Fast, High-Quality Rendering of Liquids Generated Using Large-scale SPH Simulation

Xiangyun Xiao, Shuai Zhang, and Xubo Yang
Shanghai Jiao Tong University

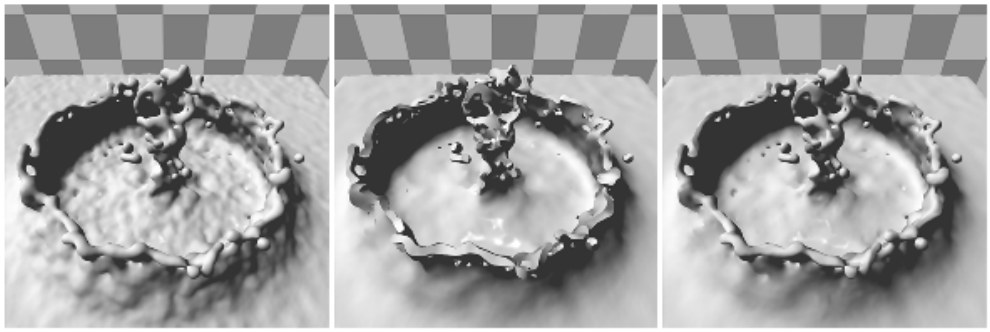

Figure 1. A comparison of three normal-estimation methods. The left image is the result of basic normal estimation, the middle image is the result of PCA, and the right image is the output of our method by blending the basic and PCA results. The basic normal estimation leads to an almost-correct result except for nonsmooth artifacts. The PCA result will fail when there are inadequate neighbor particles.

Abstract

Particle-based methods like smoothed particle hydrodynamics (SPH) are increasingly adopted for large-scale fluid simulation in interactive computer graphics. However, surface rendering for such dynamic particle sets is challenging: current methods either produce low-quality results, or they are time consuming. In this paper, we introduce a novel approach to render high-quality fluid surfaces in screen space. Our method combines the techniques of particle splatting, ray-casting, and surface-normal estimation. We apply particle splatting to accelerate the ray-casting process, estimating the surface normal using principal component analysis (PCA) and a GPU-based technique to further accelerate our method. Our method can produce high-quality smooth surfaces while preserving thin and sharp details of large-scale fluids. The computation and memory cost of our rendering step depends only on the image resolution. These advantages make our method very suitable for previewing or rendering hundreds of millions of particles interactively. We demonstrate the efficiency and effectiveness of our method by rendering various fluid scenarios with different-sized particle sets.

 ISSN 2331-7418

1. Introduction

Particle-based methods like smoothed particle hydrodynamics (SPH) [Desbrun and Gascuel 1996] for fluid simulation have received copious attention in the graphics community due to their flexibility and simplicity. Although particle-based fluid simulation has improved greatly, high-quality real-time surface rendering for large-scale fluids is still a very challenging problem for existing particle-rendering methods, significantly limiting those methods from being used in interactive applications.

Generally, in realistic liquid rendering, particle-based liquid surfaces can be represented by a level set based on an implicit function, extracted as polygonal meshes using marching cubes [Lorensen and Cline 1987] or marching tiles [Williams 2008]. Articles like [Müller et al. 2003; Zhu and Bridson 2005; Adams et al. 2007; Sin et al. 2009; Bhatacharya et al. 2011; Yu and Turk 2013] focus on the construction of implicit functions to generate smooth isosurfaces without blobby artifacts. However, due to the full-domain calculation and limited grid resolution, implicit surface polygonization methods are both time consuming and memory intensive, especially in large-scale scenes. Furthermore, they suffer from temporal discretization artifacts [Adams et al. 2006] due to the limited grid resolution.

Another class of rendering techniques which are often used for interactive applications, such as those described in [Adams et al. 2006; Müller et al. 2007; van der Laan et al. 2009; Imai et al. 2014; Imai et al. 2016], are based on particle splatting. These screen-space-based methods can render the particle sets in real time with less memory consumption and do not need any information about neighboring particles. However, these methods cannot deal with situations with noise and rarely deal with hundreds of millions of particles. Moreover, they always produce serrated edges with sphere shapes on liquid-free surfaces and significant blobby artifacts when the camera is near the fluid surface.

In this paper, we propose a novel rendering approach for particle-based fluids that can produce high-quality surface visualization in real time for large-scale scenes (see Figure 2). The key idea of our method is to merely reconstruct the surface which is visible using a ray-casting algorithm. Our method casts rays towards the liquid and then samples those rays. At the same time, we calculate the density-attribute values of those sampling points' neighbor particles step-by-step until they reach the defined iso-value. The entry points of those rays are determined by performing a particle-splatting step to generate an approximated surface. Thus, our method omits the empty space which does not contain any particles and enables the ray casting to work in a narrow band around the isosurface. To construct a smooth and detail-preserving surface, we calculate the normal directions by performing principal component analysis (PCA) among the neighbor particles around the isosurface instead of using the extracted isosurface directly. We design a GPU-based algorithm to speed up its efficiency significantly. Compared to previous related rendering methods, our

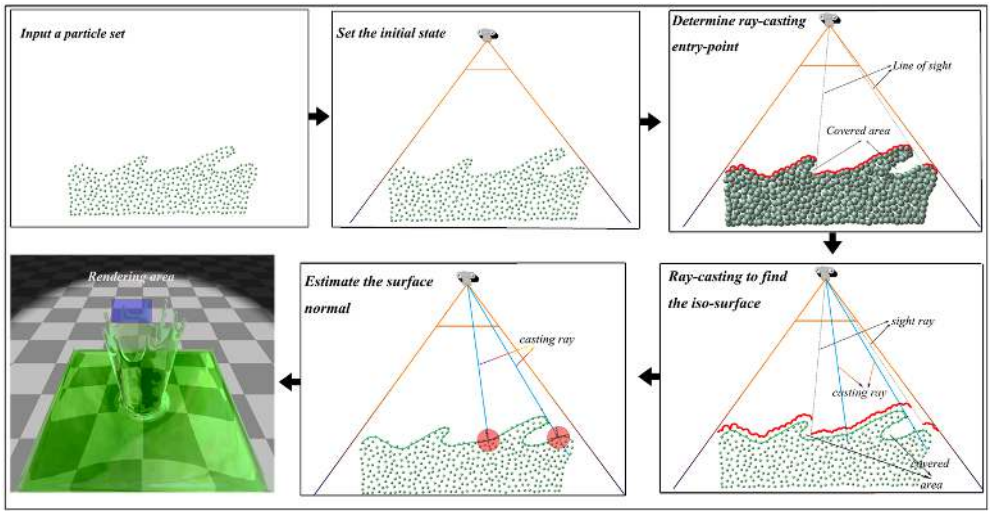

Figure 2. An overview of our surface-rendering algorithm. First step: input the position data; second step: set the initial state; third step: determine the ray-casting entry point by splatting particles; fourth step: ray casting to find the isosurface; fifth step: estimating the surface normal; finally, obtaining the rendering result.

approach produces high-quality rendering results and significantly saves computation and memory cost, especially for large-scale particle sets. Moreover, our method provides users a trade-off between speed and quality by adjusting the resolution of the view-port. Our method is straightforward and can be integrated with existing particle-based simulation schemes easily.

2. Related Work

In computer graphics, particle-based methods are increasingly used for simulating high-resolution fluids. While previous work like [Müller et al. 2003; Adams et al. 2007; Solenthaler and Pajarola 2009; Macklin and Müller 2013] have improved the simulation efficiency [Ihmsen et al. 2014], less work has focused on the high-quality surface rendering of large particle sets.

Traditionally, particle-based simulation of liquids were rendered by isosurface methods [Lorensen and Cline 1987]. In 1982, Blinn et al. introduced the earliest metaball approach [Blinn 1982], where the scalar field was constructed by summing up the values of radial basis functions. Müller et al. improved this technique by smoothing the surface [Müller et al. 2003]. Wald et al. [Wald and Seidel 2005] proposed a ray-tracing method to deal with the point-based models. Szecsi and Illes proposed a fast GPU-based metaball rendering approach [Szécsi and Illés 2012]; they used A-buffer techniques and fragmented linked lists. However, those methods always produce bumps and fail to yield flat surfaces even for a uniformly distributed

particle set. In 2007, Adams et al. [Adams et al. 2007] used a distance-based surface-tracking technique to generate smooth surfaces even for particles with different radii, and, later, Yu and Turk [Yu and Turk 2013] proposed an anisotropic kernel approach to further modify the extracted surfaces. Their method can produce impressive liquid surfaces but the time consumption is considerable.

Ray casting is widely used for volume rendering [Kruger and Westermann 2003]. It is, however, computationally expensive when applied directly over the whole particle set. A modification of this technique is to first resample particle quantities into a temporary uniform 3D grid in a simulation domain [Navratil et al. 2007] and then use traditional volume ray-casting algorithms [Drebin et al. 1988; Kruger and Westermann 2003] on the 3D grid to extract an isosurface. Fraedrich et al. [Fraedrich et al. 2010] adopted a perspective grid in view space and reduced the memory requirements compared to the uniform grid. In [Kanamori et al. 2008; Zhang et al. 2008], intersections of view-ray and particles were computed to generate the isosurface directly and thus avoided the memory consumption of a temporary grid. Ilya et al. [Rosenberg and Birdwell 2008] proposed a fast isosurface extraction technique by dividing the simulation domain into blocks and extracted isosurfaces within those blocks, which can achieve real-time performance with thousands of particles.

Surface splatting is a widely used technique for surface reconstruction [Zwicker et al. 2001]. In order to obtain a smooth surface representation based on surface splatting, Adams et al. [Adams et al. 2006] first determined depth values of the foremost particles and then blended the normals for overlapping particles. Later, Müller et al. constructed meshes in screen space which can accelerate the surface-splatting step [Müller et al. 2007]. In [Müller et al. 2007; van der Laan et al. 2009], the authors smoothed the depth buffer and calculated normals by neighbor pixel depth. In [Reichl et al. 2014], the rendering quality of existing surface-splatting methods cannot match the implicit surface polygonization methods although they can run in real time. Imai and colleagues also presented a real-time rendering method that can handle multiple refractions [Imai et al. 2014], but the method causes inaccurate refractions and suffers from the same problems as in the previous filtering-based approaches [Cords and Staadt 2009; Green 2010], that is, the resultant depth maps unnaturally warp towards the viewpoint around depth boundaries. Recently, Imai et al. presented a new screen-space method [Imai et al. 2016] to handle blobby artifacts in real time; their goal was very similar to ours, but our method can handle large-scale particle-based fluids with high quality in real time. See Figure 1 for a comparison of some of the methods.

Normal estimation from a particle set is a well-studied problem in the area of point-cloud processing. The most commonly used method is based on regression [Hoppe et al. 1992; Guennebaud and Gross 2007; Huang et al. 2009]. These techniques estimate normals with a tangent plane generated by performing PCA over neighbor particles. Due to the inherent low-pass filter property of regression mod-

```
Input: Particle position set P_{x,y,z}
  if p_{x_i,y_j,k_k} ∈ P_{x,y,z}
    Distribute p_{x_i,y_j,k_k} to GPU node G(p);
    for each GPU node
      Rendering spheres S_{p_{x_i,y_j,k_k}} ⇐ Splatting method + Point sprites;
      Depth buffers Dbuf{P_{x,y,z}} ⇐ Hardware depth test;
    end for
  end if

  Casting-rays from camera with each ray-casting entry point R(p);
  Distribute R(p) to GPU node G(p);
  for each GPU node
    Sample x_i from R_p ∩ Dbuf{P_{x,y,z}};
    s_i ⇐ Density attribute value; // (Eq. 1)
    isosurface ⇐ Ray-casting method;

    while x_i ∈ isosurface
      N_i' ⇐ Basic estimation normal; // (Eq. 2)
      C_i ⇐ Covariance matrix; // (Eq. 3)
      N_i'' ⇐ Smallest eigenvector and eigenvalue of C_i;
      N_i ⇐ Blend N_i' and N_i''; // (Eq. 5)
    end while
  end for
Output: Fluid Surface and surface normal
```

Listing 1. Pseudocode of our method.

els, these methods are robust. The normal-estimation step in our method is similar to the methods in the point-cloud process, but, in our algorithm, the particle set is just a solid representation of the fluid volume. We combine PCA with standard SPH gradient estimation to calculate normals and generate a smooth and detail-preserving representation.

3. Algorithm

In this work, we take the particles' position as input. Figure 2 gives an overview of our algorithm. We first render all particles as spheres to generate depth buffers near the camera after setting the initial state. For each pixel, we cast a view ray from the generated depth buffer and sample the fluid-density attribute of neighbor particles step-by-step until the defined iso-value. is reached. Finally, we evaluate the normal direction at the isosurface for each pixel based on the neighbor particles' distribution (see Listing 1).

3.1. Surface-Depth Estimation

In this step, we use a ray-casting technique on the GPU and create one thread for each view ray. The key to parallel computation on the GPU is the load balance, and the balance of computation for all the view rays is essential for the performance. In our method, we sample the neighbor particles from the entry point step-by-step until attaining the isosurface on each view ray. Therefore, we need to reduce the sampling times and make the entry point close to the isosurface. In our method, we address this problem by employing a technique similar to splatting methods [Adams et al. 2006; Müller et al. 2007; van der Laan et al. 2009], rendering all the particles as spheres in the view-port and generating depth buffers by usinig a hardware depth test. We use point sprites (screen-oriented quads) with depth replacement in the fragment shader to rasterize the spheres [van der Laan et al. 2009].

Note that unlike particle-splatting rendering methods, we do not use the depth buffer as the final surface. We take it as the entry point for each view ray to reduce unnecessary computation in empty space. As a result, our method doesn't have problems like the traditional particle-splatting method[Adams et al. 2006], because we only use the depth buffer as an entry point and extract the surfaces by the following two steps.

3.2. Isosurface Extraction

In our method, the isosurface is constructed based on ray-casting techniques, and the scalar field is constructed using the standard SPH density estimation [Desbrun and Gascuel 1996] which is similar to the classical metaball method [Blinn 1982]. Our method can skip empty space quickly and only use ray casting in a narrow band near the isosurface. For each sampling point x_i along the view ray, the density attribute value s_i is

$$s_i = \sum_j m_j W(x_i - p_j, r), \tag{1}$$

where m_j is the mass of particle j (it is a unit value in our experiments and we will omit it in subsequent equations), W is the smoothing kernel function, p_j is the position associated with neighbor particle j, and r is the smoothing radius specifically used in the rendering step. We set r equal to the smoothing radius h.

Unlike traditional isosurface extraction methods, we do not use the generated isosurface as the final surface representation in this step. Instead, we will further evaluate the normal direction for each view ray based on a more sophisticated scheme and use this normal vector for rendering. The advantage is that we can avoid bumpy shape and non-smooth surface (see Figure 3) and generate a high-quality surface representation at the minimum computation cost. Specifically, this ray-casting isosurface generation step has the effects of smoothing based on density and can filter the noise caused by the fluid simulation step.

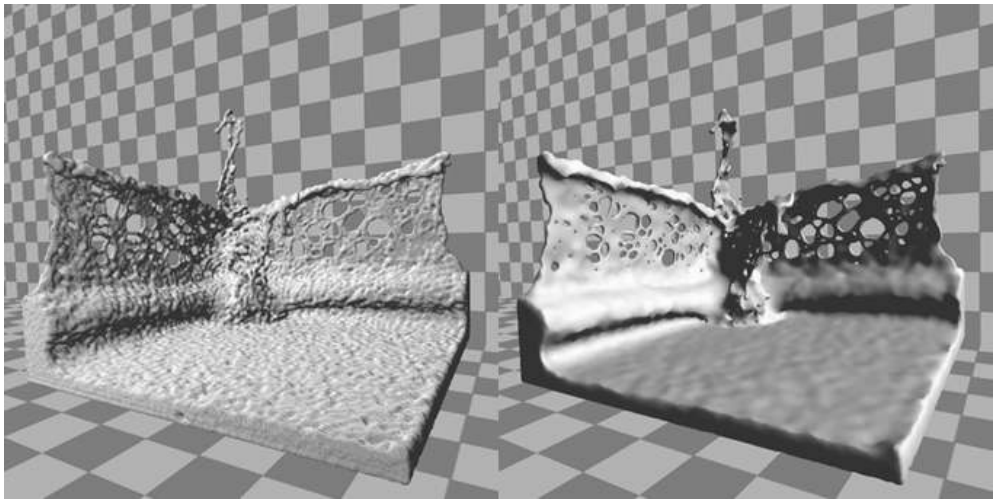

Figure 3. The left image is the output of the isosurface extraction step, and the right image is the smoother surface after the normal estimation step.

3.3. Surface Normal

Based on the isosurface, our approach will estimate a normal direction for each pixel point as the final surface representation. Because this step will be performed only once for each view ray on the GPU, we use a sophisticated estimation scheme with a relatively expensive computation without influencing much of the overall speed.

We compute the normal $N_i{}'$ of the sample point x_i at the positions $p_1, \cdots, p_n$. We first propose a basic estimation algorithm:

$$N_i{}' = \sum_j \nabla W(x_i - p_j, R). \tag{2}$$

The smoothing radius R is set to 3 in our experiments. This normal estimation follows the standard gradient calculation in [Monaghan 1992], using the spiky kernel from [Müller et al. 2003]. The basic normal estimation will give an approximate result which still suffers from the blobby effects for flat surfaces and smoothing effects for sharp corners (see Figure 1 left).

Inspired by the normal estimation techniques [Hoppe et al. 1992; Guennebaud and Gross 2007; Huang et al. 2009] for point-cloud processing and the anisotropy kernel surface-reconstruction method proposed by Yu and Turk [Yu and Turk 2013], we perform a PCA method at the query point x_i to decide the normal direction by the eigenvector with the least eigenvalue. The covariance matrix C_i is formulated as follows:

$$C_i = \frac{\sum_j W(x_i - p_j, R)(p_j - \bar{x_i})(p_j - \bar{x_i})^T}{\sum_j W(x_i - p_j, R)}; \tag{3}$$

$$\bar{x_i} = \frac{\sum_j p_j W(x_i - p_j, R)}{\sum_j W(x_i - p_j, R)}, \tag{4}$$

where $\bar{x_i}$ represents the average position of neighbor particles around a query point x_i. The smoothing kernel we used in our experiments is

$$W(x_i - p_j, R) = 1 - (\|x_i - p_j\| / R)^3.$$

We then used a Jacobi iteration method to evaluate three pairs of eigenvectors associated with eigenvalues. The eigenvector with the smallest eigenvalue will be used as the normal direction. The sign of this output vector is not determined, and we choose the direction which is the most consistent with the basic normal estimation N_i'. We define this result as N_i''.

Due to the inherent low-pass feature of PCA, the estimated normal is robust to noise and smoother than the basic normal estimation. Furthermore, fluid details like sharp corners can be preserved well. However, the method will give a poor result when there are not enough neighbor particles around the query point x_i or if C_i is a singular matrix (see Figure 1 middle). In these cases, we must use the basic normal estimation N_i'. In order to produce a smooth transition, we blend the N_i' and N_i'' in a unified form:

$$N_i = (1 - w)N_i' + wN_i'', \tag{5}$$

$$w = \min(1, e^{d(N_i' \cdot N_i'' - k)}), \tag{6}$$

where d and k are the shape control parameters for the blending function. The default value of d is 10 and k is 0.998 in our experiments. Additionally, we set the blending parameter w equal to 0.5 in our experiments. Note that both N_i' and N_i'' are normalized. When the PCA result N_i'' is very close to the basic result N_i', the final output N_i is roughly equal to N_i'' when w is close to 1. On the contrary, when N_i is far from N_i', the N_i will be dominated by N_i' as w will decrease dramatically. Finally, we perform a bilateral Gaussian filtering [Aurich and Weule 1995] on the generated normal map to get a smoother result while preserving the sharp feaures (see Figure 1 right).

3.4. Rendering

In the final step, we can render the surfaces into transparent surfaces using different materials. The generated normal map is just the layer which is nearest to the camera of the particle set. In order to produce an illusion of volume, we need to estimate the thickness of the fluid. We follow the method described in [van der Laan et al. 2009] to estimate the fluid thickness. Specifically, all the particles are rendered as spheres in world space just as in Section 3.1 but the difference is that the spheres have an alpha transparency. By enabling the alpha blending, the output of this rendering pass is an estimation of the fluid thickness. The computation and memory cost of this step only depend on the resolution of the image space.

4. Results

We have implemented our rendering algorithm entirely on the GPU using CUDA 8.0 and the Opengl Shader language. Our fluid dynamic results are mainly simulated by PBF [Macklin and Müller 2013] with a fixed time-step of 0.02s. Regarding neighbor-finding, both in simulation and surface construction, we use the GPU hash grid method described in [Owens et al. 2008]. Additionally, we follow the method of [Akinci et al. 2013] to handle the fluid-solid coupling. All of our simulation and rendering algorithms are run on a desktop PC with a NVIDIA GTX 970 graphics card and a 4.0GHz Intel(R) Core(TM)i7 CPU. We obtained rendering results with different resolutions. Figures displayed in this paper are 1024×1024.

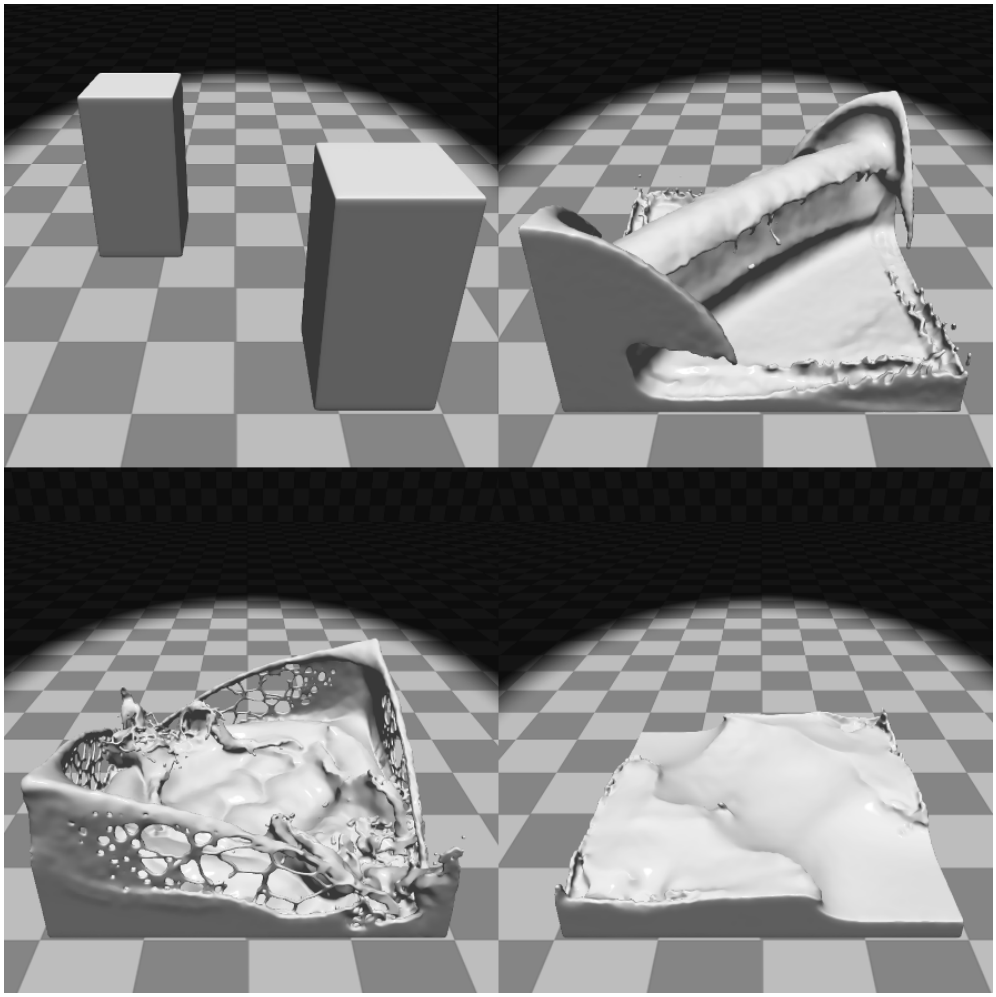

Figure 4. Breaking two fluid cubes and rendering the fluid surface in real time. The total particle number is 100k and the surface rendering takes on average 12ms.

4.1. Experiments

As in previous work [Yu and Turk 2013], we simulated the same double dam-break scenario. We can see that in the top-left of Figure 4, the surfaces of the two cubes are smooth and preserve sharp features, such as cube corners. The top-right and bottom-left subfigures display the surfaces with splash and thin fluid-sheets details. In the bottom-right, the rendered fluid surface looks smooth and soft. The rendering time was just 12ms per frame for 100k particles, while the anisotropic kernel method would need 1.6s for 24k particles. In Figure 5, we also show the surface rendering results of the single dam-breaking scene. In this case, the particle number is 600k, and the simulation and rendering frame rate is 10fps. Our surface-rendering step only takes 16% of the total time.

We also compared our method with the screen space fluid-rendering method [van der Laan et al. 2009] and the anisotropy kernel screen space fluid rendering method which was used in [Macklin and Müller 2013; Macklin et al. 2014] in the case of opaque rendering. In Figure 6, our method produced smoother surfaces than the mentioned real-time rendering methods.

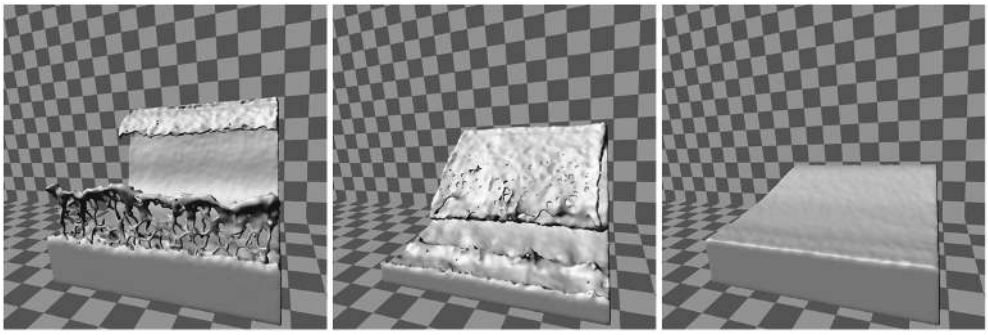

Figure 5. Surface rendering of single dam-break simulation with 600k particles and resolution 1024×1024.

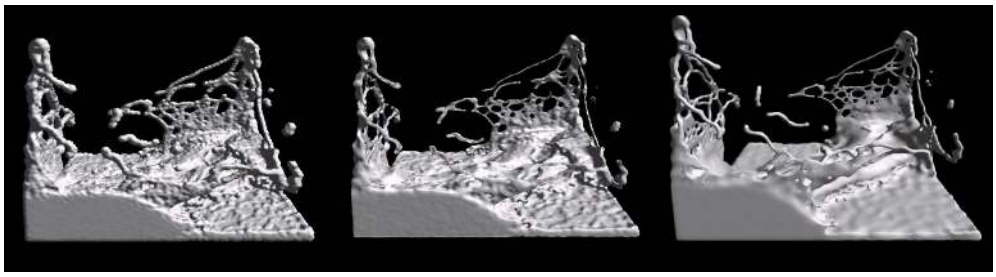

Figure 6. A comparison between the screen space fluid-rendering method (left), the anisotropy screen space fluid-rendering method (middle), and our method (right).

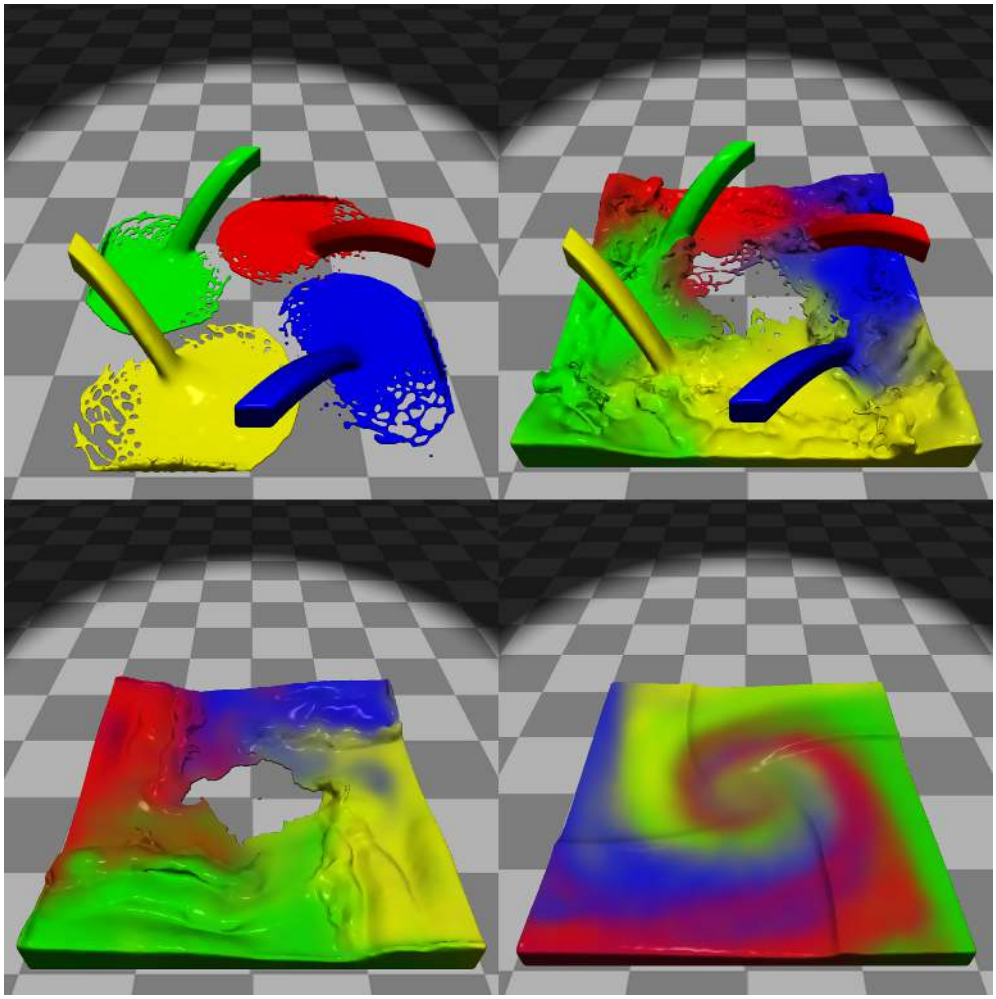

Figure 7. A scenario includes four water spouts in different directions, fluids with different color splash from the spouts and blend together in a pool finally.

Figure 7 shows a scene that includes four water spouts in different directions. Water with different colors splash from the spouts and blend together in a pool. We marked those fluid particles with target colors and calculated the color's weighted mean when processing the surface estimate. In this way, we can track the surface color and render it.

For large-scale scenes, we performed two series of experiments with different particle numbers and different resolutions. We first used Realflow [1] to create fluid particles and then imported those sets into our project to produce rendering results. Table 1 lists the details of our experiments. The particle counts were automatically created by Realflow after setting the basic sizes.

[1] http://www.realflow.com/

Scene	Particle count / Time	4738624	10532800	15201480
Waterfall (Figure 12)	Frame Time	0.0721s	0.1403s	0.1719s
	Total Time	28.8454s	56.1205s	68.3608s

Table 1. Frame rendering time and total rendering time of the waterfall scenes with different particle numbers in resolution 1024×1024. Total frame: 400.

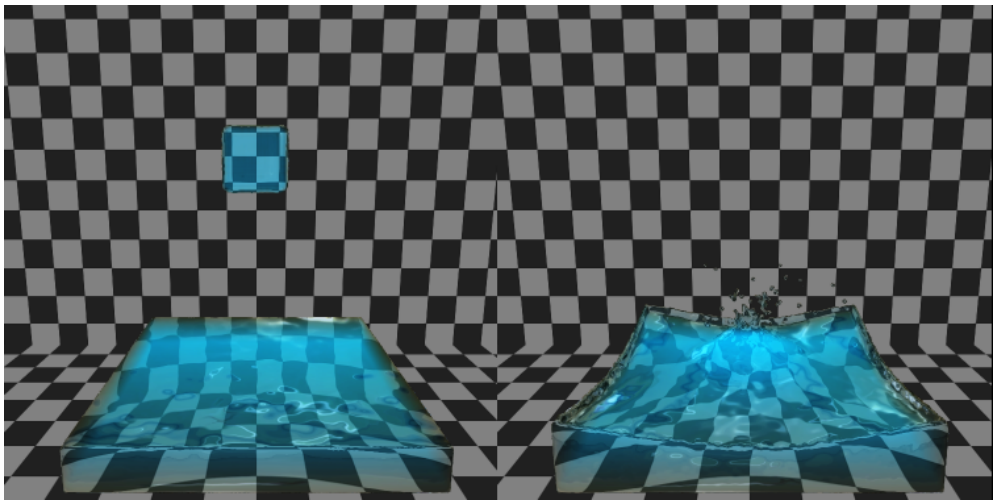

Figure 8. Transparent rendering results of water-splash scenario with 800k particles.

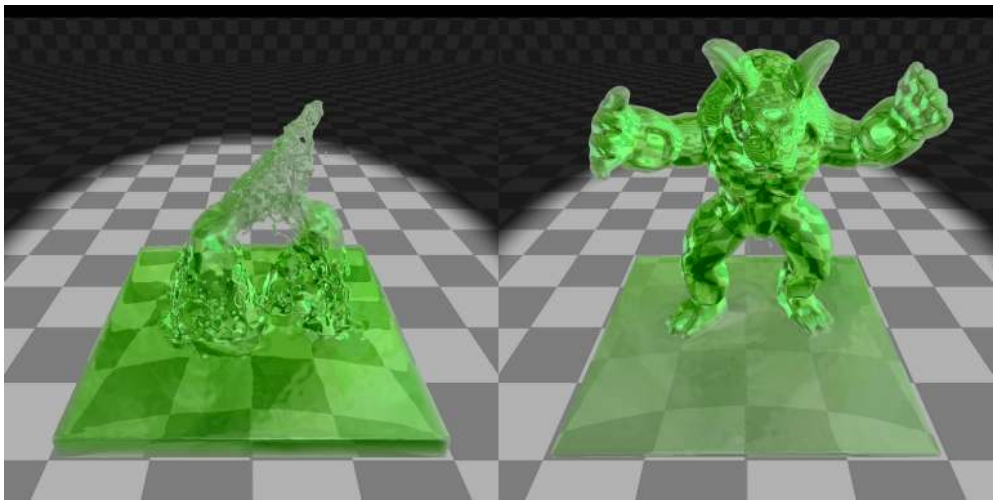

Figure 9. Fluid control-model rendering with150k particles.

In Figure 8, we show the transparent rendering results of a water-splash scenario. In this scene, a drop of liquid splashes into a larger body of water and sends up pearly spray. We can reflect and refract the surrounding environment and capture the isolated water droplets and make the fluid volume-like instead of just a visible surface.

In Figure 9, we rendered the fluid-control model [Zhang et al. 2015]. From this scene, we can conclude that our method can be integrated with existing particle-based simulation schemes easily and has the ability to render complex fluid models.

In Figures 10 and Figure 11, we show the water crown-scenes that contain millions of particles. In Figure 10, our method can produce different rendering results according to different rendering materials. In Figure 11, the particle number has reached 38 million while the rendering time is 0.2966s per frame. We also executed

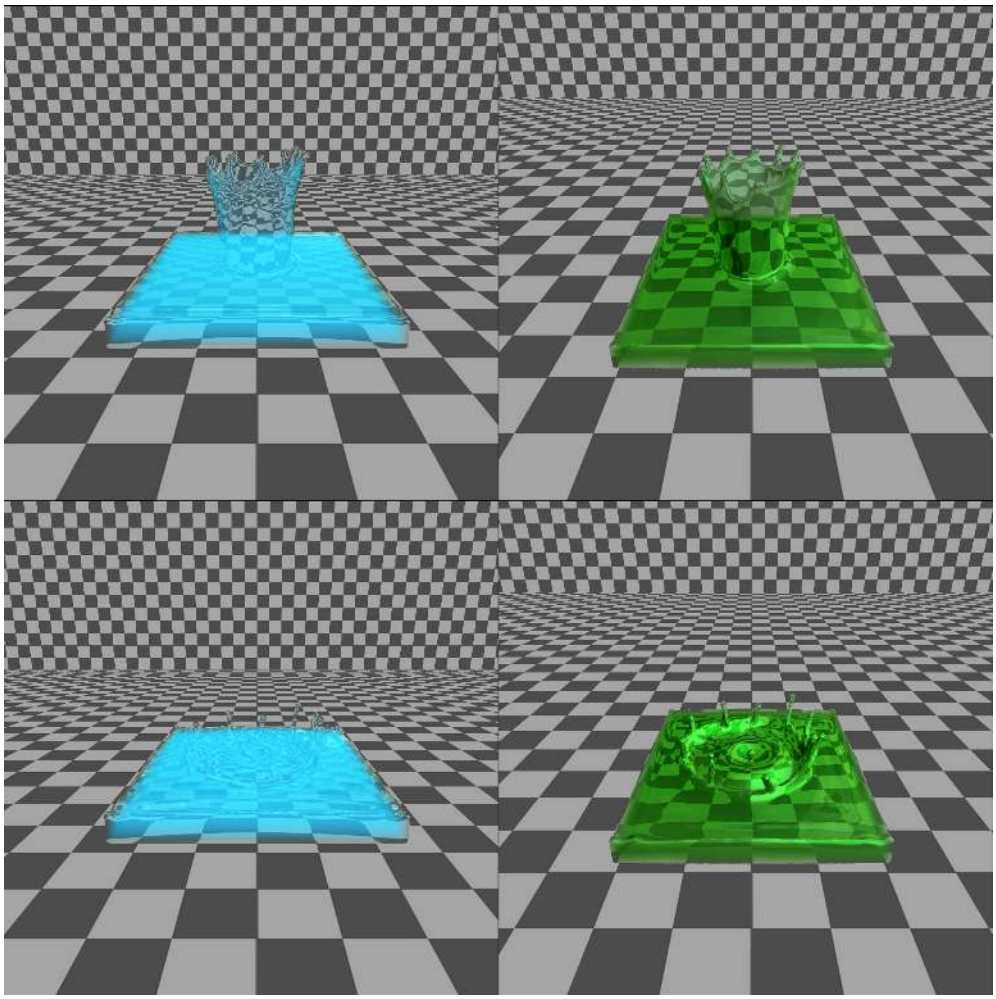

Figure 10. Transparent rendering results of water-crown scene with different materials and two million particles

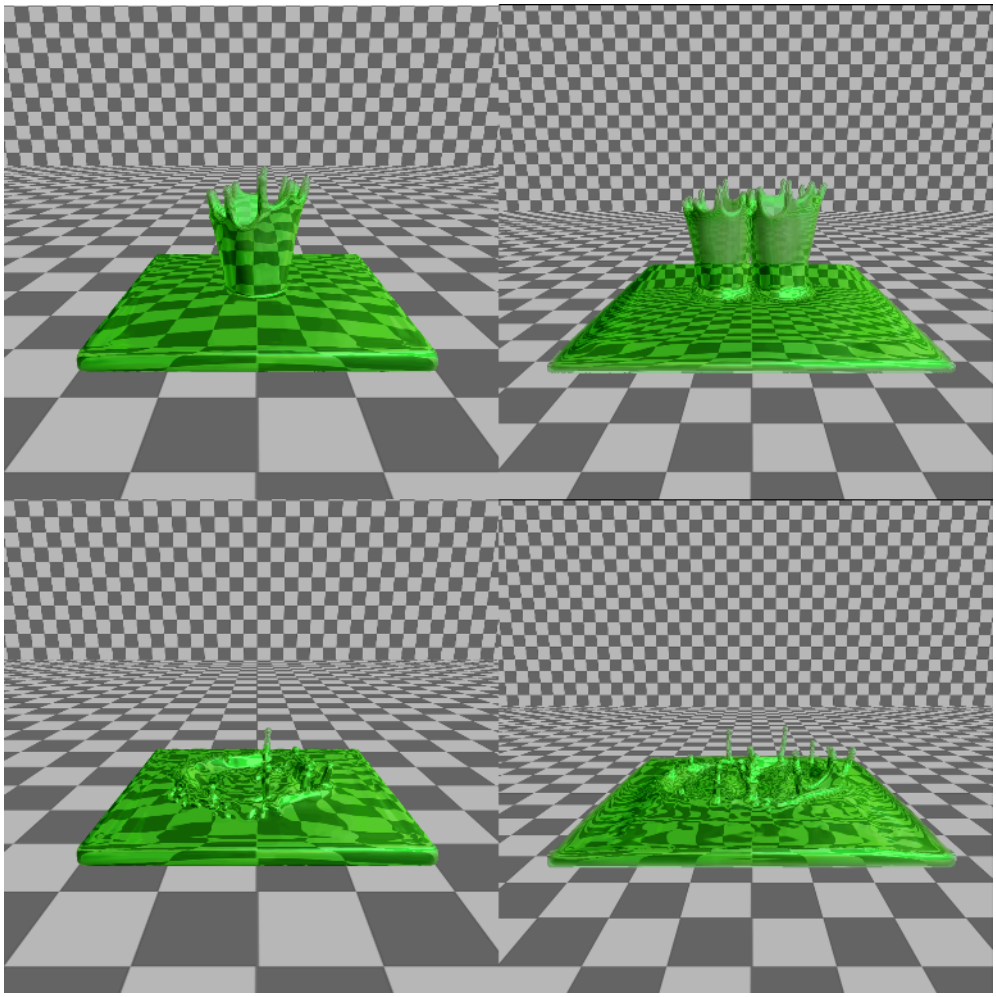

Figure 11. Rendering results of two different large-scale water-crown scenes. Left: column 12.4 million particles; Right column: 37.9 million particles.

a water-crown scene that contains more than 140 million particles which can also be run in a level of interaction (due to the limitation of our machine's storage capacity, we just performed 65 frames by Realflow in this scene).

In order to further illustrating the rendering details of our method, we implemented a waterfall scene and show the rendering results of the 133^{rd} frame and the 310^{th} frame in Figure 12. In this scene, we applied different background and illumination effects from previous results. Furthermore, users can design different scenes and adopt different rendering details, e.g., background, light direction and intensity, using our rendering system.

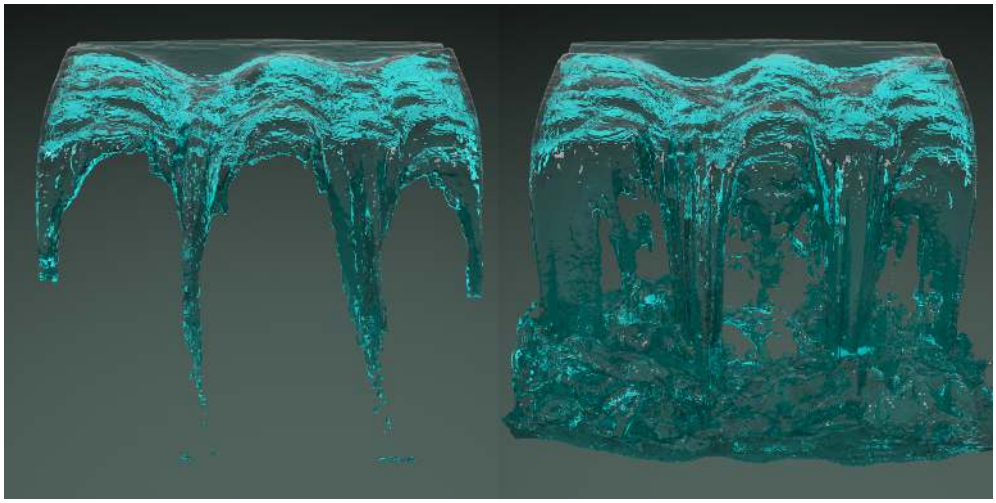

Figure 12. Rendering results of the large scale waterfall scenes. Resolution: 1024×1024, with 15.2 million particles.

4.2. Performance Evaluation

4.2.1. Data analysis

To evaluate our method, we list the rendering time with different particle numbers and different resolutions in Table 2. These results indicate that our proposed rendering method has achieved a real-time level. We can also render fluids containing more than 30 million particles interactively. In Table 3, we further list the time-cost of each rendering step in our method. From this timeing table, we can see that the main contributions of this work: surface-depth estimation, isosurface extraction, and normal computation cost of about 50% of the running time in our rendering system.

To further demonstrate the robustness of our rendering method, we selected different isovalues defined in our rendering procedure in Section 3.1 and recorded rendering times. In our method, a high isovalue will increase the sample times of rays cast from the camera and influence our rendering time. In Figure 13, isovalues change from

Resolution \ Particle count	779752	2027776	12394215	37975520	142612764
512×512	0.0097s	0.0274s	0.0891s	0.1565s	0.5926s
768×768	0.0127s	0.0362s	0.1128s	0.2119s	0.6600s
1024×1024	0.0178s	0.0514s	0.1568s	0.2966s	0.7840s
1440×1440	0.0270s	0.0779s	0.2474s	0.4494s	1.3879s
1920×1920	0.0424s	0.1226s	0.3733s	0.7062s	2.3663s

Table 2. Frame rendering time of our method with different particle numbers and different resolutions for water-crown scenes (Figure 10 and Figure 11).

Resolution \ Rendering step	Surface depth estimation	isosurface extraction	Surface normal computation	Rendering
512×512	0.0133s	0.0092s	0.0258s	0.0408s
768×768	0.0169s	0.0113s	0.0328s	0.0518s
1024×1024	0.0235s	0.0157s	0.0455s	0.0721s
1440×1440	0.0268s	0.0308s	0.0713s	0.1185s
1920×1920	0.0601s	0.0392s	0.1382s	0.1358s

Table 3. Time cost of each rendering step in Figure 11 (left column); particle count: 12.4M.

10.0 to 30.0. From the top left, the time varies from 0.0174s per frame to 0.0182s per frame, and the rendering time increased in a very slow gradient. The four sub-figures all depict that the variances of the scenes with different particle numbers are small, which indicates that the performance of our rendering method is not greatly influenced by the threshold value and maintains a robust level. We especially selected the medium isovalue in the experiments in this paper.

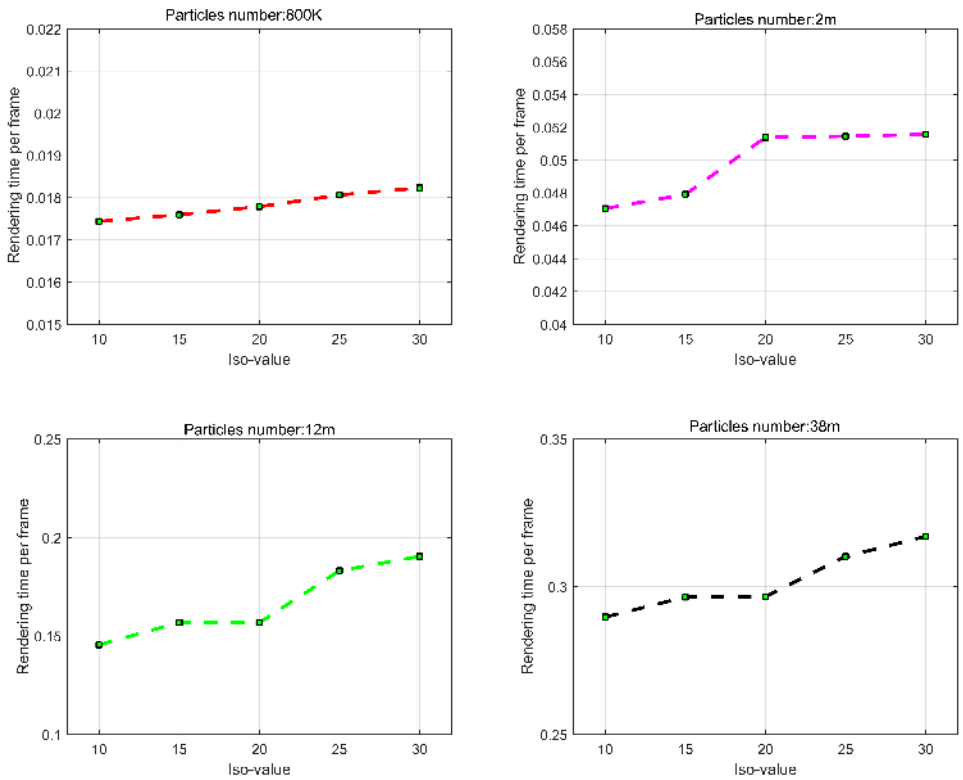

Figure 13. Time cost of our rendering method with different isovalues in different scenes with different numbers of particles (seconds).

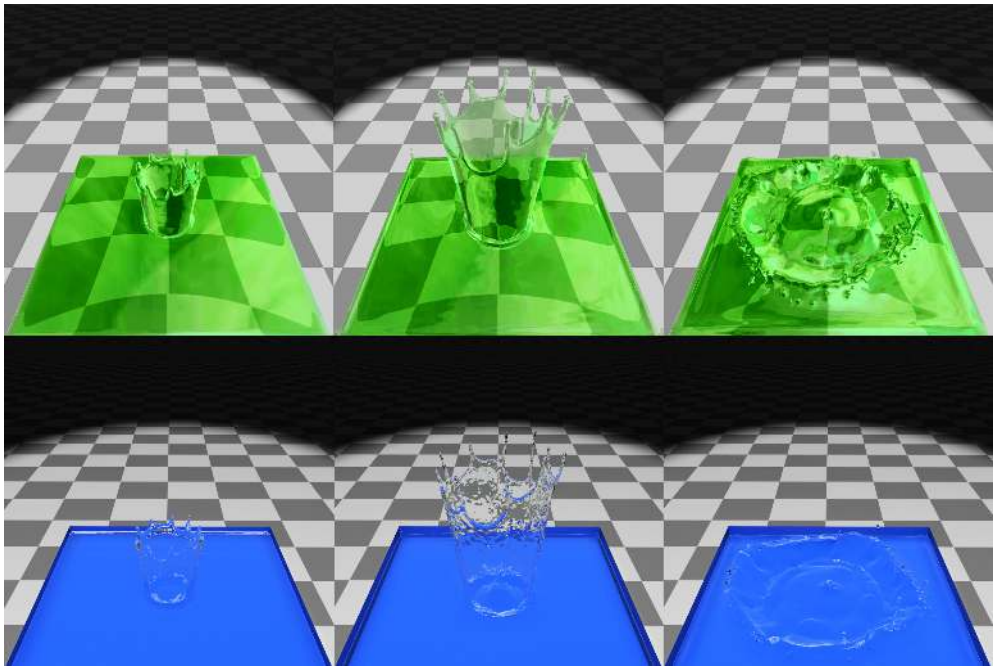

Figure 14. Rendering results comparison between our method (top) and off-line software (bottom). Particle number: 2 million, resolution: 1024×1024

4.2.2. Use as a Preview Application

Our rendering method can be use as a fast preview rendering system compared to off-line software, e.g., 3ds Max. In this section, we give the results of a comparison experiment in Figure 14. We imported the particle set generated by Realflow to our rendering system and obtained the rendering results in Figure 14 (top row), then we built the meshes generated by Realflow using the same particle set and imported them to the off-line rendering software (3ds Max 2013 [2] and V-Ray [3] 2.40.03). Figure 14 (bottom row) depicts the rendering results of the off-line software with simple rendering setup: single spot light, max depth of reflection: 2, max depth of refraction: 2.

Although the rendering results achieved by the off-line software are better than ours, the rendering time is intolerable. Table 4 lists the running time of our method and the off-line method with different particle numbers. Note that the statistical time results of the off-line method do not contain the mesh-generation step which consumes a large amount of time. These results indicate that our proposed rendering method achieved a real-time level and is suitable for high-quality preview applications, especially for large-scale SPH-based fluid.

[2] https://www.autodesk.com/products/3ds-max/
[3] https://www.vray.com/

Method	Particle No. / Time	779752	2027776	12394215	37975520	142612764
Our method	Frame Time	0.0178s	0.05138s	0.1568s	0.2966s	0.7840s
	Total Time	2.6704s	7.7076s	23.5200s	44.4990s	117.6530s
Off-line software	Frame Time	257.9521s	389.3879s	599.6077s	749.6510s	1199.7740s
	Total Time	38692.8207s	58408.1873s	89941.1550s	112447.6500s	179966.1000s

Table 4. Frame rendering time and total rendering time of our method compared to off-line software with different particle numbers. Total frames: 150, resolution: 1024×1024.

5. Conclusions

In this paper, we describe a simple and effective rendering technique for large-scale particle-based fluid. Our approach relies on the combination of three steps: determine an approximated surface by splatting particles on the view-port, find an isosurface nearest to the camera by casting view rays from the approximated surface for each pixel, and generate normals for isosurface pixels by performing PCA on the neighbor particles of the surface point. Thus, our method highlights the computation and memory resources on a narrow band near the isosurface and allows us to produce a smooth and feature-preserving high-quality surface in real time. Furthermore, our approach is straightforward to implement and can be integrated into existing particle-based fluid-simulation systems easily. Finally, our approach is a general isosurface extraction technique for particle sets which is not limited to fluid simulation. It can be used for both preview rendering and real-time isosurface visualization for large-scale particle sets in a wide range of applications.

There are several limitations to our method and possible improvements to be made in the future. First, our method is view-dependent and only renders the surface nearest to the camera. It does not support visualization for the fluid internal volume [Fraedrich et al. 2010], such as holes. Second, we only use single-layer refraction, which is not physically accurate. However, our result is close to two-interface refraction results in most cases due to the complexity of refraction [Wyman 2005], and the thickness-based shading further gives an illusion of volume of the fluid. So our result is visually convincing and can render surfaces more realistically compared to the four refractions algorithm [Imai et al. 2016]. Third, we currently use the simplest ray-casting model in which the reflection and refraction are traced only once in the rendering step. Some advanced rendering techniques like ray tracing or photon mapping can be adopted in the future to generate more realistic rendering results. Furthermore, there is a risk of over-smoothing as the number of particles is increased, so some sampling measures should be taken when we face this problem.

6. Acknowledgments

This work is based on an earlier work: "Real-time high-quality surface rendering for large scale particle-based fluids," Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, © ACM, {2017}. https://doi.org/10.1145/3023368.3023377.

References

ADAMS, B., LENAERT, T., AND DUTRÉ, P. 2006. Particle splatting: Interactive rendering of particle-based simulation data. Report CW 453, Departement Computerwetenschappen, KU Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium, July. URL: http://www.cs.kuleuven.be/publicaties/rapporten/cw/CW453.abs.html. 18, 20, 22

ADAMS, B., PAULY, M., KEISER, R., AND GUIBAS, L. J. 2007. Adaptively sampled particle fluids. *ACM Trans. Graph. 26*, 3. URL: http://doi.acm.org/10.1145/1276377.1276437, doi:10.1145/1276377.1276437. 18, 19, 20

AKINCI, N., AKINCI, G., AND TESCHNER, M. 2013. Versatile surface tension and adhesion for SPH fluids. *ACM Trans. Graph. 32*, 6, 182. URL: http://doi.acm.org/10.1145/2508363.2508395. 25

AURICH, V., AND WEULE, J. 1995. Non-linear Gaussian filters performing edge preserving diffusion. In *Mustererkennung 1995*. Springer, Berlin, 538–545. URL: http://dl.acm.org/citation.cfm?id=648280.754489. 24

BHATACHARYA, H., GAO, Y., AND BARGTEIL, A. 2011. A level-set method for skinning animated particle data. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, 17–24. URL: http://doi.acm.org/10.1145/2019406.2019409. 18

BLINN, J. F. 1982. A generalization of algebraic surface drawing. *ACM Trans. Graph. 1*, 3, 235–256. URL: http://doi.acm.org/10.1145/357306.357310. 19, 22

CORDS, H., AND STAADT, O. G. 2009. Interactive screen-space surface rendering of dynamic particle clouds. *Journal of Graphics, GPU, and Game Tools 14*, 3, 1–19. URL: https://doi.org/10.1080/2151237X.2009.10129282. 20

DESBRUN, M., AND GASCUEL, M.-P. 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation '96. Eurographics*, H. G. Boulic R., Ed. Springer, Vienna, 61–76. URL: http://dl.acm.org/citation.cfm?id=274976.274981. 18, 22

DREBIN, R. A., CARPENTER, L., AND HANRAHAN, P. 1988. Volume rendering. In *ACM Siggraph Computer Graphics*, ACM, New York, vol. 22, 65–74. URL: http://doi.acm.org/10.1145/54852.378484. 20

FRAEDRICH, R., AUER, S., AND WESTERMANN, R. 2010. Efficient high-quality volume rendering of SPH data. *IEEE Transactions on Visualization and Computer Graphics 16*, 6, 1533–1540. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5613495&isnumber=5613414. 20, 34

GREEN, S. 2010. Screen space fluid rendering for games. In *Proceedings of the Game Developers Conference*. URL: http://www.geeks3d.com/20100809. 20

GUENNEBAUD, G., AND GROSS, M. 2007. Algebraic point set surfaces. *ACM Trans. Graph. 26*, 3, 23. URL: http://doi.acm.org/10.1145/1275808.1276406. 20, 23

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph. 26*, 2, 71–78. URL: http://doi.acm.org/10.1145/142920.134011. 20, 23

HUANG, H., LI, D., ZHANG, H., ASCHER, U., AND COHEN-OR, D. 2009. Consolidation of unorganized point clouds for surface reconstruction. *ACM Trans. Graph. 28*, 5, 176. URL: http://doi.acm.org/10.1145/1618452.1618522. 20, 23

IHMSEN, M., ORTHMANN, J., SOLENTHALER, B., KOLB, A., AND TESCHNER, M. 2014. SPH fluids in computer graphics. In *Eurographics 2014 - State of the Art Reports*, The Eurographics Association, Aire-la-Ville, Switzerland, S. Lefebvre and M. Spagnuolo, Eds. 19

IMAI, T., KANAMORI, Y., FUKUI, Y., AND MITANI, J. 2014. Real-time screen-space liquid rendering with two-sided refractions. *Proceedings of NICOGRAPH International 2014*, 71–76. URL: https://ci.nii.ac.jp/naid/110009809744. 18, 20

IMAI, T., KANAMORI, Y., AND MITANI, J. 2016. Real-time screen-space liquid rendering with complex refractions. *Computer Animation and Virtual Worlds 27*, 3-4, 425–434. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1707. 18, 20, 34

KANAMORI, Y., SZEGO, Z., AND NISHITA, T. 2008. GPU-based fast ray casting for a large number of metaballs. *Computer Graphics Forum 27*, 2, 351–360. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01132.x. 20

KRUGER, J., AND WESTERMANN, R. 2003. Acceleration techniques for gpu-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, IEEE Computer Society, Los Alamitos, CA, 38. 20

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4, 163–169. URL: http://doi.acm.org/10.1145/280811.281026. 18, 19

MACKLIN, M., AND MÜLLER, M. 2013. Position based fluids. *ACM Trans. Graph. 32*, 4, 104. 19, 25, 26

MACKLIN, M., MÜLLER, M., CHENTANEZ, N., AND KIM, T.-Y. 2014. Unified particle physics for real-time applications. *ACM Trans. Graph. 33*, 4, 153. URL: http://doi.acm.org/10.1145/2461912.2461984. 26

MONAGHAN, J. J. 1992. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics 30*, 543–574. URL: http://stacks.iop.org/0034-4885/68/i=8/a=R01. 23

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, 154–159. URL: http://dl.acm.org/citation.cfm?id=846276.846298. 18, 19, 23

MÜLLER, M., SCHIRM, S., AND DUTHALER, S. 2007. Screen space meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, 9–15. URL: http://dl.acm.org/citation.cfm?id=1272690.1272692. 18, 20, 22

NAVRATIL, P. A., JOHNSON, J., AND BROMM, V. 2007. Visualization of cosmological particle-based datasets. *IEEE Transactions on Visualization and Computer Graphics 13*, 6, 1712–1718. URL: doi.ieeecomputersociety.org/10.1109/TVCG.2007.70616. 20

OWENS, J. D., HOUSTON, M., LUEBKE, D., GREEN, S., STONE, J. E., AND PHILLIPS, J. C. 2008. GPU computing. *Proceedings of the IEEE 96*, 5, 879–899. 25

REICHL, F., CHAJDAS, M. G., SCHNEIDER, J., AND WESTERMANN, R. 2014. Interactive rendering of giga-particle fluid simulations. In *Eurographics/ACM SIGGRAPH Symposium on High Performance Graphics*, The Eurographics Association, Aire-la-Ville, Switzerland, 105–116. URL: http://dl.acm.org/citation.cfm?id=2980009.2980021. 20

ROSENBERG, I. D., AND BIRDWELL, K. 2008. Real-time particle isosurface extraction. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, ACM, New York, 35–43. URL: http://doi.acm.org/10.1145/1342250.1342256. 20

SIN, F., BARGTEIL, A. W., AND HODGINS, J. K. 2009. A point-based method for animating incompressible flow. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, 247–255. URL: http://doi.acm.org/10.1145/1599470.1599502. 18

SOLENTHALER, B., AND PAJAROLA, R. 2009. Predictive-corrective incompressible SPH. *ACM Trans. Graph. 28*, 3, 40. URL: http://doi.acm.org/10.1145/1531326.1531346. 19

SZÉCSI, L., AND ILLÉS, D. 2012. Real-time metaball ray casting with fragment lists. In *Eurographics (Short Papers)*, Eurographics Association, Aire-la-Ville, Switzerland, 93–96. 19

VAN DER LAAN, W. J., GREEN, S., AND SAINZ, M. 2009. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, New York, 91–98. URL: http://doi.acm.org/10.1145/1507149.1507164. 18, 20, 22, 24, 26

WALD, I., AND SEIDEL, H.-P. 2005. Interactive ray tracing of point-based models. In *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.*, IEEE, Los Alamitos, CA, 9–16. URL: http://doi.acm.org/10.1145/1187112.1187176. 19

WILLIAMS, B. W. 2008. *Fluid surface reconstruction from particles*. Master's thesis, University of British Columbia. URL: https://open.library.ubc.ca/cIRcle/collections/24/items/. 18

WYMAN, C. 2005. An approximate image-space approach for interactive refraction. *ACM Trans. Graph. 24*, 3, 1050–1053. URL: http://doi.acm.org/10.1145/1073204.1073310. 34

YU, J., AND TURK, G. 2013. Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Trans. Graph, 32*, 1, 5. URL: http://doi.acm.org/10.1145/2421636.2421641. 18, 20, 23, 26

ZHANG, Y., SOLENTHALER, B., AND PAJAROLA, R. 2008. Adaptive sampling and rendering of fluids on the gpu. In *Proceedings of the Fifth Eurographics/IEEE VGTC conference on Point-Based Graphics*, Eurographics Association, Aire-la-Ville, Switzerland, 137–146. URL: http://dx.doi.org/10.2312/VG/VG-PBG08/137-146. 20

ZHANG, S., YANG, X., WU, Z., AND LIU, H. 2015. Position-based fluid control. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, ACM, New York, 61–68. URL: http://doi.acm.org/10.1145/2699276.2699287. 29

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. *ACM Trans. Graph. 24*, 3, 965–972. URL: http://doi.acm.org/10.1145/1073204.1073298. 18

ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. 2001. Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, 371–378. URL: http://doi.acm.org/10.1145/383259.383300. 20

Author Contact Information

Xiangyun Xiao
Digital ART Lab
School of Software
Shanghai Jiao Tong University
xiaoxiangyun@sjtu.edu.cn

Shuai Zhang
Digital ART Lab
School of Software
Shanghai Jiao Tong University
zhangshuai.03@bytedance.com

Xubo Yang
Digital ART Lab
School of Software
Shanghai Jiao Tong University
yangxubo@sjtu.edu.cn

Xiangyun Xiao, Shuai Zhang, and Xubo Yang, Fast, High-Quality Rendering of Liquids , *Journal of Computer Graphics Techniques (JCGT)*, vol. 7, no. 1, 17–38, 2018
http://jcgt.org/published/0007/01/02/

Received:	2017-08-31		
Recommended:	2017-11-25	Corresponding Editor:	Patrick Cozzi
Published:	2018-03-29	Editor-in-Chief:	Marc Olano