The Visual Computer manuscript No.
(will be inserted by the editor)

Kei Iwasaki · Yoshinori Dobashi · Fujiichi Yoshimoto · Tomoyuki Nishita

GPU-based rendering of point-sampled water surfaces

Abstract Particle-based simulations are widely used to simulate fluids. We present a real-time rendering method for the results of particle-based simulations of water. Traditional approaches to visualize the results of particle-based simulations construct water surfaces that are usually represented by polygons. To construct the water surfaces from the results of particle-based simulations, a density function is assigned to each particle and a density field is computed by accumulating the values of the density functions of all particles. However, the computation of the density field is time-consuming. To address this problem, we propose an efficient calculation of density field using a graphics processing unit (GPU). We presents a rendering method for water surfaces sampled by points. The use of the GPU permits efficient simulation of optical effects such as refraction, reflection, and caustics.

Keywords Real-time rendering · GPU · point-sampled geometry · 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 [18,5,4,19]. 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 [8,20]. 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 construct the water surface by polygons and to render these polygons. The water surface is constructed 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 [10] or the level set method [4, 15]. 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 construction 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 [14,22]. Several methods that are acceler-

K. Iwasaki
Department of Computer and Communication Sciences,
Wakayama University
E-mail: iwasaki@sys.wakayama-u.ac.jp

Y. Dobashi
Graduate School of Information Science and Technology,
Hokkaido University
E-mail: doba@nis-ei.eng.hokudai.ac.jp

F. Yoshimoto
Department of Computer and Commnucation Sciences,
Wakayama University
E-mail: fuji@sys.wakayama-u.ac.jp

T. Nishita
Graduate School of Frontier Sciences,
The University of Tokyo
E-mail: nis@is.s.u-tokyo.ac.jp

ated by the GPU have been presented [2,6]. Moreover, a point based method has been developed for visualizing iso-surfaces [3]. 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 Related 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 [10]. Many methods have employed this marching cube approach to render the water surface [9,12,19]. 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 [11]. 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.

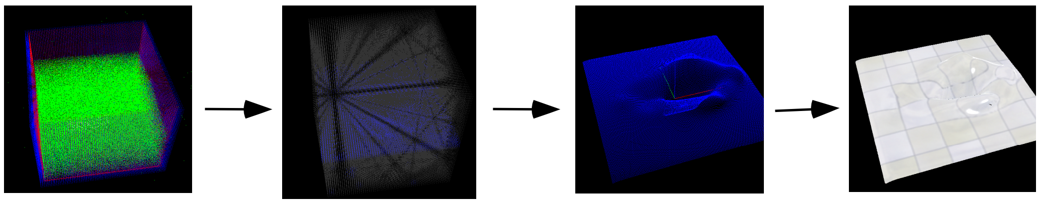
Another visualization method for fluid simulation of water involves the rendering of the iso-surface directly. Enright et al. [4] and Premoze et al. [15] employed a level set method to represent the water surface. Their methods render the water surface by using Monte Carlo path tracing methods. Whilst 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 [3]. 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 [14]. 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 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 [12]. On the other hand, point based rendering methods [14,22,2,6] are designed to render huge number of points measured by range scanners. Thus, it



(a) result of particle based simulation (b) calculate density field (c) extract iso-surface and sample by point (d) render water surface and caustics

Fig. 1 Overview of our method. (a) Our method renders water surfaces from the results of the particle-based simulation. (b) We first assign the density function to each particle and calculate the density field. (c) Then the iso-surfaces that represent the water surfaces are extracted and sampled by points. (d) Our method renders water surfaces represented by points and renders caustics taking into account refractions.

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 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 per pixel. The rendering method of caustics from water surfaces represented by surfels is described in Section 5.

4 Fast Density Calculation using the GPU

This section describes the calculation method of the density field from particles used in the fluid simulations. We create a grid in the simulation space and calculate the density 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 [21].

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

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 disk of intersection between the grid plane and the metaball, of effective radius is h, is calculated. The densities of pixels within the disk of intersection are calculated. By drawing the disks of intersection with the densities and accumulating the densities in the frame buffer, the density of each pixel, corresponding to each grid point of the grid plane, is calculated by using the GPU.

4.1 Density Calculation

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 intersection disk 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. For

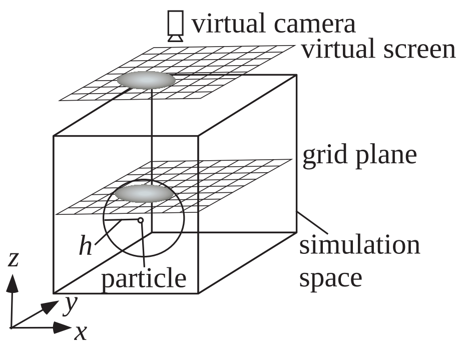


Fig. 2 Calculation of densities at each grid point by using splatting.

the computation accuracy in the accumulation of densities, our method uses floating-point buffers[1].

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 disk 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 object consists of four components. Therefore, our method calculates disks 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 object into the main memory.

4.2 Acceleration of Density Calculation using Clustering

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 clusters by using the z coordinates of the particles. Let z_{i_1}, z_{i_2}, z_{i_3}, and z_{i_4} be the z coordinates of four successive grid planes i_1, i_2, i_3, and i_4. Cluster C_i (i is the cluster number) includes the particles p^i whose z coordinate p^i_z satisfies $z_{i_1} - h \leq p^i_z \leq z_{i_4} + h$. To compute the densities on four grid planes i_1, i_2, i_3, and i_4, particles in the cluster C_i are projected.

[1] Our method uses framebuffer objects as floating point buffers.

4.3 Generation of Surfels

After the density field in the simulation space is calculated, the iso-surfaces are extracted. The density of the iso-surface is specified by the user. The iso-surfaces are sampled by points. The positions of the surfels, s_i, are set to the positions of these sample points. Normal vector, n_i, of surfel s_i is calculated by using the gradient of the densities. 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. The radius, R_i, of the surfel, s_i, is assigned and is determined so that there are no gaps between the surfels. If the distance between the sampled 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-Sampled Water Surface

This section describes the rendering method for water surfaces represented by surfels. 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-Sampled Water Surface

Our rendering method for caustics is based on Nishita's method [13] and Iwasaki's method [7]. 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 [13]. The intersection triangles between illumination volumes and the object surfaces are called caustics triangles. However, illumination volumes and caustics triangles cannot be created directly from point-sampled water surfaces, since the surfels representing water surfaces have no connectivity, To address this problem, we propose a rendering method of caustics triangles for water surfaces represented by surfels. Moreover, our rendering method are fully implemented on the GPU, whereas the previous

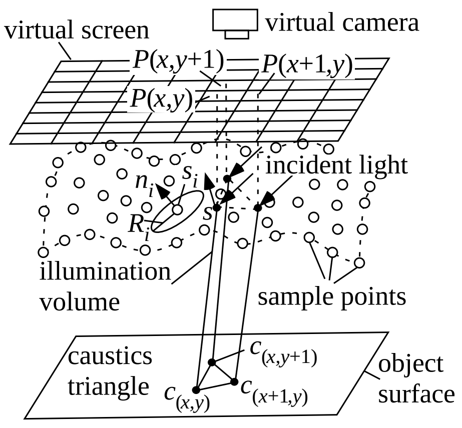


Fig. 3 Virtual screen for rendering caustics.

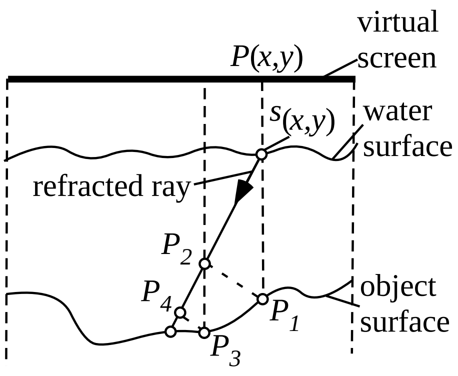


Fig. 4 Calculation of the vertex of the caustics triangle.

method [7] calculated illumination volumes and intensities of caustics triangles on the CPU.

To render caustics for point-sampled water surfaces, we set a virtual screen horizontally as shown in Fig. 3. Then point $s_{(x,y)}$ on the water surface corresponding to pixel $P(x,y)$ of the virtual screen, and the refracted ray of the incident light at $s_{(x,y)}$ are calculated. 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. The vertex $c_{(x,y)}$ of the caustics triangle corresponding to pixel $P(x,y)$ is obtained by calculating the intersection point between the refracted ray and the object surface. By relating $s_{(x,y)}$ and $c_{(x,y)}$ to pixel $P(x,y)$, $s_{(x,y)}$ and $c_{(x,y)}$ can be calculated on the fragment program.

The rendering algorithm of caustics on the GPU is follows:

1. calculate normal and depth of each point corresponding to each pixel to obtain the point and the refracted ray.
2. calcualte the intersection point (the vertex of the caustics triangle) between the refracted ray and the object surface.
3. calculate intensity at the vertex of the caustics triangle.
4. render caustics triangles.

5.1.1 Calculation of points on the water surface and normals

The position of $s_{(x,y)}$ is calculated by using the depth $d_{(x,y)}$ of the water surface from the virtual screen. Normal, $n_{(x,y)}$, and depth, $d_{(x,y)}$, at point $s_{(x,y)}$ on the water surface are calculated from the following equations,

$$n_{(x,y)} = \frac{\sum_i g(\frac{r_i(s)}{R_i})n_i}{\sum_i g(\frac{r_i(s)}{R_i})}, d_{(x,y)} = \frac{\sum_i g(\frac{r_i(s)}{R_i})d_i}{\sum_i g(\frac{r_i(s)}{R_i})}, \quad (2)$$

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 as the previous method [1]. The normal $n_{(x,y)}$ and the depth $d_{(x,y)}$ are stored as two textures, normal map and depth map, respectively.

5.1.2 Calculation of vertices of caustics triangles

By using the depth map, the position of $s_{(x,y)}$ on the water surface is calculated on the fragment program. The refracted vector at $s_{(x,y)}$ is calculated by using the normal $n_{(x,y)}$. The vertex $c_{(x,y)}$ of the caustics triangle is obtained by the intersection calculation between the refracted ray from $s_{(x,y)}$ and the object surface. The vertex $c_{(x,y)}$ is iteratively calculated on the fragment program, which is similar to [17].

Let us explain the algorithm of the intersection calculation using Fig. 4. We first calculate the point on the object surface corresponding to each pixel $P(x,y)$ by rendering the object surface to a texture. We call the texture, geometry map. Then we calculate the intersection point iteratively as follows. First, the point P_1 on the object surface corresponding to $P(x,y)$ is calculated by referring to the geometry map. Then the distance between $s_{(x,y)}$ and P_1 is calculated. If the distance is smaller than a threshold specified by the user, point P_1 is regarded as the intersection point. Otherwise P_1 is projected onto the vector of the refracted ray at $s_{(x,y)}$. The projected point is referred to as P_2. Then the point P_3 on the object surface corresponding to P_2 is obtained by using the geometry map. The distance P_2P_3 is calculated to determine whether P_2 is regarded as the intersection point. Our method repeats the above processes until the distance between the projected point on the refracted ray and the corresponding point on the object surface becomes smaller than the threshold.

The vertices of the caustics triangles are stored as a texture to calculate the intensities of the caustics triangles. The texture is referred to as intersection map. Moreover, the vertices of the caustics triangles are required to render caustics triangles. To do this, the vertices of the caustics triangles are rendered to a vertex buffer object that makes the vertices of the caustics triangles be stored in the video memory.

5.1.3 Calculation of intensities of caustics triangles

The intensity L_c at $c_{(x,y)}$ of the caustics triangle is calculated from the following equation [13]:

$$L_c = L_i \cos\theta_i T(\theta_i, \theta_t) \exp(-\sigma_t l_c) F_c f_r + L_a, \qquad (3)$$

where $L_i \cos\theta_i$ is the intensity of the incident light onto the water surface, θ_i is the incident angle between the normal $n_{(x,y)}$ and the incident light, $T(\theta_i, \theta_t)$ is the Fresnel transmittance, σ_t is the extinction coefficient of the light in the water, l_c is the length between $s_{(x,y)}$ and $c_{(x,y)}$. F_c is the flux ratio and is calculated from $F_c = S/S_c$, where S is the area of the triangle consisting of the three points on the water surface corresponding to three pixels of the virtual screen, and S_c is the area of the caustics triangle. f_r is the diffuse reflectance of the object surface and L_a is the intensity of the ambient light. To calculate the flux ratio, the area of the caustics triangle must be calculated. The area of the caustics triangle is calculated by using the intersection map that stores the vertices of the caustics triangles. The intensity L_c is stored in the video memory[2].

5.1.4 Rendering of caustics using caustics triangles

Caustics are rendered by drawing caustics triangles and accumulating the intensities of the caustics triangles. Since the vertices of the caustics triangles and the intensities of the vertices are stored in the GPU, caustics triangles are rendered efficiently.

5.2 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 Botsch et al. [1] to take into account refraction, reflection and caustics.

Before rendering, we eliminate invisible surfels by using a backface-culling method. First, the surfels are rendered only to the z-buffer with all z values having an ϵ offset added and the update of the z-buffer is enabled. ϵ is specified by the user.

Then, the update of the z-buffer is turned off so that the overlapping surfels are blended if and only if the difference of their depth values is less than ϵ. The position of the water surface and its normal corresponding to each pixel of the frame buffer is calculated by accumulating the Gaussian weighted normals and depths as described in Sec. 5.1.1. By using the position and the normal at each pixel, the reflection and refraction rays of the viewing ray are calculated per each pixel. The images of the object surface and caustics viewed from the viewpoint without the water surface are rendered to textures. Then the intersection point between the refracted ray and the object surface is calculated by using the method described in Sec. 5.1.2. The refraction of an object, with caustics through the water surface is rendered by refraction mapping of the images of the object and caustics.

[2] In our implementation, the intensity is stored in the vertex buffer object. To render the intensity to the vertex buffer object, our method uses a pixel buffer object.

6 Results

Fig. 5 shows the result of MPS simulation of making waves. The numbers of the points representing the water surface are from 55,000 to 120,000 in this animation. The average rendering frame rate of these figures is about 43fps. That is, our method can render the water surfaces, represented by points, including caustics, refraction, and reflection in real-time. These images are created on a laptop PC (CPU : Centrino Duo 2.16GHz, 2GB memory) with a NVIDIA GeForce Go 7900 GTX. The image size of these figures is 512×512. The size of the virtual screen for creating illumination volumes is 256×64.

The number of water particles used for the simulation is about 150,000. The temporary grid is subdivided into 256^3. The computational time of density calculation from the particles using the GPU is 0.14 sec. The memory for calculating the density in the GPU is only 1MB. For the software calculation, the computational time is about 8.4 sec. That is, our method using the GPU can calculate the densities about 60 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 constructing 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. 6 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. 6(a)) calculated by using the GPU based density calculation is indistinguishable from that by using the software based density calculation (Fig. 6(b)).

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-sampled water surface in real-time. Our method drastically reduces the times of the surface construction and rendering. This makes it possible to easily preview the result of the particle-based simulation.

In future work, we plan to accelerate the extraction of surfels by using the GPU. Moreover, we would like to develop a method for rendering splashes and foams using surfels.

Acknowledgements This research was partly supported by the Okawa Foundation.

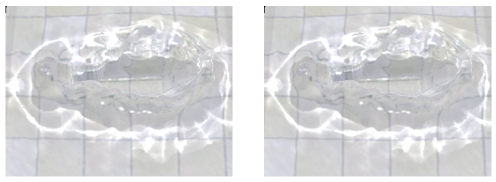


Fig. 6 Comparisons of the water surface generated by using the GPU based density calculation (a) and the water surface calculated by the software (b). These images are rendered from the particles of the simulation of dropping a parallelepiped into the water. The water surfaces with caustics viewed from above the water surfaces are rendered.

References

1. Botsch, M., Hornung, A., Zwicker, M., Kobbelt, L.: High-quality surface splatting on today's gpus. In: Proc. Eurographics Symposium on Point-Based Graphics 2005, pp. 17–24 (2005)
2. Botsch, M., Kobbelt, L.: High-quality point-based rendering on modern GPUs. In: Proc. Pacific Graphics 2003, pp. 335–343 (2003)
3. Co, C., Hamann, B., Joy, K.: Iso-splatting: A point-based alternative isosurface visualization. In: Proc. Pacific Graphics 2003, pp. 325–334 (2003)
4. Enright, D., Marschner, S., Fedkiw, R.: Animation and rendering of complex water surfaces. In: Proc. SIGGRAPH 2002, pp. 736–744 (2002)
5. Foster, N., Fedkiw, R.: Practical animation of liquids. In: Proc. SIGGRAPH 2001, pp. 23–30 (2001)
6. Guennebaud, G., L.Barthe, M.Paulin: Deferred splatting. Computer Graphics Forum **23**(3) (2004)
7. Iwasaki, K., Dobashi, Y., Nishita, T.: A fast rendering method for refractive and reflective caustics due to water surfaces. Computer Graphics Forum **22**(3), 601–609 (2003)
8. Koshizuka, S., Tamako, H., Oka, Y.: A particle method for incompressible viscous flow with fluid fragmentation. Computational Fluid Dynamics Journal **29**(4), 29–46 (1995)
9. 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), 57–66 (2001)
10. Lorensen, W., Cline, H.: Marching cubes: A high resolution 3D surface construction algorithm. In: Proc. SIGGRAPH'87, pp. 163–169 (1987)
11. Matsumura, M., Anjo, K.: Accelerated isosurface polygonization for dynamic volume data using programmable graphics hardware. In: Proc. Electronic Imaging2003, pp. 145–152 (2003)
12. Muller, M., Charypar, D., Gross, M.: Particle-based fluid simulation for interactive applications. In: Proc. Symposium on Computer Animation 2003, pp. 154–159 (2003)
13. Nishita, T., Nakamae, E.: Method of displaying optical effects within water using accumulation-buffer. In: Proc. SIGGRAPH'94, pp. 373–380 (1994)
14. Pfister, H., Zwicker, M., Baar, J., Gross, M.: Surfels: Surface elements as rendering primitives. In: Proc.SIGGRAPH 2000, pp. 335–342 (2000)
15. Premoze, S., Tasdizen, T., Bigler, J., Lefohn, A., Whitaker, R.: Particle based simulation of fluids. Computer Graphics Forum **22**(3), 335–343 (2003)
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. Stah, M., Konttinen, J., Pattanaik, S.: Caustics mapping: An image-space technique for real-time caustics. IEEE Transactions on Visualization and Computer Graphics **13**(2), 272–280 (2007)
18. Stam, J.: Stable fluids. In: Proc. SIGGRAPH'99, pp. 121–128 (1999)
19. 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), 391–400 (2003)
20. Thurey, N., Keiser, R., Pauly, M., Rude, U.: Detail-preserving fluid control. In: Proc. Symposium on Computer Animation 2006, pp. 7–12 (2006)
21. Wyvill, G., Trotman, A.: Ray-tracing soft objects. In: Proc. Computer Graphics International, pp. 439–475 (1990)
22. Zwicker, M., Pfister, H., Baar, J., Gross, M.: Surface splatting. In: Proc. SIGGRAPH 2001, pp. 371–378 (2001)

Kei Iwasaki received the B.S., M.S., and Ph.D. degrees from the University of Tokyo in 1999, 2001, and 2004, respectively. He is presently an associate professor in the Department of Computer and Communication Sciences, at Wakayama University, Wakayama, Japan. His research interests are mainly for computer graphics.

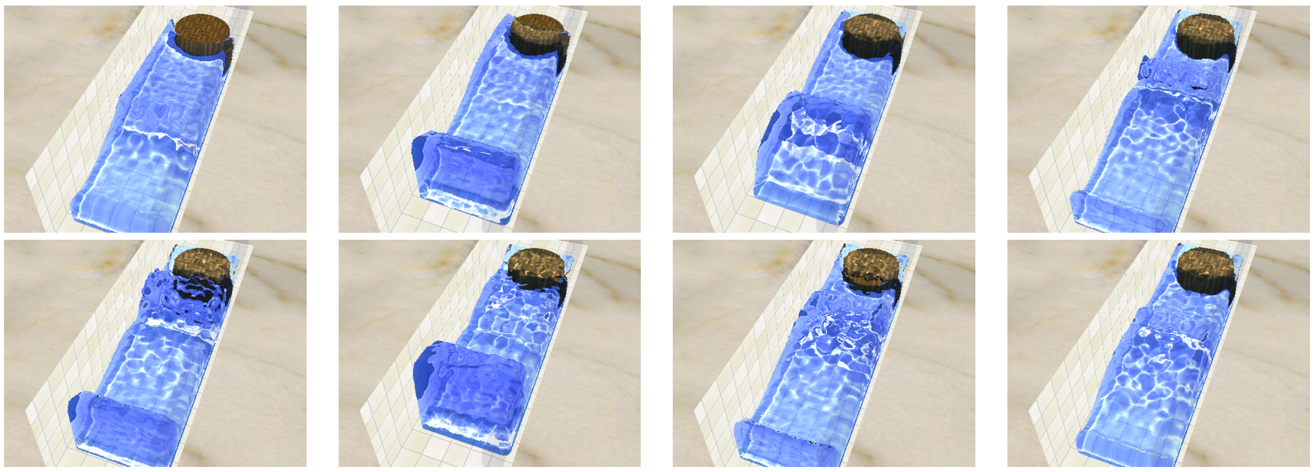
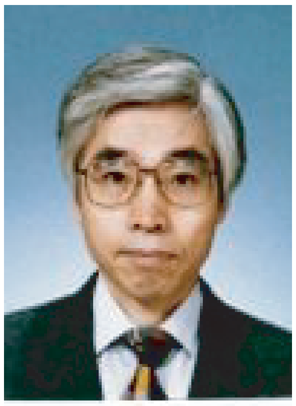


Fig. 5 Rendering the result of the MPS simulation of making waves.

Yoshinori Dobashi received the B.E., M.E., and Ph.D. in Engineering in 1992, 1994, and 1997, respectively, from Hiroshima University. He worked at Hiroshima City University from 1997 to 2000 as a research associate. He is presently an assistant professor at Hokkaido University in the graduate school of engineering, Japan since 2000. His research interests are computer graphics including lighting models.

Tomoyuki Nishita received the B.E., M.E., and Ph.D. degrees from Electrical Engineering from the Hiroshima University, Japan, in 1971, 1973, and 1985, respectively. He worked for Mazda Motor Corp. from 1973 to 1979. He has been a lecturer at the Fukuyama University since 1979, then became an associate professor in 1984, and later became a professor in 1990. He moved to the Department of Information Science of the University of Tokyo as a professor in 1998 and now is a professor at the Department of Complexity Science and Engineering of the University of Tokyo since 1999. His research interests are mainly for computer graphics.



Fujiichi Yoshimoto received the Ph.D. in computer science from Kyoto University, Japan in 1977. He is presently a professor in the Department of Computer and Communication Sciences, at Wakayama University, Wakayama, Japan. His research interests are in entertainment computing, computer graphics and CAD.