Real-time Rendering of Point Based Water Surfaces

Kei Iwasaki[1], Yoshinori Dobashi[2], Fujiichi Yoshimoto[1], and Tomoyuki Nishita[3]

[1] Wakayama University
[2] Hokkaido University
[3] The University of Tokyo

Abstract. In recent years, attention has been paid to particle-based fluid simulation, with several methods being developed to incorporate particle-based simulation into CG animations. These methods reconstruct water surfaces that are usually represented by polygons. However, the computational cost of the surface reconstruction is quite high. Therefore, it is difficult to render the result of the particle-based simulation at interactive frame rates. To address this, we present a real-time method for rendering water surfaces resulting from particle-based simulation. We present an efficient GPU accelerated surface reconstruction method from particles, sampling the water surface point by point. In addition to rendering the point based water surfaces, the use of the GPU permits efficient simulation of optical effects such as refraction, reflection, and caustics.

1 Introduction

The research into fluid simulation is one of the most important research topics in computer graphics. Many methods have been developed for the simulation of fluids such as water, smoke, and fire [1, 2, 3, 4]. Most of these methods subdivide the simulation space into grids and solve the Navier-Stokes equations by discretizing the equations, using the grids to simulate the fluid dynamics. These methods are based on the Eulerian method. On the other hand, particle-based fluid simulations have been developed that represent the fluid as particles and calculate the fluid dynamics by solving the particles dynamics [5]. Particle-based fluid simulation has received attention since this simulation method is free from the numerical diffusions in the convection terms, suffered by the Eulerian method, and the surface transformation is easy to handle.

One of the methods of visualizing particle-based simulation is to reconstruct the water surface by polygons and to render these polygons. The water surface is reconstructed as follows. Initially, a density function, (or smoothing kernel), is defined with the distance from the center of the particle as parameter. The simulation space is subdivided into a grid and the summation of the densities of the particles is calculated at each grid point. Then the water surface is extracted as an iso-surface by using either the marching cube [6] or the level set method [3, 7]. To render high quality images of the water surfaces, the simulation space must be subdivided into numerous small cells. This indicates that the computational cost of the density calculation at each cell also increases and thus the cost of the reconstruction of the water surface becomes quite high. Moreover, many small polygons are generated from a fully subdivided grid. For the animation of the particle-based fluid simulation, the processing of enormous numbers of small polygons compared to the number of screen pixels in each frame results in bandwidth bottlenecks. Therefore, these problems prevent the particle-based fluid simulation from being applied to interactive applications such as the preview of the simulation, video games and virtual reality.

In recent years, point based rendering methods have been developed, using the points as primitives instead of the polygons [8, 9]. Several methods that are accelerated by the GPU have been presented [10, 11]. Moreover, a point based method has been developed for visualizing iso-surfaces [12]. This method demonstrates that the point based visualization method for iso-surfaces can obtain storage and rendering efficiency compared with standard polygon-based methods.

Particle-based fluid simulation represents the fluid as particles and calculates the dynamics. Therefore, visualizing the particle-based fluid simulation by using point primitives is straightforward, since both of the result data of the simulation and the data from the rendering are unified into points.

This paper presents a fast rendering method, resulting in the particle-based fluid simulation without explicitly constructing polygons. In this paper, we deal with the water as a fluid and describe a rendering method for the water, represented by point primitives. To render the water surface, we have to take into account optical effects due to water surfaces such as reflection, refraction, and caustics. Rendering these optical effects is essential to increase realism. We present a fast rendering method for these effects from water surfaces, represented by points.

The contributions of our method are as follows.

– Fast generation of point primitives, representing water surfaces by using the GPU
– Fast rendering of the water surface, represented by points to obtain optical effects such as refraction, reflection, and caustics

The rest of our paper is organized as follows. Section 2 describes the related work. In Section 3, the overview of our method is presented. Section 4 describes the calculation of the density at each grid point by using the GPU. The method of rendering water surfaces, represented by points, is described in Section 5. The rendering results of point based fluid simulation are shown in Section 6. Finally, conclusions and future work are summarized in Section 7.

2 Previous Work

There have been many methods for visualizing the results of the fluid simulation. These are categorized into two types. One is to polygonize the iso-surfaces, represented by implicit functions, and then to render the polygons. Another is to directly render the implicit surface, without creating polygons. One of the methods to create the implicit surface using polygons is the marching cube method [6]. Many methods have employed this marching cube approach to render the water surface [13, 14, 4]. Moreover, a GPU accelerated iso-surface polygonization method has been proposed in recent years. Matsumura et al. proposed a fast method of iso-surface polygonization using programmable graphics hardware [15]. Reck et al. developed a hardware accelerated method to extract iso-surfaces from unstructured tetrahedral grids [16]. Although the marching cube method is efficient, representing iso-surfaces by creating polygons requires the memory for the connectivity information and two different data structures are required for points and polygons.

Another visualization method for fluid simulation of water involves the rendering of the iso-surface directly. Enright et al. [3] and Premoze et al. [7] employed a level set method to represent the water surface. Their methods render the water surface by using Monte Carlo path tracing methods. Whilst these methods can render realistic images, the computational cost for the rendering is high.

Although not for the rendering of the results of the fluid simulation, a visualization method has been developed for iso-surfaces using point primitives. Co et al. proposed a new algorithm called iso-splatting for rendering iso-surfaces using point primitives [12]. This method shows that the point based rendering of iso-surfaces can exceed the traditional polygon based approach such as a marching cube method in time and space efficiency. This method, however, does not describe the calculation method of the scalar(density) field, whose computational cost is high.

To solve these issues, we present a novel approach to render the water surface in a particle-based fluid simulation. In our method, the iso-surface, representing the water surface, is calculated efficiently by using fluid particles. Then the water surface is sampled, point by point, and rendered by surfels [8]. This makes it possible to unify the data structure into points in the simulation and then rendering, without the construction of polygons. Moreover, our method presents a fast rendering method for reflection, refraction, and caustics by use of the point sampled water surface.

3 Overview

Fig. 1 shows the overview of our method. This method deals with the results of the particle-based fluid simulation (Fig. 1(a)), calculated by particle-based simulation methods such as Moving Particle Semi-Implicit(MPS) and Smoothed Particle Hydrodynamics(SPH). Then the the water surfaces, including caustics, are rendered as shown in Fig. 1(d). To render the water surfaces including caustics, particles that represent the surfaces must be extracted. Directly rendering the particles representing surfaces is one solution to visualize the result of the particle-based fluid simulation. However, the number of particles used in the simulation is usually between about $1,000$ and $100,000$ so that the number of particles representing a surface is, at most, several ten thousands. As Muller pointed out, this is not sufficient number to render high quality images [14]. On the other hand, point based rendering methods [8, 9, 10, 11] are designed to render huge number of points measured by range scanners. Thus, it is difficult to create high quality images of water surfaces by rendering only the particles used in the simulation.

Therefore, our method generates dense sampled surfels (Fig. 1(c)), representing water surfaces from all the particles used in the simulation (Fig. 1(a)). We create a temporary grid in the simulation space, where the densities of the particles are accumulated in each grid point (Fig. 1(b)). The density at each grid point is calculated as a density function.

The cost of the density computation at each grid point is quite high, since it depends on the number of grid points and the number of particles. We present a fast method for accumulating densities of particles by using the GPU. Our density calculation method

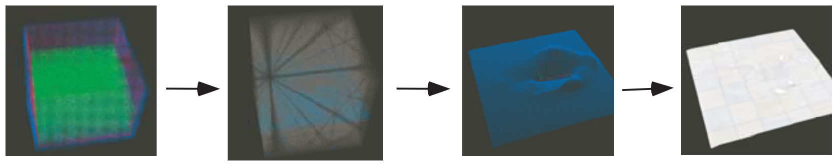

(a) result of particle based simulation
(b) calculate density
(c) extract surfels
(d) render water surface and caustics

Fig. 1. Overview of our method.

can be applied not only to the particle-based simulation, but also to the grid based simulation, since the marching cube method requires the density at each grid. The points (surfels) on the iso-surface representing the water surface are then extracted. The calculation of surfels on the water surface is explained in Section 4.

The water surfaces are rendered by splatting surfels (Fig. 1(d)). Refraction and reflection of light is calculated by using refraction and reflection mapping of surfels. The rendering method of caustics from water surfaces represented by surfels is described in Section 5.

4 Generation of Surfels of the Water Surface using a GPU

This section describes the method for generating dense surfels representing the water surface by using the particles. We create a grid in the simulation space and calculate the densities at each grid point by using particles. The simulation space is subdivided into $n_x \times n_y \times n_z$ grid points.

The density function $F(r,h)$ in this paper is calculated from the following equation [17].

$$F(r,h) = \begin{cases} \frac{405}{748\pi h}(-\frac{4}{9}a^6 + \frac{17}{9}a^4 - \frac{22}{9}a^2 + 1) & (0 \le r \le h), \\ 0 & (r > h), \end{cases} \tag{1}$$

where $a = r/h$, and where r is the distance from the center of particle to a calculation point, and h the effective radius of the particle. Although we have used this smoothing function as a density function for the prototype, other smoothing functions such as the smoothing kernel of the SPH could also be used as the density function.

The simulation space is located as shown in Fig. 2 and the z-axis is set to be the vertical direction. A virtual camera is set along the z-axis and the reference point of the virtual camera is set to be the center of the simulation space. A virtual screen is then set to be perpendicular to the z axis. The virtual screen consists of $n_x \times n_y$ pixels. Each pixel corresponds to a grid point on the grid planes perpendicular to the z axis, as shown in Fig. 2. The pixels in the screen frame buffer consist of R, G, B, and α components. To calculate the density of each grid point influenced by a particle, we use a metaball whose center is the position of the particle. The circle of intersection between the grid plane and the metaball, of effective radius is h, is calculated. The densities of pixels within the circle of intersection are calculated. By drawing the circles of intersection with the densities and accumulating the densities in the frame buffer, the density of

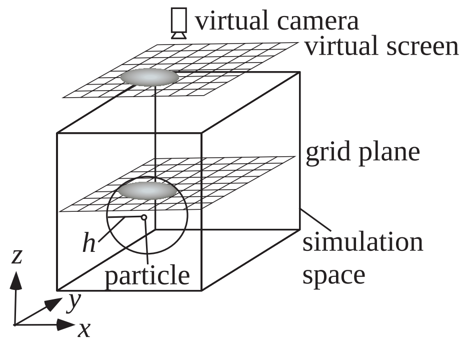


Fig. 2. Calculation of densities at each grid point by using splatting.

each pixel, corresponding to each grid point of the grid plane, is calculated by using the GPU.

The surfels on the water surface are generated using the following steps.

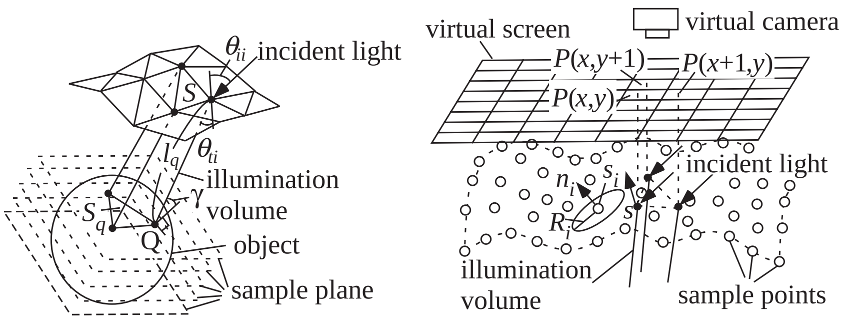
step1. Cluster the metaballs according to the z coordinate of the particle position.
step2. Project the metaballs in each cluster onto the screen and calculate the densities at each grid point.
step3. Generate the surfels on the iso-surface (water surface).

4.1 Clustering particles

As shown in Eq.(1), the density contribution from the particle at the grid point is zero, when the distance between the particle and the grid point is larger than the effective radius h. To reduce the computational time of the density calculation, the particles whose density contributions are zero are eliminated. The particles are classified into N_c clusters by using the z coordinates of the particles. Cluster C_k (k is the cluster number) includes the particles p^k whose z coordinate p_z^k satisfies $z_k \leq p_z^k < z_{k+1}$. Then the particles belonging to the cluster C_k are taken into consideration only for the calculation of the grid points of the grid planes whose z coordinate z_i satisfies $z_k - h \leq z_i < z_{k+1} + h$.

4.2 Density Calculation and Generating Surfels

To calculate the density at each grid point, texture-mapped disks are projected onto the screen corresponding to the grid planes (see Fig. 2). The disk corresponds to the circle of intersection between the grid plane and the metaball whose center is the particle and the effective radius of h. The texture mapped onto the disk represents the density function F on the disk. The densities on the disk are calculated from the distance from the center of the particle to the grid point using Eq. (1). By projecting the disks of the particles, intersecting the grid plane, onto the screen, and accumulating the densities, the densities of the grid points on each grid plane are calculated by using the GPU. In this calculation, the quantization error problem can occur since the density is quantized with 8bit precision in most graphics hardware. This problem causes the error of the



(a) illumination volume (b) virtual screen and camera

Fig. 3. Creation of illumination volume.

accumulated densities by using the GPU. To reduce the error, we use a floating point buffer for the precise calculation of accumulation of densities.

The disks are rendered by using point sprites. This makes it possible to accelerate the rendering process by the GPU. The point sprites are hardware functions that render a point by drawing a square, consisting of four vertices, instead of drawing a single vertex. The point sprites are automatically assigned texture coordinates for each vertex corner of the square. This indicates that each pixel inside the point sprite is automatically parameterized in the square. Therefore, the distance, d, from the center of the particle to each pixel of the point sprites can be calculated by using the fragment program. By comparing the distance, d, with the effective radius h, we can determine whether the pixel is within the circle of intersection or not. The density of the grid point corresponding to the pixel is calculated by inserting the distance, d, into the density function F. For the density calculation, we prepare a texture whose parameter is the distance from the calculation point to the center of the particle. The density of the pixel corresponding to the grid point is efficiently calculated by mapping this texture.

The density is scalar and the pixel of the frame buffer consists of four components. Therefore, our method calculates circles of intersection between the particle and four grid planes at once, and renders four disks by storing four densities in the RGB and α components. After drawing all the disks intersecting the four grid planes, the RGBα components are read from the frame buffer into the main memory. Then the points on the iso-surfaces for the four grid planes, corresponding to the RGBα components, are extracted. The density of the surface is specified by the user. We clear the frame buffer and draw all the disks intersecting the next four grid planes in the frame buffer. We repeat this for all grid planes that intersect the metaballs. Therefore, our method consumes texture memories only for the four grid planes.

The positions of the surfels, s_i, are set to the positions of these extracted points. The radius, R_i, of the surfel, s_i, is assigned and is determined so that there are no gaps between the surfels. Normal vector, n_i, of surfel s_i is calculated by using the gradient of the densities. If the distance between the extracted point and neighbor point is larger than a threshold, we add points on the iso-surface to fill gaps between the surfels.

5 Rendering Point Based Water Surface

This section describes the rendering method for water surfaces represented by surfels. To render the water surfaces, a disk is assigned to surfel s_i. The radius of the disk is R_i and the disk is perpendicular to normal n_i of the surfel. In this section, we first explain the rendering method of caustics due to water surfaces represented by surfels. Then the rendering method of water surfaces is described.

5.1 Rendering Caustics for Point Based Water Surface

Our rendering method for caustics is based on Nishita's and Iwasaki's methods [18, 19]. In these methods, the water surface is represented by a triangular mesh. At each vertex, the refracted direction of the incident light is calculated. Then the volumes are created by sweeping the vectors refracted from the triangle mesh. These volumes are called illumination volumes [18] (see Fig. 3). Caustics are rendered by accumulating the intensities of the areas of intersection between the object surface and the illumination volumes. The intensity, L_q, at point Q of the intersection area is calculated from the following equation,

$$L_q(\lambda) = L_i(\lambda)\cos\theta_{ii} T(\theta_{ii}, \theta_{ti}) \exp(-c(\lambda) l_q) F_q K(\lambda) + L_a(\lambda) \tag{2}$$

where λ is the wavelength, which is sampled for RGB components, $L_i(\lambda)\cos\theta_{ii}$ is the intensity of the incident light onto the water surface, $T(\theta_{ii}, \theta_{ti})$ is the Fresnel transmittance, $\exp(-c(\lambda)l_q)$ is the extinction of light from the water surface to point Q. F_q is the flux ratio and is calculated from the equation $F_q = S/S_q$, where S is the area of the triangle mesh of the water surface and S_q is the area of the intersection area between the illumination volume and the object (see Fig. 3). $K(\lambda)$ is the reflectance of the object surface and L_a is the intensity of the ambient light.

Our method is based on Iwasaki's method [19] that sets virtual planes (called sample planes) around the object and calculates the intensities of caustics on the object surface by using the intensities incident onto the sample planes. The intensities of the sample planes are calculated by accumulating the intersection triangles between the illumination volumes and the sample planes.

However, illumination volumes cannot be created, since the surfels representing water surfaces have no connectivity. To address this problem, we propose a method for creating illumination volumes from surfels.

5.2 Creation of illumination volumes

To create illumination volumes from the surfels, a virtual screen is set horizontally as shown in Fig. 3(b).The normal vector and the depth of a point that corresponds to each pixel, $P(x, y)$, of the frame buffer of the virtual screen are calculated by interpolating the normal and the depth values of the surfels representing the water surface.

The basic idea to create the illumination volumes is as follows. First, the depth of the water surface, $d_{(x,y)}$, from the virtual screen is calculated for each pixel, $P(x, y)$. Using the depth, points $s_{(x,y)}$ on the water surface are obtained. The normal vector, $n_{(x,y)}$,

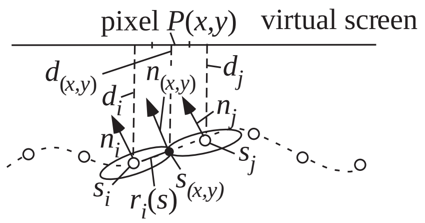


Fig. 4. Calculation of normal $n_{(x,y)}$ at point $s_{(x,y)}$ on the water surface, and depth $d_{(x,y)}$ from point $s_{(x,y)}$ to the virtual screen.

of $s_{(x,y)}$ is also calculated by interpolating the normals of the nearby surfels. Then a refraction vector at $s_{(x,y)}$ is computed. An illumination volume is created by sweeping the refracted vectors from points $s_{(x,y)}$, $s_{(x+1,y)}$, and $s_{(x,y+1)}$ (or $s_{(x+1,y+1)}$, $s_{(x+1,y)}$, and $s_{(x,y+1)}$) that correspond to neighboring pixels. In the following, the calculation method for the normal and the depth from the virtual camera is explained.

Normal, $n_{(x,y)}$, and depth, $d_{(x,y)}$, at point $s_{(x,y)}$ on the water surface are calculated from the following equations (see Fig. 4),

$$n_{(x,y)} = \frac{\sum_i g(\frac{r_i(s)}{R_i})n_i}{\sum_i g(\frac{r_i(s)}{R_i})}, d_{(x,y)} = \frac{\sum_i g(\frac{r_i(s)}{R_i})d_i}{\sum_i g(\frac{r_i(s)}{R_i})}, \qquad (3)$$

where g is a Gaussian function whose parameter is distance, $r_i(s)$, between each surfel, s_i and $s_{(x,y)}$, and returns 0 if $r_i(s)$ is larger than radius R_i.

The calculation of the normal, $n_{(x,y)}$, and depth, $d_{(x,y)}$, is accelerated by using the GPU. We create the normal map that stores the normal information at each point, $s_{(x,y)}$, corresponding to each pixel of the virtual screen. The normal map of the water surface is calculated by splatting the surfels. To create the normal map, calculated from Eq.(3), the xyz components of normal vector, n_i, of surfel, s_i, are encoded as RGB component of the color of surfel. Then the surfels are projected onto the screen with a radially decreasing Gaussian weight function $g(\frac{r_i(s)}{R_i})$. The value of Gaussian weight function $g(\frac{r_i(s)}{R_i})$ is stored in the α component of pixel, $P(x,y)$, and is used as a blending factor to calculate Eq.(3). We prepare a 1D texture for the Gaussian weight function. This texture is mapped onto each surfel and is multiplied by the RGB components of the surfel, corresponding to the normal vector of that surfel. The texture-mapped surfels are rendered and the resulting colors are accumulated in the frame buffer by using additive color blending functions. The resulting image is stored as a texture. Normal, $n_{(x,y)}$, at surface point, $s_{(x,y)}$, corresponding to pixel, $P(x,y)$, is calculated by dividing the RGB components of each pixel that store the summation of weighted color ($\sum_i g(\frac{r_i(s)}{R_i})n_i$), by the α component that stores the summation of weight ($\sum_i g(\frac{r_i(s))}{R_i})$). This per-pixel normalization is accelerated by using the fragment program of the GPU. The depth, $d_{(x,y)}$, is calculated in the same manner. The normal and depth information are read back to the main memory and the point, $s_{(x,y)}$, and refracted direction of the incident light onto $s_{(x,y)}$ are calculated. Illumination volumes are created by sweeping the refracted direction from each point, $s_{(x,y)}$.

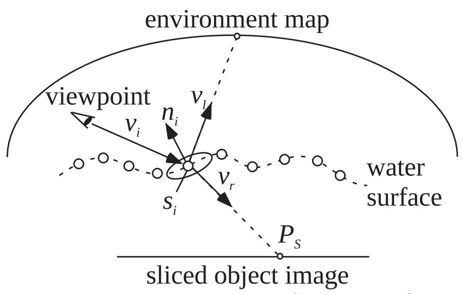


Fig. 5. Rendering water surfaces by surfels.

When the illumination volumes are created, caustics are rendered, using Iwasaki's method [19]. To render caustics due to water surfaces, sample planes are set around the objects within the water, and the intensities of caustics on the surface of the object are calculated by using the illumination distribution on the sample planes. In Iwasaki's method, an object is represented by a set of images of the object surface, that are created by rendering the object between two adjacent sample planes (see Fig. 3). These images are called the sliced object images [19]. The refracted object with caustics is rendered by refraction mapping of the sliced object images.

5.3 Rendering Water Surfaces Represented by Surfels

Water surfaces are rendered through the use of a splatting technique. Our rendering method extends the method proposed by Bostch et al. [10] to take into account refraction, reflection and caustics. The refraction of an object, with caustics through the water surface is rendered through the use of refraction mapping of sliced object images. The reflection of the environment is rendered through the use of reflection mapping. We use a three-pass rendering approach. Before rendering, we eliminate invisible surfels by using a backface-culling method. In a first pass, the surfels are rendered only to the z buffer with all z values having an ϵ offset added. ϵ is specified by the user.

In the second pass, the z-buffer update is turned off so that the overlapping surfels are blended if and only if the difference of their depth values is less than ϵ. For each visible surfel, the reflection vector v_l and the refraction vector v_r of the viewing ray are calculated (see Fig. 5). We calculate the intersection point P_S between the sliced object image and the refracted viewing ray from the surfel, and the texture coordinate of P_S for the sliced object image. The texture coordinate for the environment map of the surfel is also calculated. The surfel is rendered by mapping the sliced object image and the environment map texture onto the point sprite. At each pixel corresponding to the disk of the surfel, the Gaussian weight function is associated to blend the overlapping surfels. The RGB components of the pixel are multiplied by the Gaussian weight function by mapping the texture of the Gaussian weight onto the surfel.

In the third pass, the values of RGB components of each pixel, storing the accumulated weighted color must be normalized by dividing by the value of the alpha component that stores the accumulated Gaussian weights. The per-pixel normalization is performed by the method described in Section 5.2. This per-pixel normalization results in high quality images.

6 Results

Figs. 6 and 7 show the result of the MPS simulation rendered by our method. These figures are stills from an animation of dropping a parallelepiped into the water pool. The numbers of the points representing the water surface are from 61,500 to 77,000 in this animation. The average rendering time of these figures is about 0.039 sec (25.5fps). Our method can render the water surfaces, represented by points, including caustics, refraction, and reflection in real-time. These images are created on a desktop PC (CPU : Pentium4 3.4GHz, 2GB memory) with a nVidia GeForce6800 GT. The image size of these figures is 512×512. The size of the virtual screen for creating illumination volumes is 128×128.

The number of particles used for the simulation is about 210,000. The temporary grid is subdivided into 256^3. The computational time of density calculation from the particles using the GPU is 0.91 sec. The memory for calculating the density in the GPU is only 1MB. For the software calculation, the computational time is about 80 sec. That is, our method using the GPU can calculate the densities about 88 times faster than the method using the software. The computational time for extracting the surfels on the iso-surface is about 0.16 sec. Our GPU based method extremely reduces the time of reconstructing the water surface from the particles compared to the software based method. The relative difference between the densities calculated by the GPU and those by the software is about 1.9%. To verify the quantization error due to the GPU based density calculation, Fig. 8 shows the comparisons of the water surface that is extracted from the densities calculated by the GPU and that by the software. The image quality of the water surface (Fig. 8(a)) calculated by using the GPU based density calculation is indistinguishable from that by using the software based density calculation (Fig. 8(b)).

Fig. 9 shows the result of MPS simulation of making waves. These figures show that our method can extract complex shape of water surfaces. The average rendering frame rate of these figures is about 21.7fps. The numbers of the points representing the water surface are from 55,000 to 120,000 in this animation.

7 Conclusions and Future Work

In this paper, we have presented a fast rendering method for the particle-based simulation. To calculate the water surface from the result of the particle-based simulation, a temporary grid is created and the densities at each grid point by using the particles are calculated. We accelerate this density calculation by using a GPU based splatting method. Then the iso-surface is extracted and represented by surfels. The rendering method has been developed for a water surface, which is represented by the surfels. Moreover, our method can render the reflection, refraction, and caustics due to the point based water surface in real-time. Our method drastically reduces the time of the surface reconstruction and rendering. This makes it possible to easily preview the result of the particle-based simulation.

In future work, we plan to develop a method for improving the quality of images of the water surface by adaptively adding surfels, i.e., we need to add the surfels adaptively according to the intensity distribution at the water surface since the number of the surfels is sometimes insufficient when the light intensities drastically change at the water surface. Moreover, we would like to develop a method for rendering splashes and foams using surfels.

Acknowledgements

We thank Prometech Software Inc. for providing MPS simulation data and thank Kaori Ono and Yoshitaka Moro for their help.

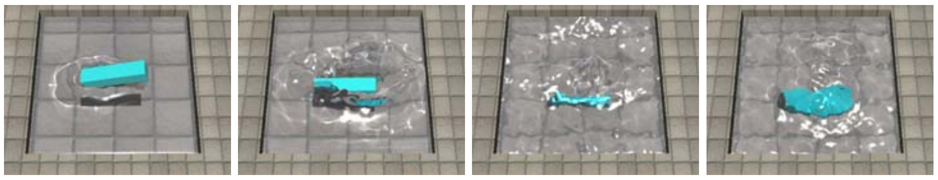


Fig. 6. Rendering the result of the MPS simulation.

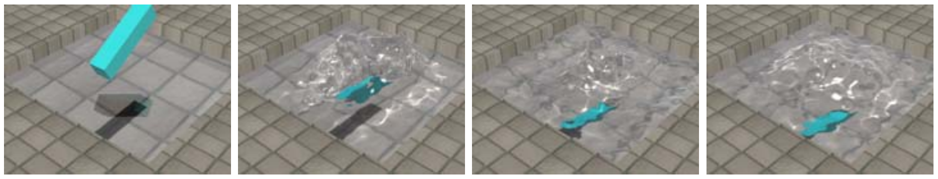
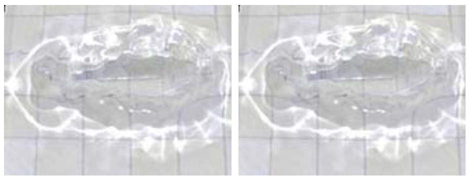


Fig. 7. Rendering the result of the MPS simulation viewed from another viewpoint.



(a) (b)

Fig. 8. Comparisons of the water surface generated by using the GPU based density calculation (a) and the water surface calculated by the software (b).

References

[1] Stam, J.: Stable fluids. In: Proc. SIGGRAPH'99. (1999) 121–128
[2] Foster, N., Fedkiw, R.: Practical animation of liquids. In: Proc. SIGGRAPH 2001. (2001) 23–30
[3] Enright, D., Marschner, S., Fedkiw, R.: Animation and rendering of complex water surfaces. In: Proc. SIGGRAPH 2002. (2002) 736–744
[4] Takahashi, T., Fujii, H., Kunimatsu, A., Hiwada, K., Saito, T., Tanaka, K., Ueki, H.: Realistic animation of fluid with splash and foam. Computer Graphics Forum **22**(3) (2003) 391–400
[5] Koshizuka, S., Tamako, H., Oka, Y.: A particle method for incompressible viscous flow with fluid fragmentation. Computational Fluid Dynamics Journal **29**(4) (1995) 29–46

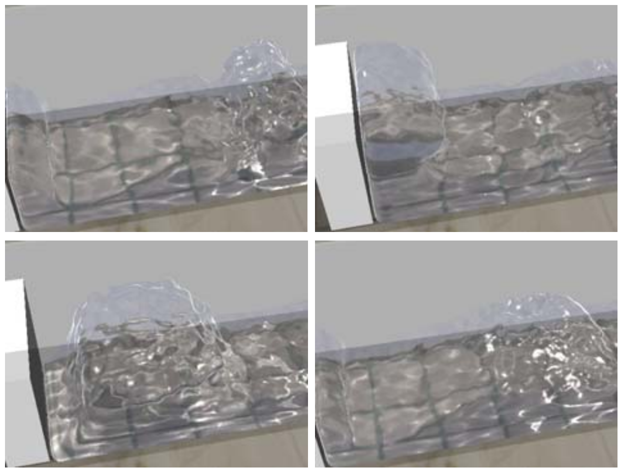


Fig. 9. Rendering the result of the MPS simulation of making waves.

[6] Lorensen, W., Cline, H.: Marching cubes: A high resolution 3D surface construction algorithm. In: Proc. SIGGRAPH'87. (1987) 163–169

[7] Premoze, S., Tasdizen, T., Bigler, J., Lefohn, A., Whitaker, R.: Particle based simulation of fluids. Computer Graphics Forum **22**(3) (2003) 335–343

[8] Pfister, H., Zwicker, M., Baar, J., Gross, M.: Surfels: Surface elements as rendering primitives. In: Proc.SIGGRAPH 2000. (2000) 335–342

[9] Zwicker, M., Pfister, H., Baar, J., Gross, M.: Surface splatting. In: Proc. SIGGRAPH 2001. (2001) 371–378

[10] Bostch, M., Kobbelt, L.: High-quality point-based rendering on modern GPUs. In: Proc. Pacific Graphics 2003. (2003) 335–343

[11] Guennebaud, G., L.Barthe, M.Paulin: Deferred splatting. Computer Graphics Forum **23**(3) (2004)

[12] Co, C., Hamann, B., Joy, K.: Iso-splatting: A point-based alternative isosurface visualization. In: Proc. Pacific Graphics 2003. (2003) 325–334

[13] Kunimatsu, A., Watanabe, Y., Fujii, H., Saito, T., Hiwada, K., Takahashi, T., Ueki, H.: Fast simulation and rendering techniques for fluid objects. Computer Graphics Forum **20**(3) (2001) 57–66

[14] Muller, M., Charypar, D., Gross, M.: Particle-based fluid simulation for interactive applications. In: Proc. Symposium on Computer Animation 2003. (2003) 154–159

[15] Matsumura, M., Anjo, K.: Accelerated isosurface polygonization for dynamic volume data using programmable graphics hardware. In: Proc. Electronic Imaging2003. (2003) 145–152

[16] Reck, F., Dachsbacher, C., Grosso, R., Greiner, G., Stamminger, M.: Realtime isosurface extraction with graphics hardware. In: Proc. Eurographics 2004 Short Presentation. (2004)

[17] Wyvill, G., Trotman, A.: Ray-tracing soft objects. In: Proc. Computer Graphics International. (1990) 439–475

[18] Nishita, T., Nakamae, E.: Method of displaying optical effects within water using accumulation-buffer. In: Proc. SIGGRAPH'94. (1994) 373–380

[19] Iwasaki, K., Dobashi, Y., Nishita, T.: A fast rendering method for refractive and reflective caustics due to water surfaces. Computer Graphics Forum **22**(3) (2003) 601–609