Low-Memory and Interactive-Rate Animation of Water-Column Based Flows

Marcelo M. MAES† Tadahiro FUJIMOTO‡ Norishige CHIBA‡
Department of Computer Science, Faculty of Engineering, Iwate University, Japan
E-mail: † marcelo@cg.cis.iwate-u.ac.jp, ‡ {fujimoto | nchiba}@cis.iwate-u.ac.jp

Abstract

We present an optimization of the height-field water-column based approach for water simulation, providing three-dimensional animation of water flow on natural terrains. Our approach eliminates the storage and management of redundant virtual pipes between columns of water and also removes output dependency for parallel implementation, making it efficient for interactive computer graphics applications. We show a GPU implementation of the proposed method that runs at interactive frame rates with rich lighting effects on the water surface, including light wavelength dependent attenuation and light scattering.

Keywords: Natural Phenomena, Physically-Based Animation, Water Simulation, Height-Field, Light Transport.

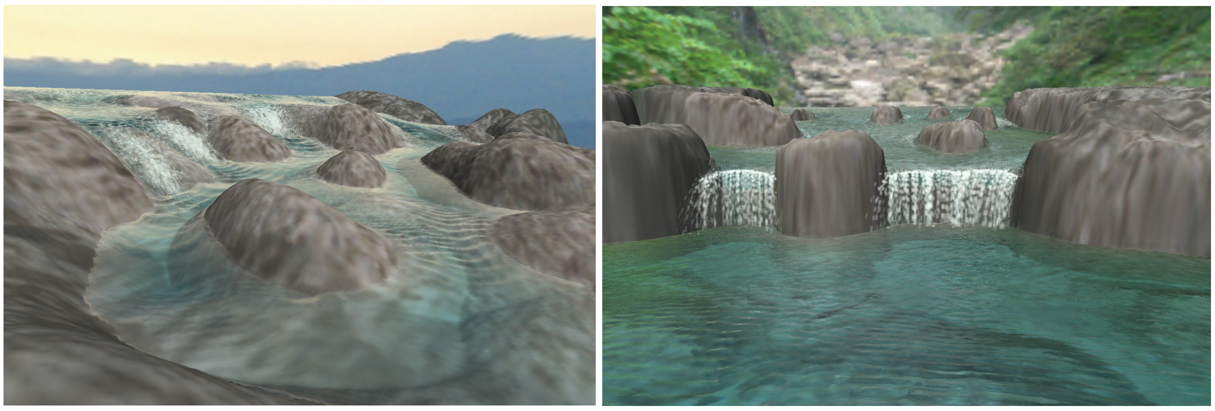


Figure 1. Water flowing on irregular terrain.

1. Introduction

Water representation and animation have been intensively investigated in computer graphics due to the complexity of its behavior and 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 animation frame. Interactive applications such as landscape design, virtual reality, and games, which often need three-dimensional water animation at interactive frame 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 the water [10]. Therefore several methods must be combined to produce realistic animations. Preferably, these methods should be based on physics to behave as its physical counterpart and to interact with each other. However, computer graphics applications don't need the same degree of accuracy as engineering applications, so they usually sacrifice accuracy for efficiency.

Water flowing on terrains generates several natural phenomena, including rivers, waterfalls, puddles, and lakes; see Figures 1 and 10. This flow is mainly dominated by gravity and the water is near vertical equilibrium against the ground [11]. Since terrains are highly irregular, 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 height-field water-column based approach for water simulation previously introduced by [23, 20, 9], and a further extension of [17]. 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, which has several advantages that our approach benefits as well:

- Hydrostatic physics calculation has low computational cost;
- The model intrinsically 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;
- The three-dimensional simulation has squared computational cost, proportional to the two-dimensional resolution of the height-field;
- 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 a coupled 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-column 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 rendering every frame;
- A single height-field to represent both terrain and water surface, reducing the geometry rendered per frame;
- Accurate rendering of refraction, light attenuation and scattering, taking into account the water depth.

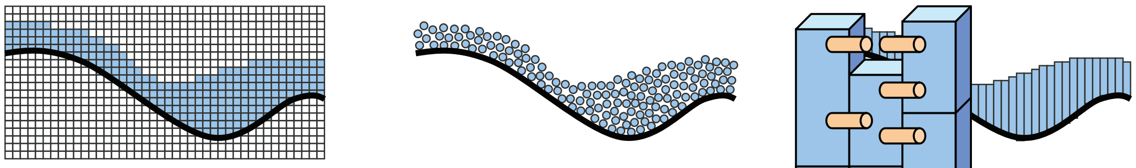
We describe the related work in fluid simulation for computer graphics in Section 2. We show our contribution to the water model and its parallelization in Section 3. We present the rendering process along with the light model for the water in Section 4. In Section 5, we present the results by showing several experiments. We conclude this work in Section 6.

2. Related 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 [10]. Numerical solutions of the NSE [1] 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 [6]. 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 [30]. 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 [8, 36] and a three-dimensional by [15]. Although these simulations run in real-time, they do not address the problem of simulating fluids with free boundaries, such as water.



(a) (b))

Figure 2. Water flow simulation on terrain using diffe
(a) regular grid; (b) particles; (c) columns of

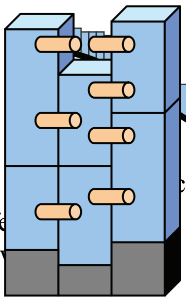


Figure 3. Water-column model with two cells per column ($n_{\text{cell}} = 2$). Virtual pipes created between overlapping cells and unobstructed vicinity.

The free boundary issue is addressed with a hybrid particle and level set method [5, 4, 16, 11]. An implicit function evolves together with the fluid simulation to track the isocontour representing the interface of the liquid. Particles are used near 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. [16] 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.

Irving et al. [11] 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. [2] 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. [21] 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 [14] 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 surface representation does not require a dense particle set, it is not volume conserving. Surface details, such as waves, depend directly on the height-field resolution, which was apparently coarse in their examples to keep interactive frame rates.

Premože et al. [25] 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 are 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 [22, 33, 34, 26] 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 include realistic surface phenomena not present in low-resolution three-dimensional simulations, they cannot accurately represent water flowing on irregular terrains and other three-dimensional effects, such as splashes and falls without the information about the water volume.

3. Proposed Efficient Method

Kass and Miller [13] 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 [23]. 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 height threshold.

Mould and Yang [20] extended this model further by running the simulation on an arbitrary height-field and by reducing the vertical isotropy by the subdivision of each column into multiple cells; see Figure 3. However, external forces on the water surface are unconditionally applied in the vertical direction. Certain phenomena also cannot be represented with this model such as vortices. They also extended the particle model by including bubbles rising inside the water.

Later, Holmberg and Wunsche [9] applied this model to simulate the natural movement of rivers, rapids and waterfalls. They use ray-tracing to render the results, which is intended to be a non-interactive simulation and animation tool.

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.

3.1 Water Volume Model

Here we review the model used in the simulation of the main water volume and the related equations, according to [23, 20, 9].

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. The number of cells per column, n_{cell} in Figure 3, can be fixed or input by the user depending on the implementation.

Virtual pipes are created horizontally between adjacent columns where their cells overlap; see Figure 3. No pipe is created vertically between stacked cells in the same column. Their height varies due to the flow of water 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:

$$p = h\rho g + p_0 + p_e \qquad (1)$$

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 columns is given by:

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

where f is a non-physical frictional coefficient, as suggested in [20] 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.

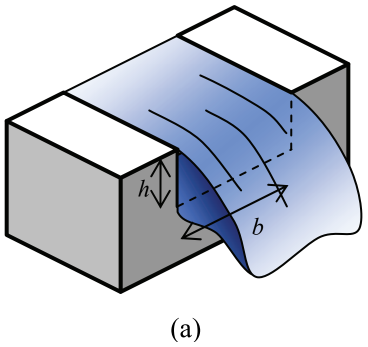
Given the flow velocity in the pipe, the volume of water that should be moved through it is:

$$V_{pipe} = \Delta t \eta c \qquad (3)$$

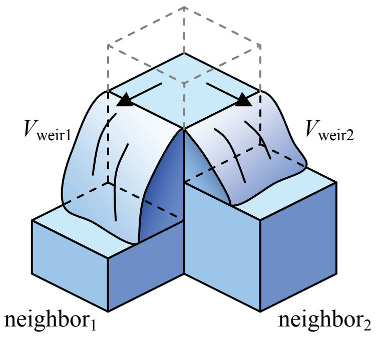
where c is the cross sectional area of the pipe, i.e. the amount of overlap between adjacent cells. The volume transferred is translated into height changes between the cells. Since mass must be conserved, all pipes that are removing fluid from a cell are scaled back if the volume of that cell becomes 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.

To model water that breaks free from the main volume of water, such as splashes and waterfalls, Holmberg and Wunsche [9] calculates the volume of water that flows through a weir; see Figure 4 (a). In their work, this model is used when the height of a wave crest becomes unstable. The flow rate through a weir is given by:



(a)



(b)

Figure 4. (a) Flow through a weir (b) applied to the discretized water-column model.

$$\zeta = \frac{2}{3} b h^{\frac{3}{2}} \sqrt{2g} \qquad (4)$$

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

$$V_{weir} = \Delta t \zeta \qquad (5)$$

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 direction; see arrows in Figure 4 (b).

3.2 Reduction of Memory Requirements

We assume the initial condition of the simulation is static, i.e. flow velocities are zero. We then note that the pressure difference between any two leveled submerged points is the same for adjacent columns; for example $(p_1 - p'_1) = (p_2 - p'_2) = (p_3 - p'_3)$ in Figure 5 (a). The resulting flow in each pipe, Equation 2, will be the same ($\eta_1 = \eta_2 = \eta_3$).

To reduce the memory storage, we calculate and store the flow η of just one pair of those points, e.g. pair with pressure difference $(p - p')$ in Figure 5 (b). Consequently, to calculate the transferred volume of water, we must check if two cells overlap for every simulation step. This process does not

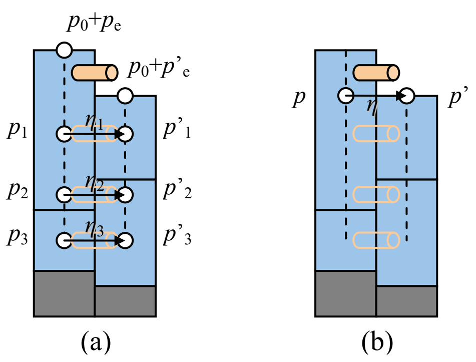


Figure 5. (a) Flow occurs due pressure difference between adjacent columns. (b) Flow storage reduction.

increase the overall complexity since the same process must be performed in the original algorithm to check whether a pipe should be created or deleted. Thus we reduce the maximum memory requirements per adjacent columns from $2 \times n_{\text{cell}}$ to only 2, i.e. one pipe between adjacent columns and one pipe connected to the air, independent of the number of stacked cells, see Figure 5 (b).

We also note that the flow through a weir, Equation 4, does not depend on the flow rate from the previous time step, and we adopt this model for all flow between a column of water and the adjacent air above a lower column, see Figure 7. Besides reducing the maximum number of stored virtual pipes between adjacent columns to only 1, we also have a single model for unobstructed water flow. The simulation results show no change in the behavior of the water surface, such as the wave propagation phenomenon; see Figure 10 (a).

On a two-dimensional rectilinear grid, each column has 8 neighbors. To avoid duplicated virtual pipes, we store only 4 unique virtual pipes per column, as shown in figure 6 (b). Storing only 1 flow velocity per neighboring column not only reduces the memory footprint, but also reduces the amount of memory access and calculation time. For example, given a height-field with resolution of 256×256, and flow velocity stored as 32-bit floating point values as shown in Figure 6 (b), a total memory of $256 \times 256 \times (2 \times n_{\text{cell}}) \times (4 \text{ floats} \times 4 \text{ bytes/float}) = 4\text{MB}$ would be necessary for $n_{\text{cell}} = 2$. This value drops to 1MB ($\frac{1}{2 \times n_{cell}} = 25\%$) with our optimization. Although such amount of memory may be low for recent hardware, the memory access and the calculation to update the flow values are $256 \times 256 \times (2 \times n_{\text{cell}}) = 262144$ times for $n_{\text{cell}} = 2$, or 65536 times (25%) with our optimization. These values become much more significant for height-fields with higher resolutions.

3.3 Simulation Parallelization

Recently commodity graphics hardware has become inexpensive, programmable, and has been used as a general purpose processing unit [7]. The GPU (Graphics Processing Unit) is capable of running vertex and fragment programs in parallel on stream processors.

The first generations of programmable graphics hardware can only perform *gather* memory operations, where any stream processor can access the input shared data (texture access), but the output can only be written in a single memory position, i.e. the rendered fragment.

Therefore, to promote parallelization of the water simulation, we have to produce an independent output. We do that by adding all water inflow from neighboring columns, and subtracting the all outflow from the column being processed.

For every column, we calculate all volume out-flowing the cells; Equations 3 and 5. If the volume is larger than the column volume, we apply the following scale factor to the outflow:

$$\text{scale factor} = \frac{V_{column}}{\sum_{i}^{out} V_{pipe_i} + V_{weir_i}} \tag{6}$$

where V_{column} is the column volume; *out* is the total of neighbors with outflow through a pipe or a weir; V_{pipe} and V_{weir} are the respective volumes out-flowing from the column to the neighbor *i*.

We can either calculate all neighbors' outflow when processing a single column, or have an extra rendering pass to store in memory the outflow scaling. We choose the latter since it performs about 5 times faster by avoiding the same calculation of the same neighbors several times.

We store the simulation data structures in two-dimensional textures: one for the flow velocity and one for the column height, as shown in Figure 6. Fragment shaders are then used to update the stored values using one-to-one pixel-to-texel mapping [7].

The input terrain is given by a height-field in a gray scale image and its height is scaled and stored in a floating-point texture. The sampling for the columns of water is created with the same resolution as the terrain height-field. To reduce the

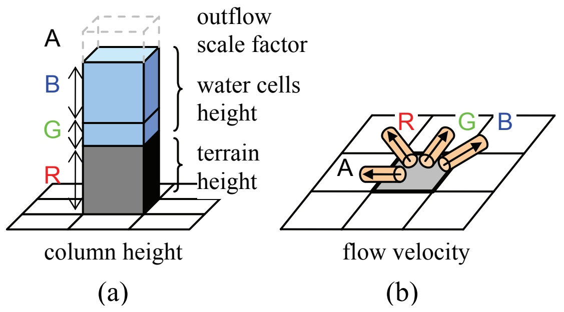


Figure 6. Simulation data textures: (a) terrain height packed with two water cells per column in one texel; (b) flow through pipes between adjacent columns and the flow direction.

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 components can store cells of a single column; see Figure 6 (a). We use the alpha channel of the height texture to store the outflow scaling factor obtained in Equation 6.

A single texel can store up to 2 cells per column ($n_{\text{cell}} = 2$), as shown in Figure 6 (a), resulting in minimum texture access during the simulation and rendering stages. More cells ($n_{\text{cell}} > 2$) can be stored in additional textures, which will hit the overall performance due to the extra memory access and cell overlap calculations.

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 6 (b) for the texture arrangement of pipes and flow direction between 8-neighboring columns.

We summarize the simulation process, together with other auxiliary textures, with the following rendering passes:

I) Update the flow velocities (Equation 2):

- read a texture with external pressure values derived from forces of interacting external objects;
- read the texture with the column heights and calculate the pressure difference (Equation 1).

II) Calculate the outflow scaling (Equation 6):

- read the updated flow and the column heights to calculate the total volume leaving each column;
- write the scaling value to the alpha channel of the column height texture (Figure 6 (a)).

III) Update the water-column volume:

- read the flow value between columns and calculate the overlapping area between adjacent cells (Equation 3);
- calculate the volume of transferred to the air above a lower column (Equations 4 and 5), including free parts disconnecting from the water-column;
- sum all inflow volumes and subtract the outflow;
- write the free volume in another texture with the averaged velocity of flows between neighbors.

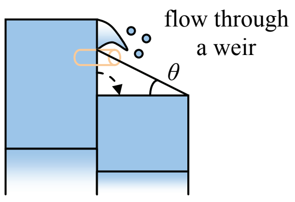


Figure 7. Slope threshold for breaking waves.

3.4 Particle System

We implement a simple particle system to illustrate the interaction of free parts of fluid, such as splashes and falls, with the proposed optimized water volume model. We do not consider inter-particles interaction [9], only the influence of gravity.

One of the most useful definitions of breaking waves is that breaking occurs when the wave slope exceeds a critical angle θ [28]. We let the user specify the slope threshold to control the particle creation and in consequence, the memory space needed to store the particles.

Only the partial volume that exceeds the slope threshold is used to generate particles, see Figure 7. The rest of the volume, i.e. bellow the threshold, is simply transferred to the lower neighbor column.

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 [20]. The volume of collided particle is then absorbed back in the main water volume.

Other two textures keep the position and the velocity of every generated particle. We use the alpha channel of these textures to store respectively the external pressure after collision, and the particle volume. The resolution of these two textures will limit the number of active particles in the system.

The velocity and positions are updated as the rendering passes summarized bellow:

I) Update the particle position

- read the previous particle position and update it according to its velocity.

II) Update the particle velocity

- read the position and the column heights to check the collision of the particle;
- if they collide, calculate the external forces and the resulting vertical external pressure, otherwise update the velocity according to the gravity acceleration.

III) Create new particles

- read the free volume texture from the GPU to the CPU and calculate the number of particles to create according to a user-defined unit volume in case one particle represents several droplets of water;
- read the position texture and search for empty slots (empty volume) to create the new particle;
- render each new particle as a point in the particle position and velocity textures

IV) Apply collisions as external pressure

- render each particle as a point in the external pressure texture;
- if the particle has collision information (calculated in pass II) write the external pressure value and the volume to merge with the water-column volume.

The particle creation requires the transfer of data from the GPU memory to the CPU memory, which drastically slows down the overall performance of the system. Therefore we combine a number of water-column simulation steps followed by one particle simulation step. This not only reduces the time consuming data transfer operation per water-column simulation step, but also provides smooth particle volume absorption after collision.

4. Rendering

We render the terrain and the water surface as a single height-field since they have the same resolution and their heights are packed together in a single texture. We then interpolate from one material to the other with a fragment shader in the GPU. This reduces the geometry and the number of texture access.

We use a vertex shader to displace the height of a planar grid mesh, which can reside in the GPU memory for maximum performance. Therefore, there is minimum data transfer to the GPU for rendering.

We render the particles as textured point sprites, with size proportional to the volume and inversely proportional to the distance from the viewer; see Figure 1. Then we blend them with the current rendered frame. At this time, no sorting and complex shading are done when rendering the particles.

4.1 Light Transport

For each fragment rendered, we calculate the light transport on the surface of the main volume of water. All access to the column's float-point texture is performed with bilinear interpolation of its heights to avoid visual artifacts due to discretization of the height-field.

We calculate light reflection **R** and refraction **T** on the water surface based on the eye direction **E** and the surface normal **N**; refer to Figure 8. We use Snell's law for the refraction of light from the air to the water, with refractive indices of 1 and 1.33 respectively.

The Fresnel factor defines the ratio of reflection and refraction of non-polarized light on a dielectric material. We use the Schlick's approximation [27] for the Fresnel reflectivity given by:

$$R = f_\lambda + (1 - f_\lambda)(1.0 - \mathbf{N} \cdot \mathbf{E})^5 \qquad (7)$$

where f_λ is the spectral distribution of the Fresnel factor at normal incidence. The spectral distribution f_λ depends on the wavelength of the light λ, but we use a single value for all spectrum. This value is ≈0.02 for water [29]. The transmission coefficient T is the complement of the reflectivity for energy conservation, given by:

$$T = 1 - R \qquad (8)$$

The intensity of light arriving in the eye direction **E** is then given by:

$$I_\mathbf{E} = RI_\mathbf{R} + TI_\mathbf{T} \qquad (9)$$

where $I_\mathbf{R}$ is the intensity of light coming from the reflected ray **R**, which is mapped to a cubemap, assuming that the environment is far away from the surface. $I_\mathbf{T}$ is the light coming from the refracted ray, calculated as follows.

We accurately compute the intersection of the refracted ray and the terrain ground through a linear search with fixed increments δ, followed by a binary search [24] with increments of $\pm\delta/2^{level}$, where *level* is the step number of the binary search. Refer to Figure 8 for a schematic diagram of the process.

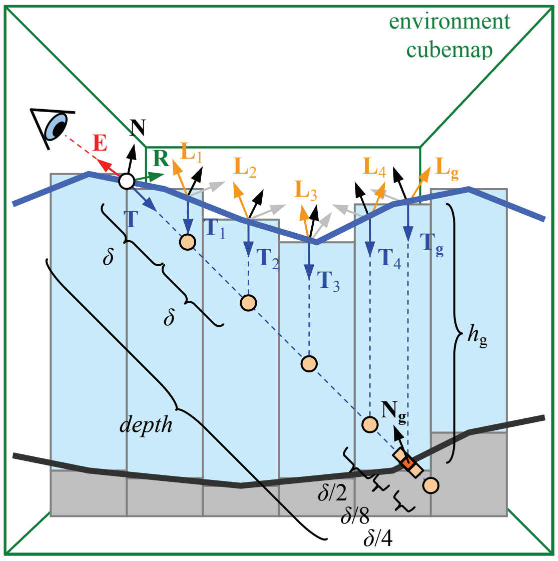


Figure 8. Light transport: reflection, refraction, attenuation, and single scattering.

The attenuation of light in transparent volumes does not only decrease the color intensity, but also deepens the color saturation and changes the hue [31]. The internal transmittance for liquid solutions is given by the Beer's law:

$$T_{\text{internal}}(\lambda, l) = 10^{-a(\lambda)cl} \tag{10}$$

where λ is the light wavelength; $a(\lambda)$ 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 pure water for the RGB wavelengths, with coefficients (0.648, 0.053, 0.036) m^{-1} respectively [35]. This approximation may introduce significant errors due to spectrum undersampling [31]; however it results in a plausible water appearance.

We assume that the highest contribution of light under the water comes from the vertical direction above. The lighting in the ground is then calculated from the intensity of light coming from the vertical refracted ray $\mathbf{T}_\mathbf{g}$, shown in Figure 8. The Fresnel term is also applied to modulate the intensity transmitted to the water. The diffuse lighting at the ground is given by the dot product $\mathbf{N}_\mathbf{g}\cdot(-\mathbf{T}_\mathbf{g})$, and the intensity is attenuated by the height *h*.

We also assume the water has impurities that not only further attenuates the light, but also contributes with scattered light. For the attenuation, we use a simple exponential decay, given by:

$$A(l) = e^{-kl} \tag{11}$$

where k is the attenuation coefficient, and l is the length of the light path.

We use a single scattering method [3, 18] at each linear search interval δ. This approach has low computational cost since it shares the height data already available from each step of the depth search process. However, if the interval δ is too large, the scattering calculation may introduce visual artifacts to the water appearance.

The vertical incoming light is modulated by $\mathbf{T}\cdot\mathbf{T}_\text{i}$. The light intensity arriving at the refracted ray in the direction $\mathbf{T}$ is the ground lighting plus the scattered light due to impurities, given by:

$$\begin{aligned} I_\mathbf{T} &= I_{\mathbf{L}_\mathbf{g}} T_{\mathbf{T}_\mathbf{g}} \left(\mathbf{N}_\mathbf{g}\cdot(-\mathbf{T}_\mathbf{g})\right) T_{\text{internal}}(\lambda, h_\text{g} + depth) A(h_\text{g} + depth) \\ &+ \sum_{i=1}^{n} I_{\mathbf{L}_i} T_{\mathbf{L}_i} (\mathbf{T}\cdot\mathbf{T}_i) T_{\text{internal}}(\lambda, h_i + i\times\delta) A(h_i + i\times\delta)\,\delta \end{aligned} \tag{12}$$

where $I_\mathbf{T}$, $I_\mathbf{Lg}$, $I_\mathbf{Li}$ are respectively the light intensities at the surface of the refracted ray from the viewer direction, the light coming from the $\mathbf{L}_\mathbf{g}$ and $\mathbf{L}_i$ directions. $T_\mathbf{Tg}$ and $T_\mathbf{Li}$ are respectively the Fresnel transmission factors (Equation 8) for the vectors $\mathbf{T}_\mathbf{g}$ and $\mathbf{L}_\text{i}$ (refer to Figure 8). n is the number of steps in the linear search inside the water and above the ground level; h_i is the height from the step position to the water surface; and *depth* is the length resulting from the ground's depth search.

5. Results

All the experiments shown here ran on an Intel Pentium 4 processor at 3.4GHz with 1GB of memory, and an NVIDIA GeForce 6600GT graphics card with 128MB of memory. OpenGL and OpenGL Shading Language were used for all GPU operations.

We first ran a simulation with different column subdivisions of one, two and three cells per column. The performance drop for two and three cells per column 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 vertical velocity. Otherwise one cell per column is sufficient for the appearance and animation effects shown in Figures 1, 9, and 10.

We implemented only one cell per column for efficiency and aiming only for the water surface appearance. See Figure 10 for some frames of the animation. Table 1 shows the performance results for different terrains running at simulation time steps of 0.005s and with the particle system

Table 1. Computational time of simulation for different height-field resolutions.

Resolution (pixels)	Simulation on the CPU	Simulation on the GPU	Speed-up
64×64	158 ms	2 ms (500 fps)	79×
128×128	665 ms	4 ms (250 fps)	166×
256×256	2705 ms	13 ms (76 fps)	208×
512×512	11055 ms	48 ms (20 fps)	230×

Table 2. Computational time of simulation and rendering for 1 animation frame.

Resolution (pixels)	10 steps of water simulation	1 step of particle simulation	Per-fragment water rendering	Point sprite particle rendering**
64×64	11 ms	50 ms *(2106)	129 ms	269 ms
128×128	31 ms	142 ms *(5800)	229 ms	697 ms
256×256	122 ms	141 ms *(7491)	677 ms	478 ms
512×512	471 ms	342 ms *(12791)	3265 ms	1155 ms

*(average number of particles generated)
**with motion blur

disabled.

For the simulation to match a real-time animation, it would be required to run at 200 frames per second (1/0.005s). That is true for smaller height-fields up to 128×128 pixels running on the GPU. For larger height-fields, the animation runs slower than the corresponding simulation time step. Nevertheless, the simulation presents a two order speed increase and its behavior can be observed at interactive frame rates.

We set up the next experiment with a single height-field but with different resolutions to compare the interaction of the water-column model and the particle system; see Figure 1 for an animation frame. Since we perform per-fragment lighting, the viewer position must be the same so that the water surface covers the same amount of pixels in the viewport. The slope threshold that controls the particle generation was set to 2. On a preliminary test we measure the number of particles generated by the system and we set the textures for the particles' position and velocity as the next power-of-two resolution. Finally, we set the ratio of 10 steps of the water-column simulation for 1 step of the particle simulation. With a time step of 0.003s for the water-column simulation, the particle animation is updated every 0.03s (about 33 frames per second). Table 2 shows the performance results.

One of the bottlenecks found for the water rendering was the texture access for the terrain height displacement. It consumed over 50% of the rendering time for the larger resolution of the height-fields. This is mainly due to the limitation of the first generation of graphics cards with float-point texture access from vertex shaders. However, the larger the resolution of the height-field was, more vertices falling on the same pixel of the viewport had to be processed. This not only increases the processing time by calculating redundant pixels, but it also generate aliasing artifacts, see Figure 10 (b). This indicates that a LOD (Level Of Detail) method for the height-field visualization is necessary.

In Figure 9, we show how different properties related to Equations 10 and 11 affect the water appearance. We also include the results of a different sampling of the absorption spectrum for chlorophyll-rich oceanic waters [19].

Our particle rendering implementation still needs depth sorting and other visual improvements [32]. The life cycle of the particles must also be extended to include mist, bubbles and foam formation after the impact with the main volume of water.

6. Conclusion

We have proposed an optimization of the height-field water-column volume model for efficient three-dimensional water flow simulation on terrains. We showed that the proposed 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 under the water can also be accurately rendered without simplifications.

The results of simulations running on a legacy graphics card for different terrains showed a two order increase in speed with interactive frame rates. In addition, the compact memory storage makes the proposed method an attractive approach for water flow in natural scenery for computer graphics animation.

As future work, we would like to apply a LOD method for the height-field visualization, and to optimize the particle system. We also would like to improve the lighting model in the water by sampling more rays for the ground illumination to achieve caustics, and to use HDR (High Dynamic Range) environment cubemaps for better lighting.

Acknowledgement

This work was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (B) 19300022.

References

[1] Anderson, J. D. Computational Fluid Dynamics: The Basics with Applications. McGraw-Hill, 1995.
[2] Chiba, N., Sanakanishi, S., Yokoyama, K., Ootawara, I., Muraoka, K., and Saito, N. Visual Simulation of Water Currents Using a Particle-based Behavioural Model. The Journal of Visualization and Computer Animation, Vol. 6, No. 3, pp. 155-171, 1995.
[3] Dobashi, Y., Kaneda, K., Yamashita, H., Okita, T., and Nishita, T. A Simple, Efficient Method for Realistic Animation of Clouds. In Proceedings of ACM SIGGRAPH 2000, pp. 19-28, 2000.
[4] Enright, D., Marschner, S., Fedkiw, and R. Animation and Rendering of Complex Water Surfaces. ACM Transactions on Graphics, Vol. 21, No. 3, pp. 736-744, 2002.
[5] Foster, N., and Fedkiw, R. Practical Animation of Liquids. In Proceedings of ACM SIGGRAPH 2001, pp. 23-30, 2001.
[6] Foster, N., and Metaxas, D. Realistic Animation of Liquids. Graphical Models and Image Processing, Vol. 58, No. 5, pp. 471-483, 1996.
[7] GPGPU. http://www.gpgpu.org/
[8] Harris, M. Fast Fluid Simulation on the GPU. GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics. Addison-Wesley Professional, 2004.
[9] Holmberg, N., and Wunsche, B. Efficient Modeling and Rendering of Turbulent Water over Natural Terrain. In Proceedings of GRAPHITE 2004, pp. 16-18, 2004.
[10] Iglesias, A. Computer graphics for water modeling and rendering: a survey. Future Generation Computer Systems, Vol. 20, No. 8, pp.1355-1374, 2004.
[11] Irving, G., Guendelman, E., Losasso, F., and Fedkiw, R. Efficient Simulation of Large Bodies of Water by Coupling Two and Three Dimensional Techniques. ACM Transactions on Graphics, Vol. 25, No. 3, pp. 805-811, 2006.
[12] Iwasaki, K., Dobashi, Y., and Nishita, T. A Fast Rendering Method for Refractive and Reflective Caustics Due to Water Surfaces. Computer Graphics Forum, Vol. 22, No. 3, pp. 601-609, 2003.
[13] Kass, M., and Miller, G. Rapid, stable fluid dynamics for computer graphics. In Proceedings of ACM SIGGRAPH 1990, pp. 49-57, 1990.
[14] Kipfer, P., and Westermann, R. Realistic and Interactive Simulation of Rivers. In Proceedings of Graphics Interface, pp. 41-48, 2006.
[15] Liu, Y., Liu, X., and Wu, E. Real-Time three-dimensional Fluid Simulation on GPU with Complex Obstacles. In Proceedings of Pacific Conference on Computer Graphics and Applications, pp. 247-256, 2004.
[16] Losasso, F., Gibou, F., and Fedkiw, R. Simulating water and smoke with an octree data structure. ACM Transactions on Graphics, Vol. 23, No. 3, pp. 457-462, 2004.
[17] Maes, M. M., Fujimoto, T., and Chiba, N. Efficient Animation of Water Flow on Irregular Terrains. In Proceedings of GRAPHITE 2006, pp.107-115, 2006.
[18] Miyazaki, R., Dobashi, Y., and Nishita, T. A Fast Rendering Method of Clouds Using Shadow-View Slices. In Proceeding Computer Graphics and Imaging 2004, pp. 93-98, 2004.
[19] Morel, A., and Prieur, L. Analysis of Variations in Ocean Color. Limnology and Oceanography, Vol. 22, pp. 709-722, 1977.
[20] Mould, D., and Yang, Y. H. Modeling Water for Computer Graphics. Computers & Graphics, Vol. 21, No. 6, pp. 801-814, 1997.
[21] Müller, M., Charypar, D., and Gross, M. Particle-Based Fluid Simulation for Interactive Applications. In Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation, pp. 154-159, 2003.
[22] Neyret, F., and Praizelin, N. Phenomenological Simulation of Brooks. In Proceedings of Eurographics Workshop on Animation and Simulation, pp. 53-64, 2001.
[23] O'Brien, J.F., and Hodgins, J. K. Dynamic Simulation of Splashing Fluids. In Proceedings of Computer Animation, pp. 198-205, 220, 1995.
[24] Policarpo, F., Oliveira, M. M., and Comba, J. L. D. Real-Time Relief Mapping on Arbitrary Polygonal Surfaces. Symposium on Interactive three-dimensional Graphics and Games, pp. 155-162, 2005.
[25] Premože, S., Tasdizen, T., Bigler, J., Lefohn, A., and Whitaker, R. T. Particle-Based Simulation of Fluids. Computer Graphics Forum, Vol. 22, No. 3, pp. 401-410, 2003.
[26] Rochet, F. Simulation Réaliste de Ruisseaux en Temps Réel. Masters thesis, Université Joseph Fourier, France, 2005.
[27] Schlick, C. An Inexpensive BRDF Model for Physically-Based Rendering. Computer Graphics Forum, Vol. 13, No. 3, pp. 233-246, 531, 1994.
[28] Schlicke, T. Breaking Waves and the Dispersion

of Surface Films. PhD thesis, University of Edinburgh, 2001.
[29] Schneider, J., and Westermann, R. Towards Real-Time Visual Simulation of Water Surfaces. In Proceedings of the Vision Modeling and Visualization Conference, pp. 211-218, 2001.
[30] Stam, J. Stable Fluids. In Proceedings of ACM SIGGRAPH 1999, pp. 121-128, 1999.
[31] Sun, Y., Fracchia, F. D., and Drew, M. S. Rendering the Phenomena of Volume Absorption in Homogeneous Transparent Materials. In Proceedings of IASTED International Conference on Computer Graphics and Imaging, pp. 283-288, 1999.
[32] Takahashi, T., Fujii, H., Kunimatsu, A., Hiwada, K., Saito, T., Tanaka, K., and Ueki, H. Realistic Animation of Fluid with Splash and Foam. Computer Graphics Forum, Vol. 22, No. 3, pp. 391-400, 2003.
[33] Thon, S., and Ghazanfarpour, D. A Semi-Physical Model of Running Water. In Eurographics UK 2001 Conference Proceedings, pp. 53-59, 2001.
[34] Thon, S., and Ghazanfarpour, D. Real-Time Animation of Running Waters Based on Spectral Analysis of Navier-Stokes Equations. Computer Graphics International, pp. 333-346, 2002.
[35] Weber, M. J. Handbook of Optical Materials. CRC Press, 2002.
[36] Wu, E., Liu, Y., and Liu, X., An Improved Study of Real-Time Fluid Simulation on GPU. Journal of Computer Animation and Virtual World, Vol. 15, No. 3-4, pp. 139-146, 2004.

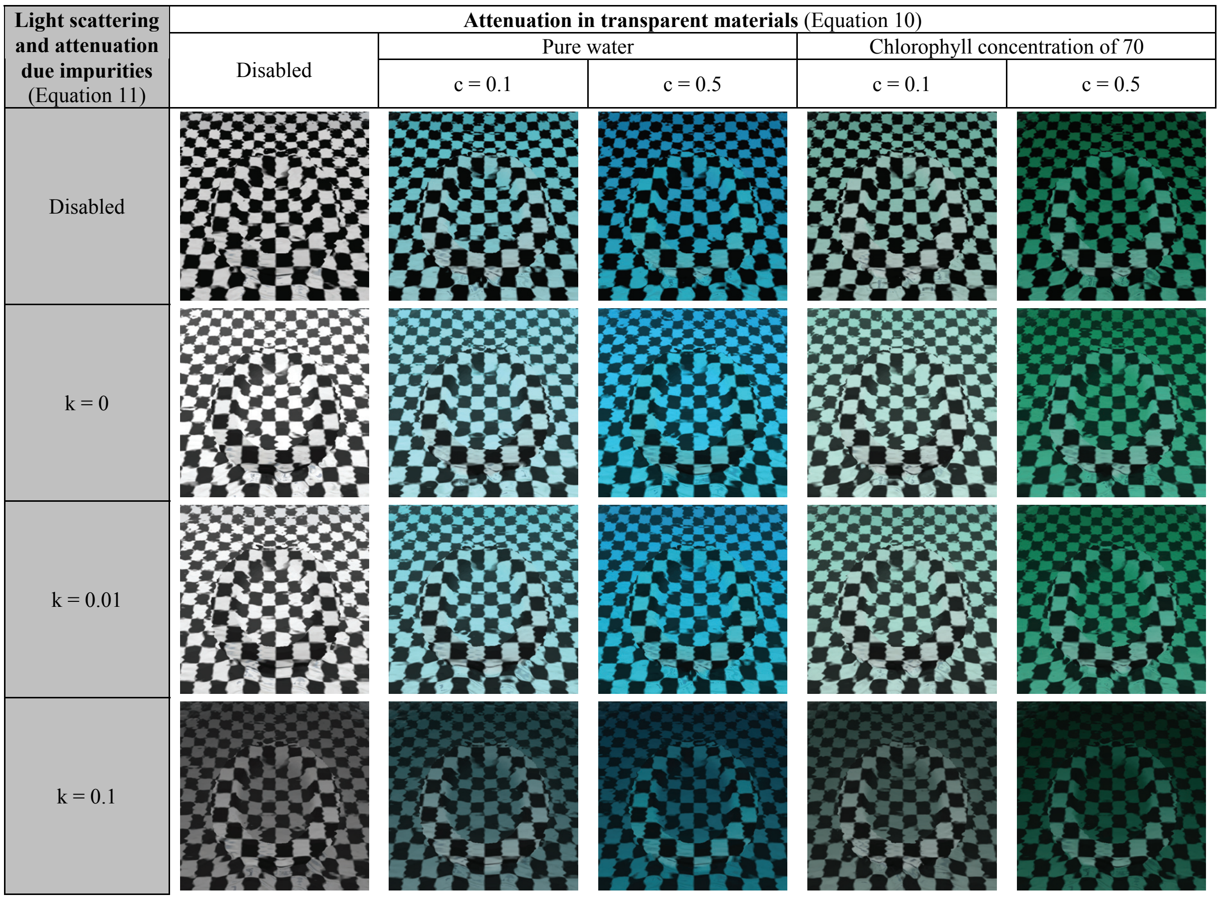
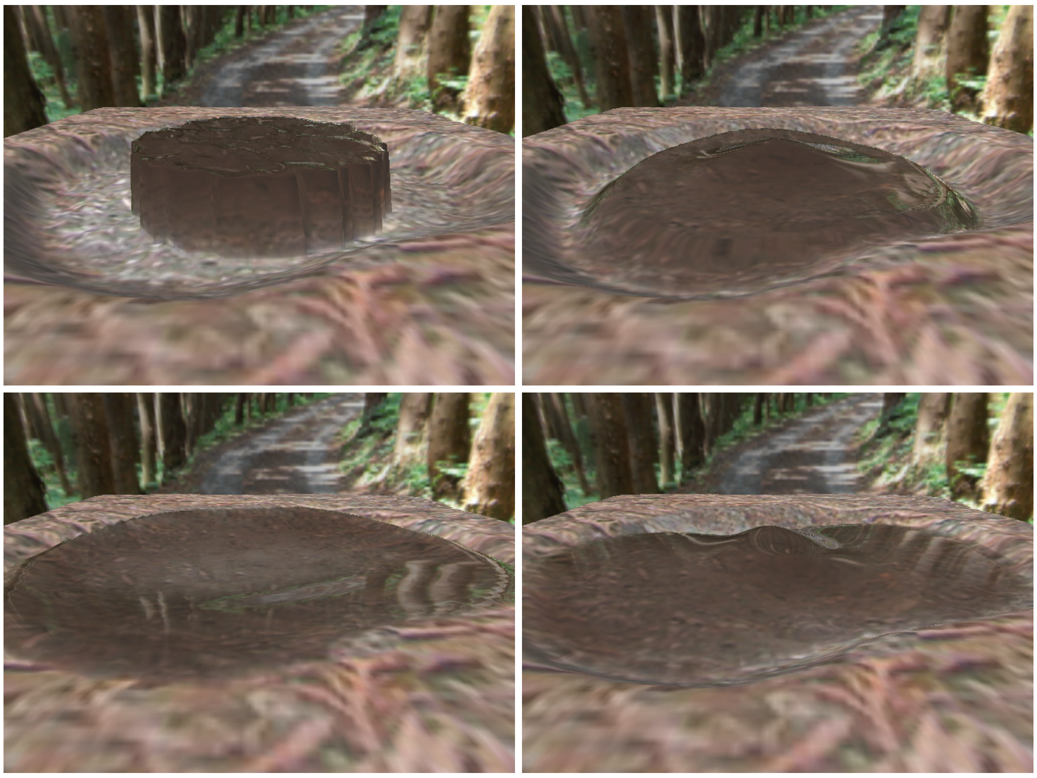
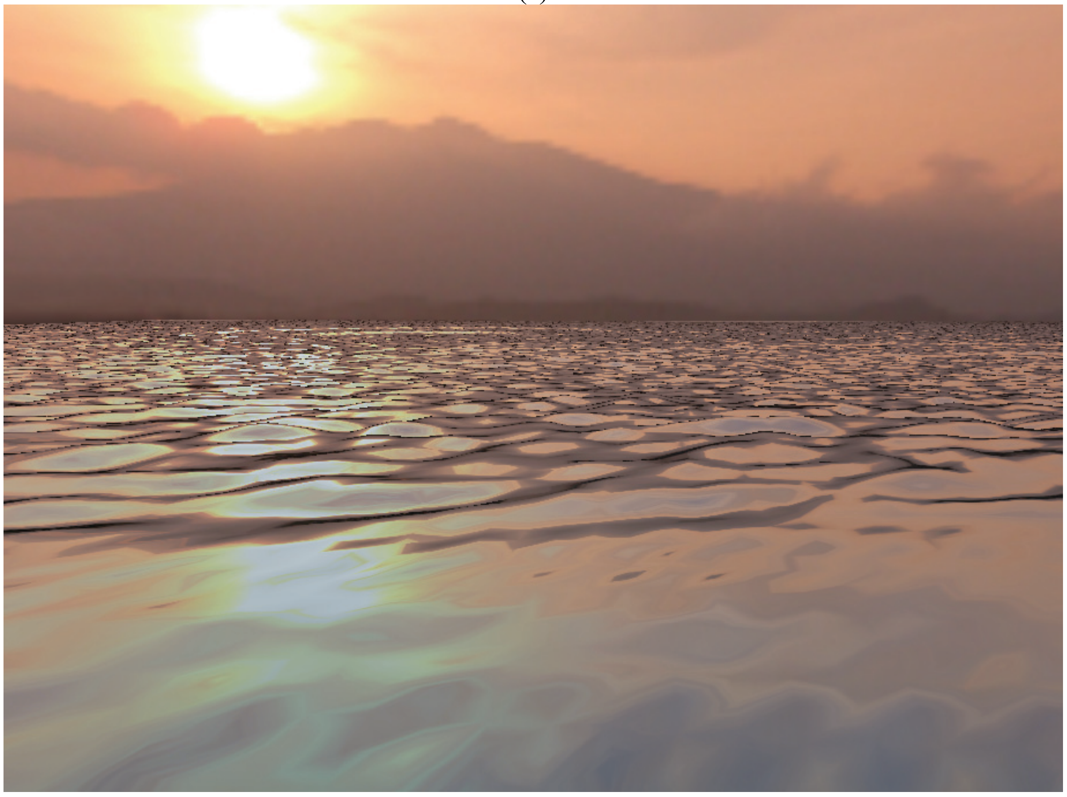

Light scattering and attenuation due impurities (Equation 11)
Attenuation in transparent materials (Equation 10)
Disabled
Pure water
Chlorophyll concentration of 70
c = 0.1
c = 0.5
c = 0.1
c = 0.5
Disabled
k = 0
k = 0.01
k = 0.1

Figure 9. Light attenuation and scattering: 2m-height relief of the character "O" in the middle, 7m of water at the top, 0.5m of water at the bottom.



(a)



(b)

Figure 10. Selected animation frames: (a) water collapsing (dimensions: 1.5m^2×0.15m, height-field resolution: 64×64); (b) lake ripples (dimensions: 200m^2×50m, height-field resolution: 512×512).