Efficient Animation of Water Flow on Irregular Terrains

Marcelo M. Maes*
Iwate University, Japan

Tadahiro Fujimoto†
Iwate University, Japan

Norishige Chiba‡
Iwate University, Japan

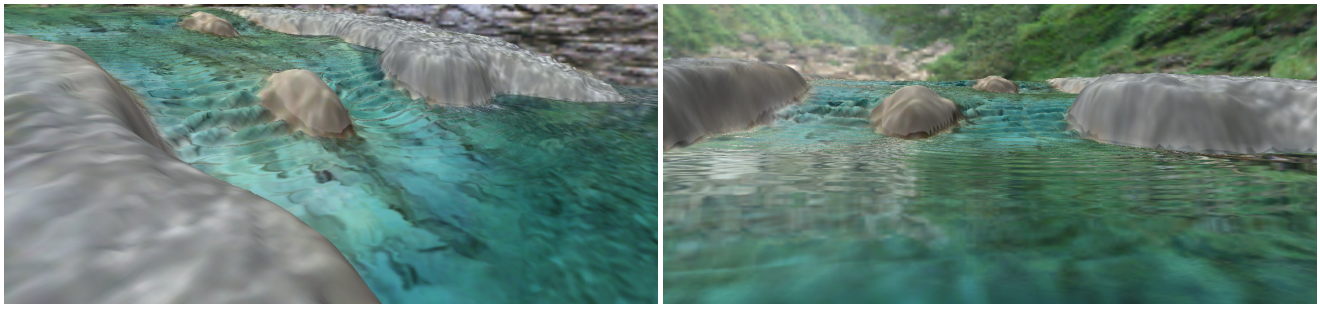


Figure 1. Water flowing on irregular terrain.

Abstract

We present an optimization of the water column-based height-field approach of water simulation by reducing memory footprint and promoting parallel implementation. The simulation still provides three-dimensional fluid animation suitable for water flowing on irregular terrains, intended for interactive applications. Our approach avoids the creation and storage of redundant virtual pipes between columns of water, and removes output dependency for the parallel implementation. We show a GPU implementation of the proposed method that runs at near interactive frame rates with rich lighting effects on the water surface, making it efficient for water animation on natural terrains for Computer Graphics.

Keywords: Natural phenomena, physically based animation, water simulation, height field.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.6.8 [Simulation and Modeling]: Types of Simulation—Animation.

1 Introduction

Water representation and animation have been thoroughly investigated in Computer Graphics due to the complexity of the phenomenon and its visualization. Although recent research focuses on efficient methods to solve the computational expensive water simulation, these methods still require minutes of calculation time for every frame. Interactive applications such as landscape design, virtual reality, and games, which often need three-dimensional water animation at interactive rates, either lack realistic solutions or they have to rely on a two-dimensional plane-based simplification of the water surface.

Due to the complexity of the water behavior, there is no single method that can capture all the subtle effects of water [Iglesias 2004]. Therefore several methods must be combined to produce realistic animations. Preferably, these methods should be based on physics to behave as its physical counterpart. However, Computer Graphics applications don't need the same degree of accuracy as engineering applications, usually sacrificing accuracy for efficiency.

Water flowing on terrains generates several natural phenomena, including rivers, waterfalls, puddles, and lakes. This flow is mainly dominated by gravity and the water is near vertical equilibrium against the ground [Irving et al. 2006]. Since terrains are highly irregular, the water does not lie homogeneously over the terrain. This requires an efficient simulation method with good spatial handling, but without loss of visual details. It is also desirable the visualization to be reasonably simple, making the method suitable for Computer Graphics animation and interactive applications.

We present an optimization of the water column-based height-field approach, previously proposed by [O'Brien and Hodgins 1995; Mould and Yang 1997; Holmberg and Wunsche 2004]. The general idea of these methods is to calculate the hydrostatic pressure in columns of water and the flow due to pressure difference through virtual pipes between adjacent columns. The water columns have variable height and lie directly on the terrain, therefore the flow calculations are spatially performed only where necessary. The method is composed of three interacting systems: a water volume model, a particle model for splashes and bubbles, and an external object interaction model. We show in this work an optimization of the water volume model. This model has several advantages that our approach benefits as well:

- Hydrostatic physics calculation is has low computational cost;
- The model implicitly generates water surface phenomena, such as the propagation of waves;
- All variables are physically based, allowing other physical systems to interact with the water volume model;

*e-mail: marcelo@cg.cis.iwate-u.ac.jp
†e-mail: fujimoto@cis.iwate-u.ac.jp
‡e-mail: nchiba@cis.iwate-u.ac.jp

http://doi.acm.org/10.1145/1174429.1174447

- The three-dimensional simulation has squared computational cost, proportional to the resolution of the two-dimensional grid;
- The top of all columns are known resulting in a straightforward water surface geometry extraction as a height field;
- Low computational cost of optical effects on the water surface inherited from other two-dimensional methods.

There are some limitations as a general solution for fluid simulation:

- The model suffers from vertical isotropy due the column representation;
- Breaking waves and free parts, such as splashes, foam, and bubbles can not be directly represented, requiring an additional particle system;
- Calculation time step must be small otherwise the system becomes unstable and oscillates, which vexes most time-forward integration methods.

Our contributions to the optimization of the water volume model are:

- Low memory footprint by reducing the number of redundant virtual pipes between columns of water, without affecting the results of the physical simulation;
- Parallel promotion of the algorithm by removing output dependency on the shared data;
- Implementation of both the simulation and rendering processes on commodity graphics hardware, thus reducing data transfer for every frame;
- Data structure packing in two-dimensional textures for graphics hardware storage;
- A single height field to represent both terrain and water surface, reducing the geometry rendered per frame;
- Accurate rendering of refraction, light transmittance and attenuation, taking into account the water depth.

In Section 2 we describe related work in fluid simulation for Computer Graphics. In Section 3, we show in detail the proposed model, and in Section 4, the parallel implementation. In Section 5, we present the results by showing several examples. In Section 6 we discuss the advantages, drawbacks and future directions, and we conclude this work in Section 7.

2 Previous Work

To solve the Navier-Stokes equations (NSE) for fluid dynamics, computational models require a lot of computer resources in terms of memory storage and calculation time [Iglesias 2004]. Numerical solutions of the NSE [Anderson 1995] can be categorized in Eulerian (grid-based) and Lagrangian (particle-based) approaches. The first subdivides the space in a regular grid and observes the fluid that passes through it. The second tracks disjoint elements of fluid through time.

One of the first attempts to carry out a full three-dimensional NSE-based simulation in Computer Graphics was the work of [Foster and Metaxas 1996]. They subdivided the three-dimensional space in a regular grid, and solved the Navier-Stokes equations by discretizing the pressure and velocities respectively at the grid's center and faces. They used marker particles to track the fluid surface, and alternatively a height field for liquids.

The most important contribution for stability is the work of [Stam 1999]. The method is made unconditionally stable by applying a semi-Lagrangian method for the advection term of the NSE. A two-dimensional implementation on the GPU was presented by [Harris 2004; Wu et al. 2004] and a three-dimensional by [Liu et al. 2004]. Although these simulations run in real-time, they do not address the problem of simulating fluids with free boundaries, such as water.

The free boundary issue is addressed with a hybrid particle and level set method by [Foster and Fedkiw 2001; Enright et al. 2002; Losasso et al. 2004; Irving et al. 2006]. An implicit function evolves together with the fluid simulation to track the isocontour representing the interface of the liquid. Particles are used around the interface in the coarse grid of the simulation to accurately adjust the surface of the liquid.

Eulerian approaches are not spatially efficient in simulating water flow on terrains. Since terrains may be highly irregular, the grid structure may waste storage space that never contains liquid; see Figure 2 (a).

Losasso et al. [2004] proposed the use of adaptive meshes to alleviate the resolution problem of grid-based methods. They add finer resolution where visual details are necessary. They apply an unrestricted octree structure to increase resolution, and present a new method of discretizing pressure and velocity. Their method reduces the simulation time for fluid simulation with fine detail, without increasing accuracy error.

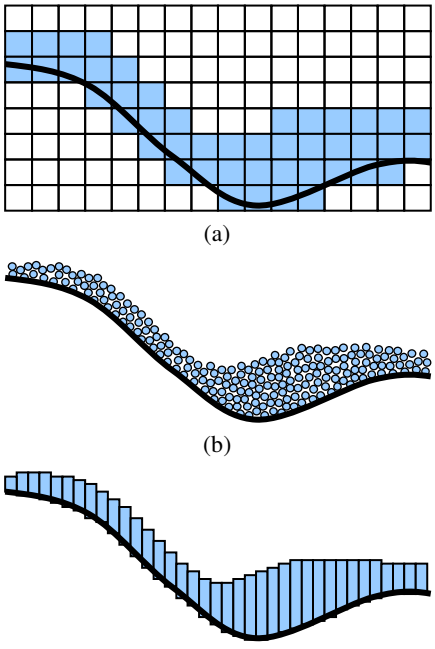


Figure 2. Water flow simulation on terrain (black curve) using different methods: (a) regular grid subdivision stores cells that may be always empty throughout the simulation; (b) particles increase surface details, as well as calculation time; (c) columns of water with variable height has a good trade-off between number of stored cells and surface sampling.

Recently, Irving et al. [2006] proposed a hybrid method of two-dimensional grid composed of tall cells with linear pressure

profile, and a three-dimensional grid near the interface of the fluid. They use a NSE-based solver over both structures by interpolating tall cells values accordingly. They apply the particle/level set method to track the surface of the fluid only in the three-dimensional region. They state this combination has performance gains for flows heavily dominated by gravity, like in shallow water regime. Like other NSE-based solvers, the calculation time is still in the order of minutes per frame.

Particle-based methods represent water throughout the terrain only where needed. Even having a better spatial distribution, these methods usually require smaller time steps to avoid particles bursting away due to attraction and repulsion forces.

Chiba et al. [1995] proposed a quasi-physical method in which particle interactions occur within a voxel space to reduce interactions with distant particles and to perform collisions against obstacles. To reconstruct the water surface, they use an implicit function influenced by the particles. They point out that the number of particles must be high to avoid surface artifacts.

Müller et al. [2001] used Smoothed Particles Hydrodynamics (SPH) to simulate fluids by interpolating physical quantities, such as viscosity and pressure, defined at discrete particles. They use point splatting and marching cubes to render the surface of liquids. They state that tracking and rendering the fluid surface for interactive applications remain a challenge.

Kipfer and Westermann [2006] presented a GPU accelerated particle simulation using the SPH method. They use three sorted linear lists to lookup for particle collisions and a height field over the particles to represent the surface of the water. Although this representation of the surface does not require a dense particle set, it is not volume conserving. The surface details, such as waves, depend directly on the height field resolution, which was apparently small in their examples to keep interactive frame rates.

Premoze et al. [2003] used the Moving-Particle Semi-Implicit (MPS) method to simulate fluids with a level set method to reconstruct the surface. They ran a low-resolution simulation for instant feedback, and then increased the number of particles for the final simulation. Since the MPS method is fully Lagrangian, the fluid particles should be present only where they are needed. However, even a simple polygonal scene must be converted into the particle representation.

Lagrangian approaches usually require a considerable amount of particles to represent the details of the fluid surface, thus increasing storage space and computation time. Additional particles do not contribute only to the surface representation, they also increase the overall number of particles in the simulation; see Figure 2 (b). The surface reconstruction is also complex because of continuous topology change.

To alleviate the complexity of a three-dimensional simulation of water flow on terrains, some works [Neyret and Praizelin 2001; Thon and Ghazanfarpour 2001; Thon and Ghazanfarpour 2002; Rochet 2005] focus only on what is seen in brooks and rivers, i.e., waves and ripples on the water surface near the vicinity of obstacles and banks. The water surface is assumed to be two-dimensional and discretized in a regular grid to run the fluid simulation. Based on the resulting velocity field, ripples and shock waves are extracted; then bump maps are placed and animated on the surface.

Although these methods realistically include phenomena not present in low-resolution three-dimensional simulations, they cannot represent water flowing on irregular terrains and other three-dimensional effects such as splashes and falls.

3 Physical Simulation

Kass and Miller [1990] first proposed to perform water simulation with the assumptions of the water surface being a height field and the horizontal velocity constant through a vertical column of water. Their model uses a simplified subset of the fluid dynamics in two-dimensions. However they do not model the interaction of external objects and free parts such as splashes.

Our physical model is based on the work introduced by O'Brien and Hodgins [1995]. The model is composed of a volume of water which is divided into vertical columns in a rectilinear grid. Each of these columns is connected to its neighbors by virtual pipes. The flow in the pipes is derived from the physical laws of hydrostatic pressure. The model also supports external forces on the surface that are applied as external pressure. Spray particles are created when the upward velocity of a portion of the surface exceeds a certain threshold.

Mould and Yang [1997] extended this model by running the simulation on an arbitrary height field and by reducing the vertical isotropy through the division of each column into multiple cells; see Figure 3. They also extended the particle model by including bubbles rising inside the water. Later, Holmberg and Wunsche [2004] applied this model to simulate the natural movement of rivers, rapids and waterfalls.

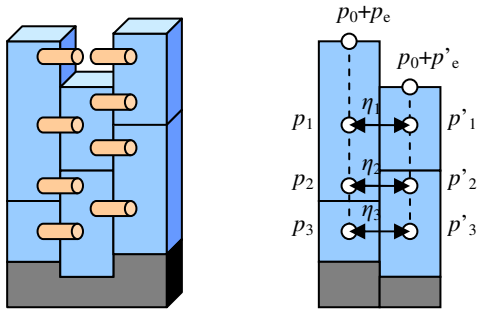


Figure 3. Columns of water with two cells each. (a) Virtual pipes are created between overlapping cells of adjacent columns and the air above the adjacent column. (b) Flow occurs due pressure difference between adjacent columns.

This model has the same advantage of Lagrangian models: since each column lies directly on the terrain, the calculation is spatially performed only where needed; see Figure 2 (c). The height of the columns is variable, and the surface sampling is directly related to the discretization of the rectilinear grid over the height field. Therefore, the water surface can also be represented as a height field over the terrain.

In the next sub-sections we show how to optimize the core of the water volume simulation, followed by its parallel implementation on the GPU.

3.1 Water Volume Model

Here we review the model used in the simulation and the related equations. All vertical columns start with a pre-defined height that can be input by the user, and which varies over time during the simulation. Source and sink columns retain their height to allow in- and out-flows to the system. Virtual pipes are created horizontally between adjacent columns where their cells overlap; see Figure 3 (a). No pipe is created vertically between stacked cells in the same column. Their height varies due to the flow through pipes between neighboring columns.

The flow in these virtual pipes is determined by the physics of hydrostatics. The pressure at one point of the column is given by

$$p = h\rho g + p_0 + p_e \tag{1}$$

where h is the height of water above the calculated point; ρ is the density of the fluid; g is the gravity acceleration; p_0 is the atmospheric pressure; and p_e is the pressure due to external forces.

The flow velocity due to the pressure difference between two points in adjacent cells is given by

$$\eta = f\eta_0 + \Delta t \frac{(p_{head} - p_{tail})}{\rho l} \tag{2}$$

where f is a non-physical frictional coefficient, as suggested in [Mould and Yang 1997] to produce a gradual loss of energy; η_0 is the flow velocity in the previous time step; Δt is the simulation time step; and l is the length of the pipe. Given the flow in the pipe, the volume of water that should be moved through it is

$$V = \Delta t \eta c \tag{3}$$

where c is the cross sectional area of the pipe, i.e. the amount of overlap between the cells. The volume transferred is translated into height changes between the cells. Since mass must be conserved, all pipes that are removing fluid from a cell are scaled back if the volume of that cell becomes negative. When the height of a cell reaches a threshold, the cell is considered dry and does not transfer fluid out to its neighbors.

Since the flow velocity depends on the previous time step, it must be stored in memory for each virtual pipe. As the height of the columns changes throughout the simulation, virtual pipes must be created and deleted as the overlap between adjacent cells changes.

Here we note that the pressure difference between any two submerged points is the same for two adjacent columns; for example $(p_1 - p'_1) = (p_2 - p'_2) = (p_3 - p'_3)$ in Figure 3 (b). The resulting flow in each pipe, Equation (2), will be the same. The volume of water transferred in each pipe differs and depends on the amount of overlap between the adjacent cells. Therefore, to reduce memory storage, we calculate and store the flow of just one pair of those points. Consequently, to calculate the transferred volume of water, we must check if two cells overlap or not for every simulation step. This process does not affect the overall performance since the same process must also be performed in the original algorithm to check whether a pipe must be created or deleted. Thus we reduce the maximum memory requirements per adjacent columns from 2× the number of stacked cells, see Figure 3 (a), to only 2 (the pipe between adjacent columns and the pipe connected to the air) independent of the number of stacked cells.

To model water that breaks free from the water volume, such as splashes and waterfalls, [Holmberg and Wunsche 2004] calculates the volume of water that flows though a weir. In their work, this model is only used when the height of a wave crest becomes unstable, or when the wave height is 0.78 of the water depth. The flow rate through a weir is given by

$$flowrate = \frac{2}{3} b h^{\frac{3}{2}} \sqrt{2g} \tag{4}$$

where b is the width of the column; and h the height of the unobstructed water. The volume of water transferred is

$$V = flowrate \times \Delta t \tag{5}$$

The assumption of flow through a weir is a good approximation since the flow direction is discretized to one of the neighbors, and the flow will occur only in the unobstructed directions.

We note that Equation 4 does not depend on the flow rate from the previous step, and we adopt this model for all the flow between a column of water and the adjacent air above a lower column. Besides reducing the maximum number of stored virtual pipes between adjacent columns to 1, we have a single model for unobstructed water flow when coupled with a particle system. The simulation results show no change in the behavior of the water surface, such as the wave propagation phenomenon.

3.2 Parallel Implementation

Our goal is to bring water flow over terrains at interactive frame rates to Computer Graphics applications. One way of improving the speed of the simulation is to run it in parallel in dedicated processor or distributed architecture. Recently commodity graphics hardware has become inexpensive, programmable, and has been used as a general purpose processing unit [GPGPU]. The processor is capable of running vertex and fragment programs in parallel on multiple dedicated processors.

To avoid communication between parallel processes and random access in the output shared memory storage, which are both not available in programmable graphics hardware; we have to gather all water inflow to the water column being processed and subtract the outflow from itself. To maintain consistency of calculation, we have implemented a single function that calculates the outflow of water in a single column to all neighboring columns. That way it is possible to scale down the outflow in case the volume becomes negative, thus conserving mass in the system.

Following the common procedure for general-purpose computation on GPUs [GPGPU], we store the data structures in two two-dimensional textures, one for the column height and the other for the flow velocity, as shown in Figure 4. Fragment programs are then used to update the stored values using one-to-one pixel-to-texel mapping.

The input terrain is given by a height-map and the height is stored in a floating-point texture. The grid for the columns of water is

created with the same resolution as the terrain height-map. To reduce the access to texture memory, we pack the terrain and the water columns in a single RGBA texture, where the red component has the terrain height, and the other three components can store up to three cells of a single column; see Figure 4 (a).

The flow velocity between two adjacent columns is the same for any pair of points at the same height, regardless of how many cells a column has. Hence we only need to store one flow velocity value per one pair of adjacent columns, rather than allocating and maintaining all virtual pipes between the fluid cells. See Figure 4 (b) for the texture arrangement of pipes and flow direction between 8-neighboring columns. The flow rate through a weir is not stored since it does not depend on values calculated in previous time steps.

With the texture arrangement explained above, we minimize the memory storage necessary for the simulation. The volume of water transferred between cells is computed in a second rendering pass, based on the flow velocity calculated in a first pass.

Because the textures' format and size are the same for both the height and flow values, we swap them with a third texture that serves as a frame buffer. Thus we avoid copying the results to different memory places. The pseudo-code below shows the initialization of the GPU using the OpenGL extensions: Framebuffer Object, Float Texture, and Shader Objects.

Generate and bind framebuffer object

Generate and bind three RGBA two-dimensional floating-point textures, with filtering to nearest, and wrapping to clamp.

Associate each texture with one of the framebuffer object's color attachments.

Associate each color attachment with *flow_velocity*, *height*, and *buffer* aliases.

Draw into buffer with alias *height*.

Load fragment shader to scale values of the height map to the physical heights for the simulation.

Render a quad to write the terrain height values and initial water cells heights.

The next pseudo-code shows a simulation step with the same OpenGL extensions and nomenclature as above.

Draw into buffer with alias *buffer*.

Bind texture with alias *flow_velocity* to read previous time step values.

Use fragment shader to calculate the pressure (Equation 1) and the flow velocity (Equation 2) between adjacent columns.

Render a quad to update the flow velocity values.

Swap *buffer* and *flow_velocity* aliases.

Draw into buffer with alias *buffer*.

Bind texture with aliases *flow_velocity* and *height*.

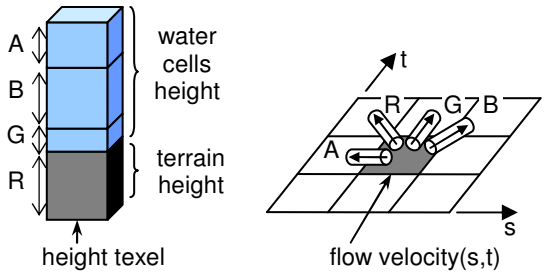
Use fragment shader to check in- and out-flows between overlapping cells by accessing the flow velocity texture, and to calculate the flow rate through a weir (Equation 4).

Render a quad to update height values obtained from the transferred volume of water (Equations 3 and 5).

Swap *buffer* and *height* aliases.

After the simulation step, the texture name associated with the color attachment with alias *height* has the terrain and water column heights needed for rendering.

In our implementation we calculate the outflow of the cell and its neighbors on the fly instead of storing the value in an additional lookup texture.



(a) (b)

Figure 4. Stored textures: (a) height of terrain packed with height of fluid cells of one column; (b) flow through pipes between adjacent columns and the flow direction convention (arrows).

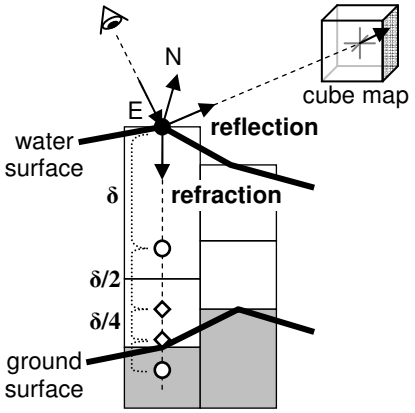


Figure 5. Lighting on the water surface: reflection ray is mapped to an environment cube map; refraction ray intersects the terrain through a linear search with fixed increments δ (hollow circles), followed by a binary search on the last δ (diamonds).

3.3 Particle System

We implement a simple particle system to interact with the water volume model. The particles are used to represent free parts of fluid, such as splashes. One of the most useful definitions of breaking waves [Schlicke 2001] is that breaking occurs when the wave slope exceeds a critical value. Instead of fixing the critical value, we let the user specify the slope threshold to control the particle creation.

We use one extra texture to write the initial velocity and the volume of the free part. This texture must be transferred from the GPU to the CPU to allocate new particles. Two other textures keep the position and the velocity of every generated particle. The resolution of these two textures will limit the number of particles running in the system.

We do not consider inter-particles interaction; they are only

influenced by gravity. The velocity and positions are updated by running fragment shaders on the GPU. When a particle collides with the main body of water, it generates pressure on the surface, derived from a friction force and a buoyant force [Mould and Yang 1997]. This external pressure and the volume of the particle are written in texture memory, and used by the water volume model simulation. Finally, the volume of collided particle is absorbed back in the main water volume.

Our implementation checks if new particles must be created at every simulation step. A more efficient way is to accumulate the volume of free parts and generate them after an amount of frames.

4 Rendering

We render the terrain and the water surface as height fields. Since the terrain height and the water cells height are packed together in a single texture, we can render both surfaces with just one height field and interpolate from one material to the other with a fragment shader in the GPU. This also reduces the number of texture access, consequently reducing the rendering speed.

We use a vertex shader program to displace the height of a grid mesh, and to calculate the normals based on the heights available from the texture. Since all data necessary for this process is already available on the GPU, there is no additional data transfer to and from the GPU for the calculations.

A fragment shader can be used to perform per-pixel calculations for lighting, reflection and refraction of the water surface. The reflection and refraction are calculated based on the eye direction (E) and the surface normal (N); see Figure 5. The reflected ray is mapped to a cube map, assuming that the environment is far away from the surface.

Instead of mapping the refraction to a cube map or assuming that the underlying ground is flat at a certain distance from the surface, we accurately compute the intersection of the refracted ray and the terrain ground through a linear search with fixed increments, followed by a binary search [Policarpo et al. 2005]. Refer to Figure 5 for a schematic diagram of the process.

Although the refraction could be calculated per-pixel, it would require a depth search and float-point texture interpolation per-pixel. In order to maintain interactive frame rates, we perform the refraction calculation in the vertex shader. The length of the refracted ray (from the vertex to the ground) and the texture coordinates (at the intersection with the ground) are interpolated on the GPU through varying variables, which are then available in the fragment shader.

The attenuation of light in transparent volumes does not only decrease the color intensity, but also deepens the color saturation and changes the hue [Sun et al. 1999]. The internal transmittance for liquid solutions is given by the Beer's law

$$T_{\text{int}\,ernal}(\lambda_i) = 10^{-a(\lambda_i)cl} \tag{6}$$

where λ_i is the wavelength for sample i; $a(\lambda_i)$ is the absorption spectrum of the material; c is the solution concentration; and l is the length of the light path. We sample the absorption spectrum of water for the RGB wavelengths. This approach may introduce significant errors due to sub-sampling of the spectrum [Sun et al. 1999]; however it is an acceptable approximation in our case.

The Fresnel term defines the ratio of reflection and refraction of non-polarized light from a dielectric material. We use the Fresnel approximation proposed in [Loviscach 2003], given by

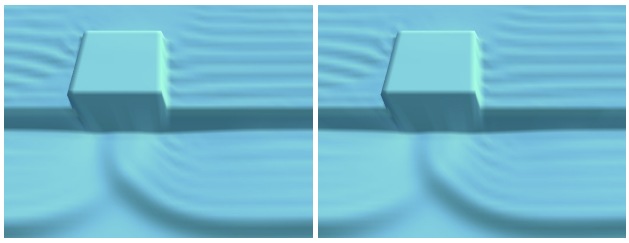
$$Fresnel = clamp(1.0 - 2.5\mathbf{N} \cdot -\mathbf{E}) \tag{7}$$

where *clamp* restricts the values to [0, 1]; **N** is the surface normal; and **E** is the eye direction.

We render the particles as shaded translucent spheres and blend them with the current rendered frame. At this time, no sorting is done when rendering the particles.

5 Results

All the experiments shown here ran on an Intel Pentium 4 at 3.4GHz processor and 1GB of memory, and an NVIDIA GeForce 6600GT graphics card with 128MB of memory. OpenGL and OpenGL Shading Language were used for all graphics operations. The time step of the simulation was set to 0.005s, which would require 200 frames per second animation for a real-time simulation. The viewport resolution was set to 640x480 pixels.



(a) (b)

Figure 6. Appearance comparison with different number of cells per column: (a) one cell per column; (b) three cells per column.

We first ran a simulation with different column subdivisions of one, two and three cells per column. The results are shown in Figure 6. There is no visual difference on the surface of water, but the performance decreased as the number of cells increased. The performance drop for two and three cells per columns was respectively around 60% and 80%. The number of cells per column must be carefully chosen since it has a significant impact in the simulation performance. More cells per column must be used when the application requires more samples of the velocity, derived from height changes, in the vertical direction to interact with objects inside the fluid.

We tested whether the column subdivision still reduced the vertical velocity isotropy or not. We ran a simulation with two cells per column and calculated the root mean square error between the vertical velocity of the bottom cell and the interpolated vertical velocity of the water surface at the same height as the bottom cell. The graph in Figure 7 shows the average vertical velocity at the water surface, the calculated error, and the percentage the error represents relative to the average velocity at the surface. We can see from Figure 7 that the error, after 200 frames of simulation, is steady around 20%. This error is

large if the application requires the interaction with the internal velocities in the fluid. In this case, the column subdivision becomes necessary to reduce the vertical isotropy. For applications requiring only the visual representation of the water surface, the number of cells per column becomes irrelevant.

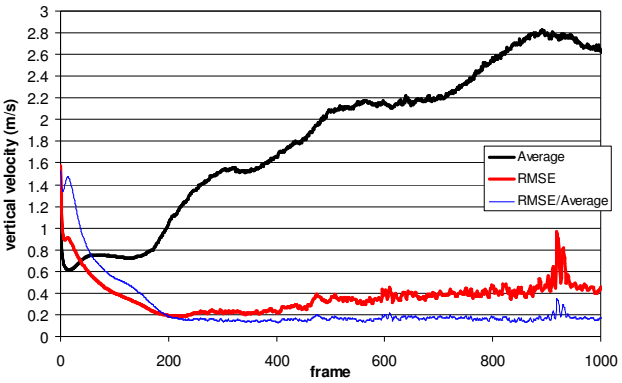


Figure 7. Root Mean Square Error (RMSE) between the vertical velocity at the bottom cell and the interpolated vertical surface velocity at the same height throughout 1000 frames of simulation.

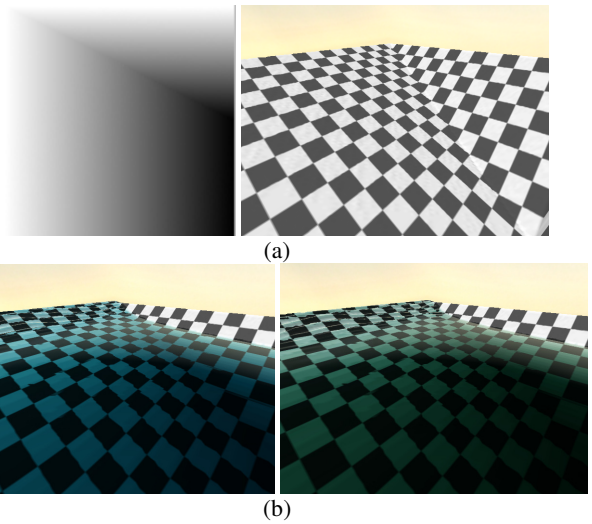
In the next experiment, we verified the speed increase by running the simulation on the GPU. The proposed model allows any number of cells of water in a single column, though we implemented only one cell for the GPU so far. The simulations were rendered using the fixed functionality of the graphics pipeline for comparison purposes. Table 1 shows the results for different terrains. We also ran the simulation for the same terrains rendered with full lighting effects. Table 1 shows the performance of both simulation and rendering time combined. Figure 1 and 8 show selected frames from these animations.

Terrain Map	Resolution (pixels)	FPS CPU	FPS GPU	Speed-up	FPS GPU + FX
Puddle	64 x 64	8.75	147	17x	127
Fractal	128 x 128	2.34	33.2	14x	31
River	256 x 256	0.53	8.29	16x	7.82
Lake	512 x 512	0.12	2.09	17x	1.16

Table 1. Comparison of performance (shown in Frame Per Seconds) for different terrains: on CPU (FPS CPU); on GPU rendered with fixed functionality (FPS GPU); and on GPU with full lighting effects (FPS GPU + FX).

In the next example we show different internal transmittance values for water. We used a height map consisted of two planes forming a slope shown in Figure 8 (a). The maximum depth is 15.5m, and in the shallowest part the water has depth of 0.5m. Figure 8 (b) shows the internal transmittance of pure water and chlorophyll concentration of 70, by just sampling the transmittance graph for chlorophyll-rich green oceanic waters [Morel and Prieur 1977].

We ran a simulation of a fountain with free parts. The source of the water is located at the top of the fountain. Two sharp steps make the water break into small waterfalls. Figure 9 shows the results of the simulation for a maximum of 65536 particles, using two 256×256 textures. The simulation runs at approximately 2 frames/seconds, with possible further optimizations to increase the frame rate.



(a)

(b)

Figure 8. (a) Height map for a two-plane slope; (b) light attenuation and internal transmittance for pure water and chlorophyll-rich water.

6 Discussion

The results of the experiments showed that there is an average increase of a factor of 16 in speed when running the simulation on the graphics hardware. The combination of the simulation data in the rendering pipeline and the height field representation of surfaces resulted in near real-time animations of water flowing on terrains for the smaller terrain maps. The compact representation of the simulation data, i.e., two RGBA textures of same resolution for a given terrain, makes the water column-based simulation an attractive option for interactive applications.

The obvious drawback of the water column-based model is the vertical isotropy. External forces on the water surface are unconditionally applied in the vertical direction, and certain water phenomena cannot be represented within this model, such as vortices. Another drawback is that this model suffers from instability over large time steps, which can cause undesired oscillations on the water surface.

Some visual artifacts appear on the interpolated ground texture due to the height map resolution and the calculation of the refraction in the vertex shader; see Figure 9 (b). The height map resolution can be used as trade-off between interactively previewing the animation and a final high-quality rendering.

Without a particle model, the volume of water transferred through the weir flow is immediately deposited in the neighboring cell. This results in an unnatural rise of the column height, as discussed in [Holmberg and Wunsche 2004]. The particle system requires additional computational time, affecting the overall performance. The main drawback is the cost of the data transfer from the GPU to the CPU, since the CPU must allocate memory for the particles.

On the rendering side, the terrain ground could have better lighting, including self-shadowing [Policarpo et al. 2005] and other underwater phenomena [Loviscach 2003; Iwasaki et al. 2003] such as caustics. The particle rendering needs a substantial improvement, such as in [Takahashi et al. 2003].

7 Conclusion

We have proposed an optimization of the height-field based water volume model for efficient three-dimensional water flow simulation on terrains. We showed that the model has the advantage of low memory consumption. Running the simulation in parallel on graphics hardware also aids with realistic rendering of water surfaces with considerably less data transfer per frame. The irregular terrain ground under the water can also be accurately rendered without simplifications.

The results of simulations running on the GPU of different terrains showed a considerable increase in speed with near interactive frame rates. In addition, the compact memory storage makes the proposed method an attractive approach for water flow on natural scenery for Computer Graphics animation.

Acknowledgements

This work was partially supported by the Japanese Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Exploratory Research 17650021.

References

ANDERSON, J. D. 1995. Computational Fluid Dynamics: The Basics with Applications, McGraw-Hill.

CHIBA, N., SANAKANISHI, S., YOKOYAMA, K., OOTAWARA, I., MURAOKA, K., AND SAITO, N. 1995. Visual Simulation of Water Currents Using a Particle-based Behavioural Model, *The Journal of Visualization and Computer Animation*, 6, 3, 155-171.

ENRIGHT, D., MARSCHNER, S., FEDKIW, AND R. 2002. Animation and Rendering of Complex Water Surfaces, *ACM Transactions on Graphics*, 21, 3, 736-744.

FOSTER, N., AND FEDKIW, R. 2001. Practical Animation of Liquids, In *Proceedings of ACM SIGGRAPH 2001*, 23-30.

FOSTER, N., AND METAXAS, D. 1996. Realistic Animation of Liquids, *Graphical Models and Image Processing*, 58, 5, 471-483.

GPGPU. http://www.gpgpu.org/

HARRIS, M. 2004, Fast Fluid Simulation on the GPU, *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*. Addison-Wesley Professional.

HOLMBERG, N., AND WUENSCHE, B. 2004. Efficient Modeling and Rendering of Turbulent Water over Natural Terrain. In *Proceedings of GRAPHITE 2004*, 16-18.

IGLESIAS, A. 2004. Computer graphics for water modeling and rendering: a survey, *Future Generation Computer Systems*, 20, 8, 1355-1374.

IRVING, G., GUENDELMAN, E., LOSASSO, F., AND FEDKIW, R. 2006. Efficient Simulation of Large Bodies of Water by Coupling Two and Three Dimensional Techniques. ACM Transactions on Graphics, 25, 3, in press.

IWASAKI, K., DOBASHI, Y., AND NISHITA, T. 2003. A Fast Rendering Method for Refractive and Reflective Caustics Due to Water Surfaces, *Computer Graphics Forum*, 22, 3, 601-609.

KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In *Proceedings of ACM SIGGRAPH 1990*, 49-57.

KIPFER, P., AND WESTERMANN, R. 2006. Realistic and Interactive Simulation of Rivers. In *Proceedings of Graphics Interface*, 41-48.

LIU, Y., LIU, X., AND WU, E. 2004. Real-Time three-dimensional Fluid Simulation on GPU with Complex Obstacles, In *Proceedings of Pacific Conference on Computer Graphics and Applications*, 247-256.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure, *ACM Transactions on Graphics*, 23, 3, 457-462.

LOVISCACH, J. 2003. Complex Water Effects at Interactive Frame Rates. *WSCG International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 11, 1, 298-305.

MOREL, A., AND PRIEUR, L. 1977. Analysis of variations in ocean color, *Limnology and Oceanography*, 22, 709-722.

MOULD, D., AND YANG, Y. H. 1997. Modeling Water for Computer Graphics, *Computers & Graphics*, 21, 6, 801-814.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-Based Fluid Simulation for Interactive Applications, In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 154-159.

NEYRET, F., AND PRAIZELIN, N. 2001. Phenomenological Simulation of Brooks, In *Proceedings of Eurographics Workshop on Animation and Simulation*, 53-64.

O'BRIEN, J.F., AND HODGINS, J. K. 1995. Dynamic Simulation of Splashing Fluids, In *Proceedings of Computer Animation*, 198-205, 220.

POLICARPO, F., OLIVEIRA, M. M., AND COMBA, J. L. D. 2005. Real-Time Relief Mapping on Arbitrary Polygonal Surfaces, Symposium on Interactive three-dimensional Graphics and Games, pp. 155-162.

PREMOZE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A., AND WHITAKER, R. T. 2003. Particle-Based Simulation of Fluids, *Computer Graphics Forum*, 22, 3, 401-410.

ROCHET, F. 2005. *Simulation Réaliste de Ruisseaux en Temps Réel*, Masters thesis, Université Joseph Fourier, France.

SCHLICKE , T. 2001 *Breaking Waves and the Dispersion of*

Surface Films. PhD thesis, University of Edinburgh.

STAM, J. 1999. Stable Fluids, In *Proceedings of ACM SIGGRAPH 1999*, 121-128.

SUN, Y., FRACCHIA, F. D., AND DREW, M. S. 1999. Rendering the Phenomena of Volume Absorption in Homogeneous Transparent Materials, In *Proceedings of IASTED International Conference on Computer Graphics and Imaging*, 283-288.

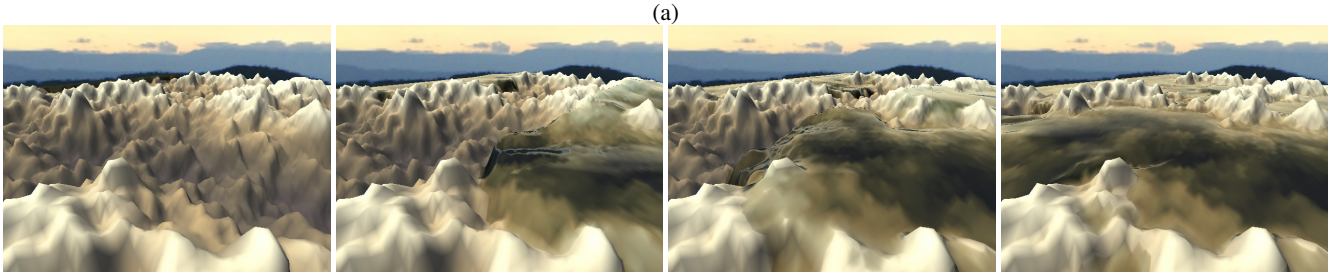
TAKAHASHI, T., FUJII, H., KUNIMATSU, A., HIWADA, K., SAITO, T., TANAKA, K., AND UEKI, H. 2003. Realistic Animation of Fluid with Splash and Foam, *Computer Graphics Forum*, 22, 3, 391-400.

THON, S., AND GHAZANFARPOUR, D. 2001. A Semi-Physical Model of Running Water, In *Eurographics UK 2001 Conference Proceedings*, 53-59.

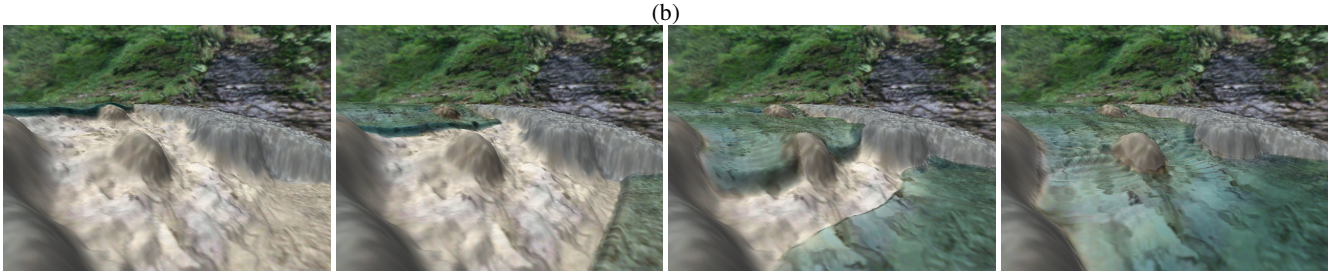
THON, S., AND GHAZANFARPOUR, D. 2002. Real-Time Animation of Running Waters Based on Spectral Analysis of Navier-Stokes Equations, *Computer Graphics International*, 333-346.

WU, E., LIU, Y., AND LIU, X., 2004. An Improved Study of Real-Time Fluid Simulation on GPU, *Journal of Computer Animation and Virtual World*, 15, 3-4, 139-146.

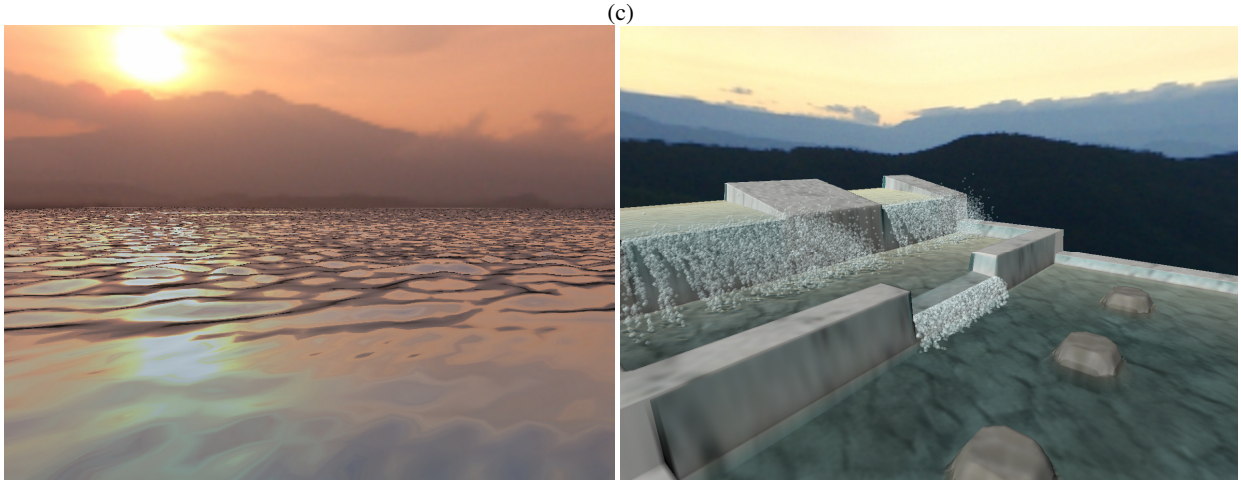
(a)



(b)



(c)



(d) (e)

Figure 9. Animation frames for different terrains: (a) water collapsing in a puddle 1.5m×1.5m×0.15m; (b) fractal terrain flooding 12m×12m×2m (generated with Perlin noise); (c) river flow 50m×50m×10m; (d) lake sunset 200m×200m×50m; (e) fountain 20m×20m×5m.