float *vx,float *vy,float *vz,float↩
     dt) {
int i,j,k,i0,j0,k0,i1,j1,k1;
float xf,yf,zf,s0,t0,u0,s1,t1,u1,dtx,dty,↩
    dtz;

dtx=dt*(NX); dty=dt*(NY); dtz=dt*(NZ);

FOR_EACH_CELL1
xf=(float)i-(dtx*vx[IX(i,j,k)]);
yf=(float)j-(dty*vy[IX(i,j,k)]);
zf=(float)k-(dtz*vz[IX(i,j,k)]);

if(xf<0.5f) xf=0.5f;
if(xf>(NX+0.5f)) xf=NX+0.5f;
i0=(int)xf; i1=i0+1;

if(yf<0.5f) yf=0.5f;
if(yf>(NY+0.5f)) yf=NY+0.5f;
j0=(int)yf; j1=j0+1;

if(zf<0.5f) zf=0.5f;
if(zf>(NZ+0.5f)) zf=NZ+0.5f;
k0=(int)zf; k1=k0+1;

s1=xf-i0; s0=1-s1;
t1=yf-j0; t0=1-t1;
u1=zf-k0; u0=1-u1;

d[IX(i,j,k)]=s0*(t0*(u0*d0[IX(i0,j0,k0)]
+u1*d0[IX(i0,j0,k1)])
+(t1*(u0*d0[IX(i0,j1,k0)]
+u1*d0[IX(i0,j1,k1)])))
+s1*(t0*(u0*d0[IX(i1,j0,k0)]
+u1*d0[IX(i1,j0,k1)])
+(t1*(u0*d0[IX(i1,j1,k0)]
+u1*d0[IX(i1,j1,k1)])));

if(b==0) {
if((IF1(i,j,k,0x80000000))&&(IF1(i,j,k,0↩
    x40000000))) {
if(d[IX(i,j,k)]>0.2)
(g->w_a_s_bds[_IX(i,j,k)])=0x08000000;
else
(g->w_a_s_bds[_IX(i,j,k)])=0x10000000;
}}
END_FOR_EACH_CELL

g->S_cells=0;
g->W_cells=0;

FOR_EACH_CELL1
if(b==0) {
if(IF2(g,i,j,k,0x08000000)) {
g->W_cells++;

if((IF2(g,i-1,j,k,0x10000000))||
(IF2(g,i+1,j,k,0x10000000))||
(IF2(g,i,j-1,k,0x10000000))||
(IF2(g,i,j+1,k,0x10000000))||
(IF2(g,i,j,k-1,0x10000000))||
(IF2(g,i,j,k+1,0x10000000))||
(IF2(g,i-1,j-1,k,0x10000000))||
(IF2(g,i-1,j+1,k,0x10000000))||
(IF2(g,i-1,j,k-1,0x10000000))||
(IF2(g,i-1,j,k+1,0x10000000))||
(IF2(g,i+1,j-1,k,0x10000000))||
(IF2(g,i+1,j+1,k,0x10000000))||
(IF2(g,i+1,j,k-1,0x10000000))||
(IF2(g,i+1,j,k+1,0x10000000))||
(IF2(g,i-1,j-1,k-1,0x10000000))||
(IF2(g,i-1,j+1,k-1,0x10000000))||
(IF2(g,i-1,j-1,k+1,0x10000000))||
(IF2(g,i-1,j+1,k+1,0x10000000))||
(IF2(g,i+1,j-1,k-1,0x10000000))||
(IF2(g,i+1,j+1,k-1,0x10000000))||
(IF2(g,i+1,j-1,k-1,0x10000000))||
(IF2(g,i+1,j+1,k+1,0x10000000))||
(IF2(g,i,j-1,k-1,0x10000000))||
(IF2(g,i,j+1,k-1,0x10000000))||
(IF2(g,i,j-1,k+1,0x10000000))||
(IF2(g,i,j+1,k+1,0x10000000))) {
g->w_a_s_bds[IX(i,j,k)]=0x20000000;
g->S_cells++;
}}}
END_FOR_EACH_CELL
```

```
set_bnd(g,b,d);
}
```

To render the water surface, exclusively cells marked as water surface are drawn as discs. The source code used to splat render the water surface, whose cells information in the 1D array `w_a_s_bds`), follows:

```
void draw(GRID *g,float alpha_blend) {
float x,y,z;
int i,j,k,l=0;

GLsizei DrawPrimitiveRequiredVertex=1;
GLsizei VertexCount=g->S_cells*↩
    DrawPrimitiveRequiredVertex;
GLsizeiptr VertexSize=VertexCount*3*sizeof(↩
    GLfloat);
GLfloat *VertexData=(GLfloat*) malloc(↩
    VertexSize);

FOR_EACH_CELL2
if(IF2(i,j-1,k+1,0x20000000))) {
VertexData[l+0]=x;
VertexData[l+1]=y;
VertexData[l+2]=z;
l+=3;
}
END_FOR_EACH_CELL

glPushMatrix();
glColor4f(0.6,0.7,0.8,alpha_blend);
glPointSize(22);
glBindBuffer(GL_ARRAY_BUFFER,1);
glBufferData(GL_ARRAY_BUFFER,VertexSize,↩
    VertexData,GL_STREAM_DRAW);
glVertexPointer(3,GL_FLOAT,0,0);
glEnableClientState(GL_VERTEX_ARRAY);
glDrawArrays(GL_POINTS,0,VertexCount);
glDisableClientState(GL_VERTEX_ARRAY);
glPopMatrix();

free(VertexData);
}
```

4. Experimental Results Analysis

The presented algorithm was tested on a Intel(R) Core(TM) i7 CPU 920@2.67GHz, with 8.0Gbytes of DDR3 RAM, an NVIDIA GTX 295 graphics card, running an Windows 7 64 bits Operative System. The surface tracking algorithm was implemented as non-parallel single-core CPU version. To test the proposed algorithm, a small graphics simulation scenario was implemented. This scenario consists in water being poured into a container (white box) from a pipe (green cylinder), as shown in Figs. 4 and 5. At every time step each grid cell marked as water surface was rendered as an disc (white disks). Figs. 4 and 5 illustrate the simulation running before using our surface tracking and rendering algorithms (i.e., rendering all grid cells where density value was higher than 0.2) and after (i.e., rendering only the water surface cells).

Both simulation scenarios 'Before' and 'After' using our surface tracking algorithm, illustrated in Figs. 4 and 5, ran on a 32^3 grid of cells, i.e., 'Grid Size'. For each of the simulation scenarios, the frames per second (FPS), and the total

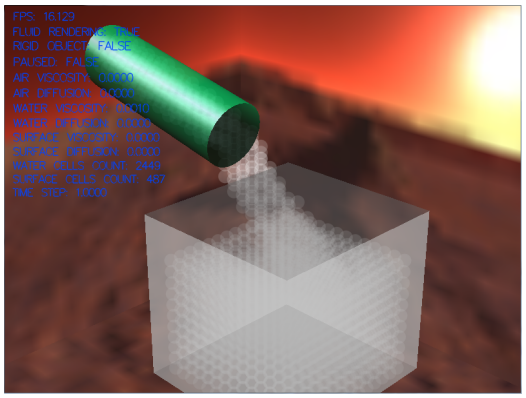

Figure 4: *Water (surface or not) Rendering.*

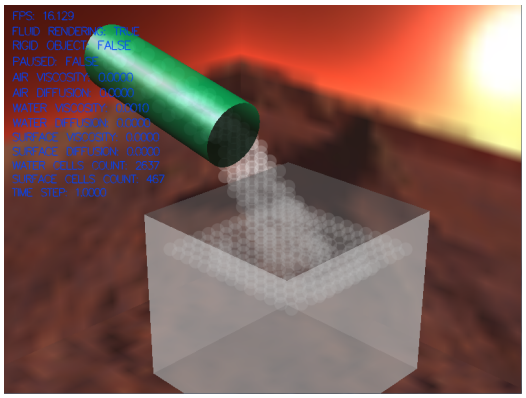

Figure 5: *Surface Water Rendering.*

of grid cells that are water cells 'TWC' and water surface cells 'TWSC', data was gathered, as shown in Table 1.

	Grid Size	# FPS	# TWC	# TWSC
Before	32^3	16	2449	487
After		16	2637	467

Table 1: *Comparison between all water cells rendering and surface rendering scenarios.*

Looking at Figs. 4 and 5, and Table 1, two major conclusions can be drawn. First, the algorithm did not penalized the total frame rate of the simulation. Second, the total number of water elements (surface or not) and water surface elements to render is a negligible portion of the total number of elements of the simulation grid (respectively less than 10% and less than 2% of the total number of elements of an 32^3 grid). Altogether, both conclusions favour the possibility that, if our algorithm was extended to store for each cell's 26 neighbours if they are surface cells or not, a better surface water rendering can be made using reconstruction techniques in real-time.

5. Conclusions and Future Work

In this paper a novel water surface tracking technique was presented. This technique does not the penalize the performance of the fluid simulation. The proposed technique, implemented on an 3D Eulerian NS-based implicit fluid solver,

can also be extended to explicit solvers. It diverges from previous approaches, applied to Eulerian simulations (i.e., implicit-based methods). The presented technique simultaneously tracks directly the water, air, and water surface elements, instead of computing the evolution of a implicit function, that describes the water surface shape. Therefore, since the presented technique is fully integrated in an 3D Eulerian solver, no extra computation to evolve the implicit function along the velocity field are required.

Currently, the presented technique does not include the tracking for each grid cell of which of its 26 neighbours are either water surface cells or not. Nevertheless, all the presented code was devised taking this future extension in consideration. Therefore, hexadecimal values are used instead of simple string expressions for as less as possible memory usage. This information would allow also a fast water surface rendering with some modifications in the water rendering algorithm.

There are several further work directions. First, with the tracked air, water, and water surface cells, and a few modifications to the linear solver, used both in the diffusion and projection steps, to implement air-water coupling (i.e., multiphase fluids interaction) to the simulation. Second, to implement the presented algorithm in the CUDA-based version presented in [AG10]. Third, to improve the tracking algorithm and to replace the current surface splat rendering with an water surface reconstruction and rendering algorithm. All these modifications, are intended to be implemented and tested for real-time purposes, in either non-parallel CPU- or CUDA-based versions. Fourth, to add reflection to the reconstructed water surface. Finally, to optimize and port the non-parallel CPU-based simulator to MPI, OpenMP, and OpenCL versions.

Acknowledgements

This work has been partially supported by the Fundação para a Ciência e a Tecnologia (FCT) and the Instituto de Telecomunicações (IT), through the MOGGY (A Browser-Based Massive Multiplayer Online Game Engine Architecture) project, whose reference is PTDC/EIA/70830/2006.

References

[AG10] AMADOR G., GOMES A.: A CUDA-Based Implementation of Stable Fluids in 3D with Internal and Moving Boundaries. In *Proceedings of the 2010 International Conference on Computational Science and Its Applications (ICCSA '10)* (Washington, DC, USA, Mar. 2010), IEEE Computer Society Press, pp. 118–128. Stable fluids+CUDA. 6

[Ash05] ASH M.: *Simulation and Visualization of a 3D Fluid*. Master's thesis, Université d'Orléans, France, Orléans, France, Sept. 2005. 2

[BD06] BABOUD L., DÉCORET X.: Realistic Water Volumes in Real-Time. In *Eurographics Workshop on Natural Phenomena* (2006). 1

[BM07] BRIDSON R., MÜLLER M. F.: Fluid simulation. In *ACM SIGGRAPH 2007 Course Notes (SIGGRAPH '07)* (New York, NY, USA, Aug. 2007), ACM Press, pp. 1–81. 1, 2

[CCE05] CLINE D., CARDON D., EGBERT P. K.: Fluid Flow for the Rest of Us: Tutorial of the Marker and Cell Method in Computer Graphics. Unpublished, 2005. 1, 2

[dlAMC10] DE LA ASUNCIÓN M., MANTAS J., CASTRO M.: Programming CUDA-Based GPUs to Simulate Two-Layer Shallow Water Flows. In *Proceedings of the 16th International Euro-Par Conference*, vol. 6272. Springer-Verlag, Heidelberg, Berlin, Germany, Sept. 2010, pp. 353–364. 1

[EF02] ENRIGHT D., FEDKIW R.: Robust Treatment of Interfaces for Fluid Flows and Computer Graphics. In *Proceedings of the 9th International Conference on Hyperbolic Problems Theory, Numerics, Applications* (July 2002), pp. 153–164. 1

[ELD08] ELSEN E., LEGRESLEY P., DARVE E.: Large calculation of the flow over a hypersonic vehicle using a GPU. *J. Comput. Phys. 227*, 24 (Dec. 2008), 10148–10161. 1

[EMF02] ENRIGHT D., MARSCHNER S., FEDKIW R.: Animation and rendering of complex water surfaces. *ACM Trans. Graph. 21*, 3 (July 2002), 736–744. 1

[ETK*07] ELCOTT S., TONG Y., KANSO E., SCHRÖDER P., DESBRUN M.: Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph. 26*, 1 (Jan. 2007). 1

[Fed02] FEDKIW R.: Coupling an Eulerian fluid calculation to a Lagrangian solid calculation with the ghost fluid method. *J. Comput. Phys. 175*, 1 (Jan. 2002), 200–224. 1

[FF01] FOSTER N., FEDKIW R.: Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)* (New York, NY, USA, Aug. 2001), ACM Press, pp. 23–30. 1, 2

[Fle07] FLECK B.: Real-Time Rendering of Water in Computer Graphics. Unpublished, 2007. 1

[Gje09] GJERMUNDSEN A.: *LBM vs. SOR Solvers on GPUs for Real-Time Snow Simulations*. Master's thesis, Norwegian University of Science and Technology, Trondheim, Sœr-Trœndelag, Norway, Dec. 2009. 2

[HCW*10] HE J., CHEN X., WANG Z., CAO C., YAN H., PENG Q.: Real-time adaptive fluid simulation with complex boundaries. *Vis. Comput. 26* (Apr. 2010), 243–252. 1

[Igl04] IGLESIAS A.: Computer graphics for water modeling and rendering: a survey. *Future Gener. Comput. Syst. 20*, 8 (Nov. 2004), 1355–1374. 1

[KIS07] KEENAN C., IGNACIO L., SARAH T.: Real-Time Simulation and Rendering of 3D Fluids. In *GPU Gems 3*. Addison-Wesley Professional Co., 2007, ch. 30, pp. 633–675. 1

[KLL*07] KIM B., LIU Y., LLAMAS I., JIAO X., ROSSIGNAC J.: Simulation of bubbles in foam with the volume control method. *ACM Trans. Graph. 26*, 3 (July 2007). 1

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23* (Aug. 2004), 457–462. 1, 2

[LSSF06] LOSASSO F., SHINAR T., SELLE A., FEDKIW R.: Multiple interacting liquids. *ACM Trans. Graph. 25*, 3 (July 2006), 812–819. 1

[MÖ9] MÜLLER M.: Fast and robust tracking of fluid surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '09)* (New York, NY, USA, Aug. 2009), ACM Press, pp. 237–245. 1

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '03)* (Aire-la-Ville, Switzerland, July 2003), Eurographics Association Press, pp. 154–159. 1

[Mik04] MIKLÓS B.: Real time fluid simulation using height fields. Unpublished, July 2004. 1

[MLTOA10] MARUZEWSKI P., LE TOUZÉ D., OGER G., AVELLAN F.: SPH High-Performance Computing simulations of rigid solids impacting the free-surface of water. *J. of Hydraulic Research 48*, Extra Issue (2010) (2010), 126–134. 1

[MSJT08] MÜLLER M. F., STAM J., JAMES D., THÜREY N.: Real time physics. In *ACM SIGGRAPH 2008 Course Notes (SIGGRAPH '08)* (New York, NY, USA, Aug. 2008), ACM Press, pp. 1–90. 2

[NÖ8] NÄTTERLUND M.: *Water Surface Rendering*. Master's thesis, Umeå University, Umeå, Sweden, Mar. 2008. 2

[PTB*03] PREMOZE S., TASDIZEN T., BIGLER J., LEFOHN A. E., WHITAKER R. T.: Particle-Based Simulation of Fluids. *Comput. Graph. Forum 22*, 3 (Nov. 2003), 401–410. 1

[Sta99] STAM J.: Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)* (New York, NY, USA, Aug. 1999), ACM Press/Addison-Wesley Publishing Co., pp. 121–128. 2

[Sta02] STAM J.: A simple fluid solver based on the FFT. *J. Graph. Tools 6* (Sept. 2002), 43–52. 1

[Sta03a] STAM J.: Flows on surfaces of arbitrary topology. In *ACM SIGGRAPH 2003 Papers (SIGGRAPH '03)* (New York, NY, USA, July 2003), ACM Press, pp. 724–731. 1

[Sta03b] STAM J.: Real-Time Fluid Dynamics for Games. In *Proceedings of the 2003 Game Developer Conference (GDC '03)* (Mar. 2003). 1, 2

[Sus03] SUSSMAN M.: A second order coupled level set and volume-of-fluid method for computing growth and collapse of vapor bubbles. *J. Comput. Phys. 187*, 1 (May 2003), 110–136. 1

[TR04] THÜREY N., RÜDE U.: Free Surface Lattice-Boltzmann fluid simulations with and without level sets. In *Proceedings of the 2004 International Workshop on Vision, Modeling, and Visualization (VMV '04)* (Amsterdam, The Netherlands, Nov. 2004), vol. 2, IOS Press, pp. 199–207. 1

[TY09] TAN J., YANG X.: Physically-based fluid animation: A survey. *Science in China Series F: Information Sciences 52*, 5 (May 2009), 723–740. 1

[Yam09] YAMAMOTO K.: Real time two-way coupling of fluids to deformable bodies using particle method on GPU. In *ACM SIGGRAPH ASIA 2009 Posters (SIGGRAPH Asia '09)* (New York, NY, USA, Dec. 2009), ACM Press. 1