Real-Time High-Quality Surface Rendering for Large Scale Particle-based Fluids

Xiangyun Xiao * Shuai Zhang * Xubo Yang†

Digital ART Lab, School of Software
Shanghai Jiao Tong University

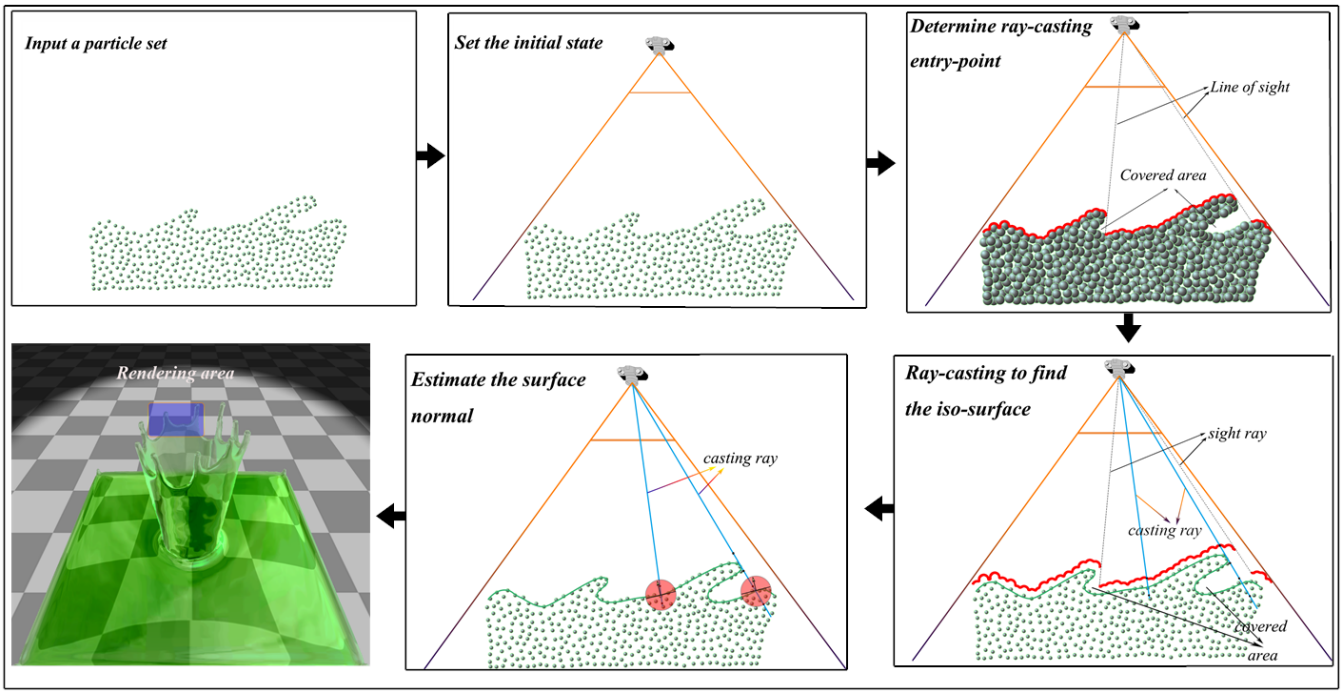

Figure 1: *An overview of our surface rendering algorithm. First step, input the position data; Second step, set the initial state; Third step, determine ray-casting entry-point by splatting particles; Forth step, ray-casting to find the iso-surface; Fifth step, estimate the surface normal; Finally, obtain the rendering result.*

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 coarse results, or time consuming. We introduce a novel approach to render high-quality fluid surface in screen space by an efficient combination of particle splatting, ray-casting and surface normal estimation techniques. We apply particle splatting to accelerate ray-casting process, and estimate surface normal using Principal Component Analysis (PCA). We adopt GPU technique to further accelerate our method. Our method can produce high-quality smooth surface while preserving thin and sharp details of large scale fluids. The computation and memory cost of our rendering step only depend on the image resolution. These advantages make our method very suitable for previewing or rendering hundreds of millions particles interactively. We demonstrate the efficiency and effectiveness of our method by rendering various fluid scenarios with different-sized particle sets.

Keywords: large-scale fluids, real-time rendering, particle splatting, ray-casting, normal estimation

*Equally contributed authors
†Corresponding author e-mail:yangxubo@sjtu.edu.cn

Concepts: •**Computing methodologies** → **Rendering; Ray tracing;**

1 Introduction

Particle-based methods like Smoothed Particle Hydrodynamics (SPH)[Desbrun and Gascuel 1996] for fluid simulation has received copious attention in the graphics community due to their flexibility and simplicity. Although the particle-based fluid simulation has been improved greatly, the high quality real-time surface rendering for large-scale fluid is still very challenging to existing particle rendering methods, thus significantly limits the particle-based methods to be applied in interactive applications.

Generally, to render liquids realistically, particle-based liquid surfaces can be been represented by a level set based on an implicit function and then extracted as polygonal meshes using march-

I3D '17,, February 25-27, 2017, San Francisco, CA, USA
ISBN: 978-1-4503-4886-7/17/03...$15.00
DOI: http://dx.doi.org/10.1145/3023368.3023377

ing cubes [Lorensen and Cline 1987] or marching tiles [Williams 2008]. Papers 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] were reported focusing on the construction of the implicit function to generate smooth iso-surfaces without blobby artifacts. However, due to full domain calculation and limited grid resolution, the implicit surface polygonization methods are both time consuming and memory intensive especially in large scale scenes. Furthermore, they also suffer from the temporal discretization artifacts[Adams et al. 2006] due to the limited grid resolution.

Another class of rendering techniques such as [Adams et al. 2006; Müller et al. 2007; van der Laan et al. 2009; Imai et al. 2014; Imai et al. 2016] which are often used for interactive applications are based on particle splatting. Those screen space based methods are able to render the particle sets in real-time with less memory used need no neighboring information. However, those methods cannot deal with noise situation and rarely have the ability to deal with hundreds of millions particles. Moreover, they always produce serrated edges with sphere shape on the fluid boundary, and significantly blobby artifacts when camera is near the fluid surface.

In this paper, we propose a novel rendering approach for particle based fluid which is capable of producing high-quality surface visualization in real-time for large scale scenes. The key idea of our method is to merely reconstruct the surface which is visible based on the 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 reaches the defined iso-value. The entry points of those rays are determined by performing a particle splatting step to generate an approximated surface. Therefore, our method omits the empty-space which does not contain any particles and enable the ray-casting to work in a narrow band of the iso-surface. To construct a smooth and detail-preserving surface representation, we calculate the normal directions by performing Principal Component Analysis (PCA) among the neighbour particles around the iso-surface instead of using the extracted iso-surface directly. We also design a GPU based algorithm which can speed up its efficiency significantly. Compared to previous related rendering methods, our approach is able to produce high-quality rendering results but significantly save computation and memory cost especially for large scale particle sets. Moreover, by adjusting the resolution of view-port, our method provides the trade-off between speed and quality to users. In addition, 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. Previous works like [Müller et al. 2003; Adams et al. 2007; Solenthaler and Pajarola 2009; Macklin and Müller 2013] had improved the simulation efficiency [Ihmsen et al. 2014], while, less researches were reported focusing on the high-quality surface rendering of large particle sets.

Traditionally, particle-based liquids were rendered by iso-surface methods [Lorensen and Cline 1987]. Early in 1982, Blinn [Blinn 1982] introduced the earliest metaball approach, the scalar field was constructed by summing up the radial basis function values. Muller et al. [Müller et al. 2003] improved this technique by smoothing the surface. Wald et al. [Wald and Seidel 2005] proposed a ray tracing method to deal with the point based models. Recently, Szecsi and Illes [Szécsi and Illés 2012] proposed a fast GPU-based metaball rendering method, they employed the A-buffer technique and fragment linked list. However, those methods always produce bumps and fail to yield flat surfaces even for a uniform 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 radiuses, and later, Yu and Turk [Yu and Turk 2013] proposed an anisotropic kernel approach to further modify the extracted surfaces. They can produce impressive liquid surfaces but the time consumption is considerable.

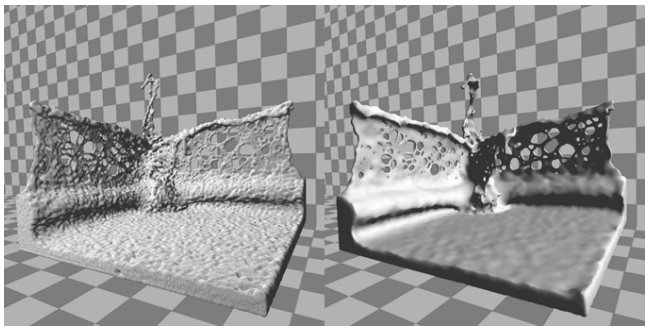

Figure 2: *The left image is the output of the iso-surface extraction step and the right image is the smoother surface after the normal estimation step.*

Also, ray-casting technique is widely used for volume rendering [Kruger and Westermann 2003]. It would be computationally expensive when applying the ray-casting over the whole particle set directly. A modification of this technique was to firstly resample particle quantities into a temporary uniform 3D grid in simulation domain [Navratil et al. 2007] and then performed a traditional volume ray-casting algorithm [Drebin et al. 1988; Kruger and Westermann 2003] on the 3D grid to extract an iso-surface. 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 iso-surface directly and avoid the memory consumption of a temporary grid. Ilya D. Rosenberg [Rosenberg and Birdwell 2008] proposed a fast iso-surface extraction technique on particles by dividing the simulation domain into blocks and extracting iso-surfaces within those blocks, which can achieve real-time performance with thousands of particles.

For surface reconstruction, surface splatting [Zwicker et al. 2001] is a widely used technique. In order to obtain a smooth surface representation based on surface splatting, Adams et al. [Adams et al. 2006] first determined the depth values of the foremost particles and blended the normals for the overlapping particles. Later, [Müller et al. 2007] constructed meshes in screen space which can accelerate surface splatting step. 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 catch up with the implicit surface polygonization methods although they can run in real-time. In the same time, Imai and their colleagues also presented a real-time rendering method that could handle multiple refractions [Imai et al. 2014] , but they would cause inaccurate refractions and suffered the same problems in the previous filtering-based approaches [Cords and Staadt 2009; Green 2010] that the resultant depth maps unnaturally warp towards the viewpoint around depth boundaries. Recently, they presented a new screen-space method [Imai et al. 2016] to handle with blobby artifacts in real time, whose goal was very similar to us, but our method can handle large scale particle-based fluids in high quality in real-time.

Normal estimation from a particle set is a well-studied problem

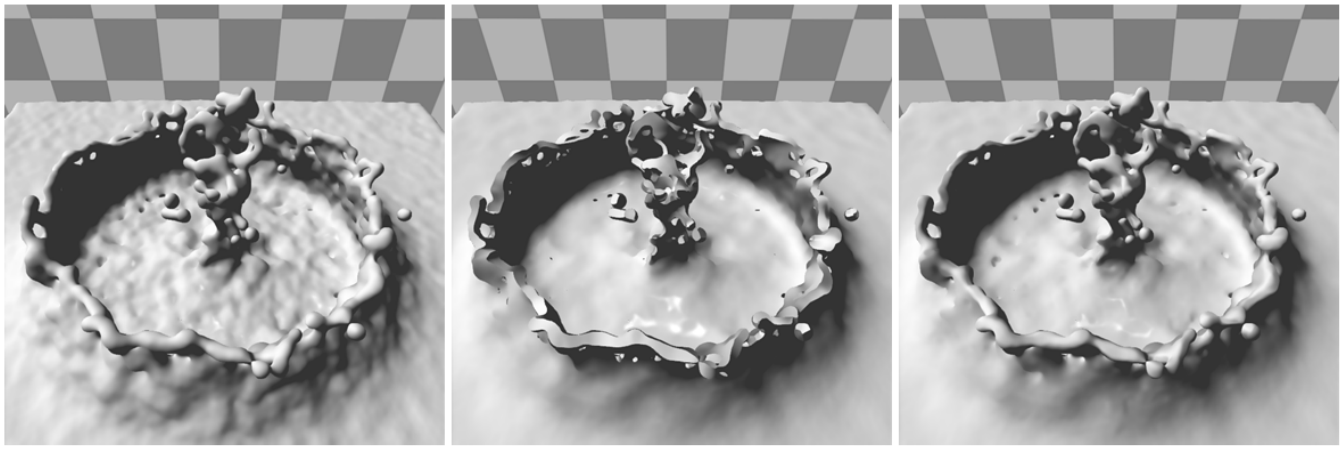

Figure 3: *A comparison of three normal estimation methods. Left image is the result of the basic normal estimation, middle image is the result of PCA, right image is the output of our method by blending the basic and PCA results. The basic normal estimation leads to an almost correct result except the non-smooth artifacts. The PCA result will fail under inadequate neighbor particles.*

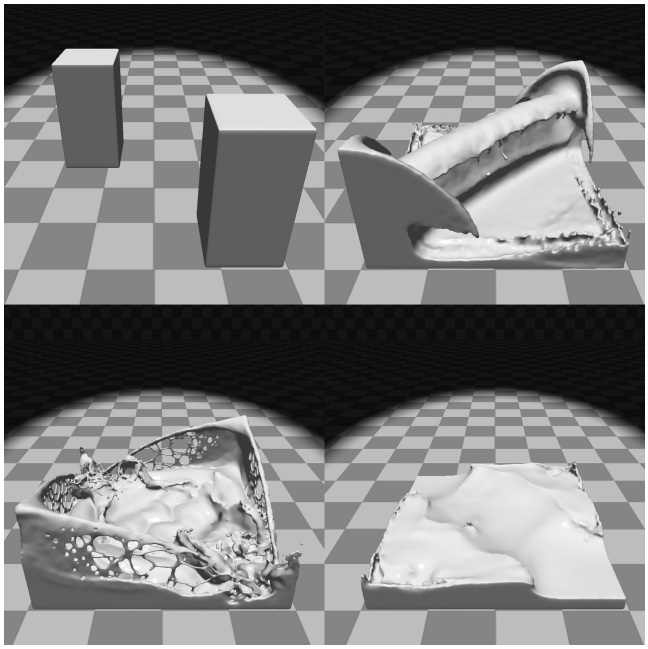

Figure 4: *Breaking two fluid cubes and rendering the fluid surface in real-time. The total particle number is 100k and the surface rendering takes 12ms averagely.*

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 the normal with the tangent plane generated by performing PCA over neighbor particles. Due to the inherent low pass filter property of regression model, these methods are robust. The normal estimation step in our method is similar to the case in point cloud process, but in our algorithm particle set is a solid representation of the fluid volume instead of the points on the model surface. We combine PCA with the standard SPH gradient estimation to calculate the normal of particle set surface and generate a smooth and detail-preserving representation.

3 Algorithm

In this work, we take the particles' position as input, Figure 1 is the overview of our algorithm. we render all particles as spheres to generate depth buffers nearest to the camera after setting the initial state. For each pixel, it will cast a view ray from the generated depth buffer and sample the fluid density attribute of neighbor particles step by step until reaches the defined iso-value. Finally, we evaluate the normal direction at the iso-surface for each pixel based on the neighbour particles' distribution.

3.1 Surface depth estimation

In this step, we will apply the ray-casting technique on GPU, and create one thread for each view ray. The key to paralleling computation on GPU is the load balance. Therefore, the balance of computation for all the view rays is essential for the performance. The operations on each view ray are sampling the neighbor particles from the entry point step by step until hitting the iso-surface. Therefore, we need to reduce the sampling times by making the entry point close to the iso-surface. 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 adopting the 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 final surface, instead, we take it as the entry point for each view ray to reduce unnecessary computation in empty-space. Also, our method doesn't have these problems that the traditional particle splatting method[Adams et al. 2006] suffered from, because we only use the depth buffer as an entry point, and extract the surfaces by the following two steps.

3.2 Iso-surface extraction

In our method, the iso-surface is constructed based on ray-casting technique 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 use ray-casting only in a narrow band near

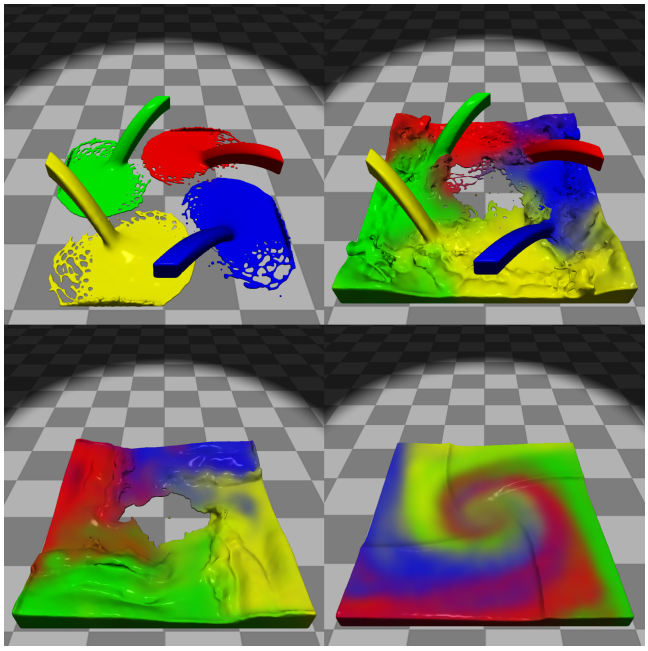

Figure 6: *A scenario includes four water spouts in different directions, fluids with different color splash from the spouts and blend together in a pool finally.*

the iso-surface. For each sampling point x_i along the view ray, the density attribute value is s_i:

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

Where m_j is the mass of particle j, which is an unit value in our experiments, we will omit it in subsequent equations. W is the smoothing kernel function. p_j is the position associated with neighbor particle j. r is the smoothing radius specifically used in the rendering step and we set r equals to the smoothing radius h. Different from traditional iso-surface extraction methods, we do not use the generated iso-surface 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 2) and generate a high-quality surface representation at the minimum computation cost. Specially, this ray-casting iso-surface generation step has the effects of smoothing based on density and can filter the noise caused by the fluid simulation step.

3.3 Surface normal

Based on iso-surface, our approach will further estimate a normal direction for each pixel point as the final surface representation. Because this step will perform only once for each view ray on GPU, we apply 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 as:

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

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

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

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

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

$\bar{x_i}$ represents the average position of neighbour 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 follow Jacobi iteration method to evaluate three pairs of eigenvectors associating with eigenvalues. The eigenvector with the least eigenvalue will be used as the normal direction. Note that the sign of this output vector is undetermined and we choose the direction which is most consistent with the basic normal estimation $N_i{}'$. We refer this result to PCA regarding N_i''.

Due to the inherent low pass feature of PCA, the estimated normal is robust to the noise and smoother than the basic normal estimation. Furthermore, fluid details like sharp corners can be preserved well. However, it will give a poor result when there are no enough neighbour particles around the query point x_i or if C_i is a singular matrix (see Figure 3 middle). In these cases, we must use the basic normal estimation N_i'. In order to produce a smooth transition, we will blend the N_i' and N_i'' to handle the results 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 equals to 0.5 in our experiments. Note that both N_i' and N_i'' here 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. In 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 feathers (see Figure 3 right).

3.4 Rendering

In the final step, we can render the surfaces into transparent surfaces and we can also use different materials to render them. The generated normal map is just a 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 like Section 3.1 while 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 image space resolution.

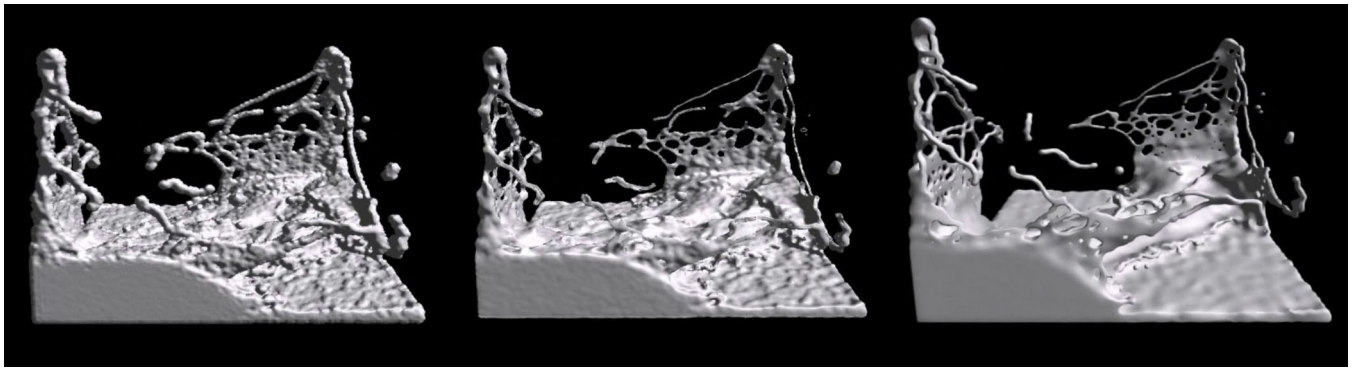

Figure 5: *A comparison among the screen space fluid rendering method (left), the anisotropy screen space fluid rendering method (middle) and our method (right).*

Resolution \ Number of particles	779752	2027776	12394215	37975520	142612764
512×512	0.00936	0.0274	0.0891	0.1565	0.5926
768×768	0.0127	0.0362	0.1128	0.2119	0.6600
1024×1024	0.0178	0.05138	0.1568	0.2966	0.7840
1440×1440	0.0270	0.0779	0.2474	0.4494	1.3879
1920×1920	0.0424	0.1226	0.3733	0.7062	2.3663

Table 1: *Rendering time(seconds) of our method with different particle numbers and different resolutions.*

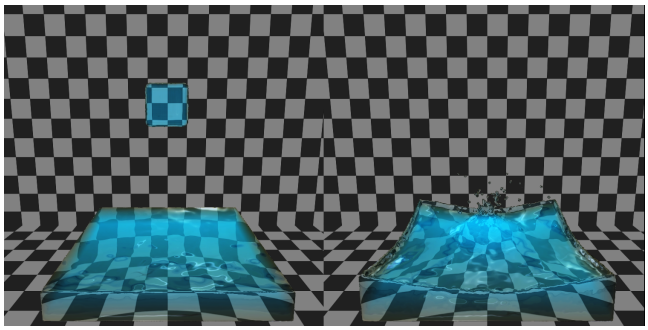

Figure 7: *Transparent rendering results of water splash scenario with 800k particles.*

4 Result

We have implemented our rendering algorithm entirely on GPU using CUDA 8.0 and 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 neighbour 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 algorithm 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 thought different resolutions. Figures displayed in this paper are 1024*1024.

Compared to [Yu and Turk 2013], we simulated the same double dam break scenario. In Figure 4, We can see that in the top left, the surfaces of the two cubes are smooth and preserve the sharp features like the cube corners. The top right and bottom left images display the surfaces with splash and thin fluid sheets details. In the bottom right image, the rendered fluid surface looks smooth and soft. The rendering time was just 12ms per frame for 100k particles, while anisotropic kernel method would need 1.6s for 24k particles. In order to further demonstrate our rendering results, 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. See figure 5, our method produced smoother surfaces than the two real-time rendering methods.

Figure 6 shows a scenario which includes four water spouts in different directions, where fluids with different colors splash from the spouts and blend together in a pool finally. We marked those fluid particles with target color aiming at calculating the color weighted average when we processed the surface estimate step. In this way, we can track the surface color of fluids and render it.

For large scale scenes, we performed two series of experiments with difference particle numbers and difference resolutions. We first used Realflow to create fluid particles, and then imported these sets into our project to produce rendering results. Table1 lists the details of our experiments, note that the numbers of particles were automatically created by Realflow after setting the basic sizes. These results indicate that our proposed rendering method has achieved a real-time level. Note that we can also render those sets contained more than 30 millions particles interactively.

In the animation of Figure 7, we show the transparent rendering results of our first water splash scenario which was simulated by PBF, drops of liquid splash into a larger body of water one by one, sending up pearly spray, We can reflect and refract the surrounding environment and capture the isolated water droplets successfully while making the fluid volume-liked instead of just a visible surface. In Figure 8 and Figure 9, we show the water crown scenes that contain millions particles. Note that Figure 8, our method can produce different rendering results according to different materials and in Figure 9, the particle number has been reached 38 millions while the rendering time is 0.2966s per frame. We also performed

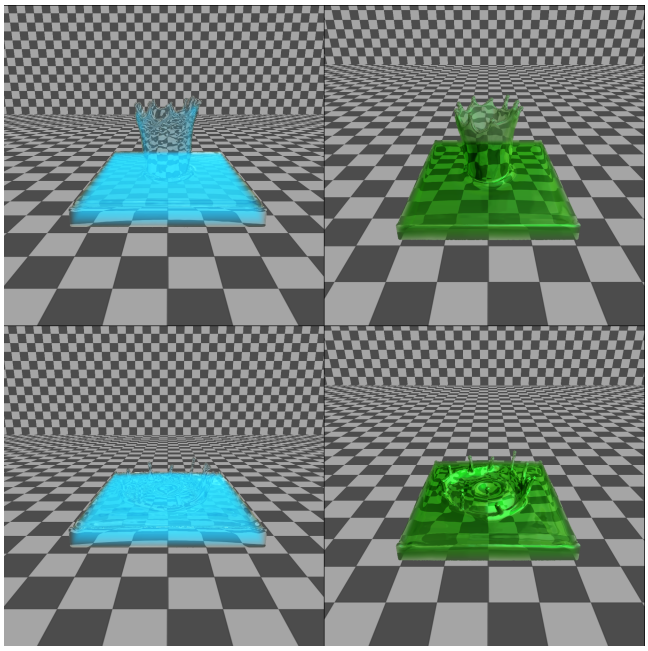

Figure 8: *Transparent rendering results of water crown scene with different materials. Particle number:2 millions.*

a water crown scene that contain more than 140 millions 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).

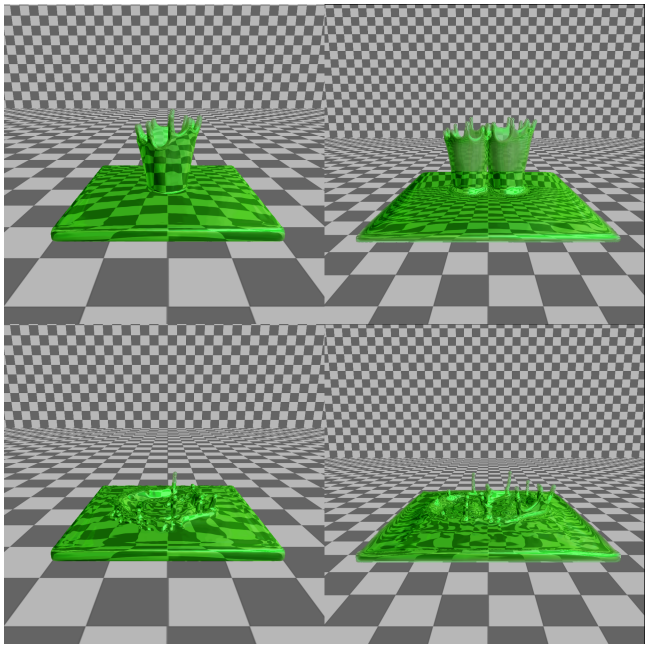

Figure 9: *Rendering results of two different large scale water crown scenes. Particle numbers: 12.4 millions(left column), 37.9 millions(right column).*

To further prove the robustness of our rendering method, we selected different iso-values defined in our rendering procedure in section 3.1 and recorded rendering times. In our method, a high iso-

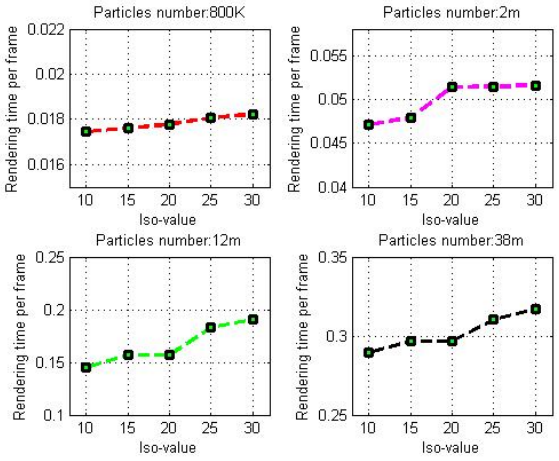

Figure 10: *Time cost of our rendering method with different iso-values in different scenes contained different numbers of particles (seconds).*

value will increase the sample times of rays casted from the camera and influence our rendering time. In Figure 10, iso-values change from 10.0 to 30.0. Focusing on 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 subfigures all depict that the variances of the scenes with different particle numbers are small, which prove that the performance of our rendering method is not greatly influenced by the threshold value and maintain a robust level. Specially, we selected the medium value in this paper.

5 Conclusion

In this paper, we describe a simple and effective rendering technique for large scale particle-based fluid. Our approach relies on the combinations of three techniques: determine an approximated surface by splatting particles on the view-port, find an iso-surface nearest to the camera by casting view rays from the approximated surface for each pixel, and generate normals for iso-surface pixels by performing PCA on the neighbor particles of the surface point. Therefore, our method highlight the computation and memory resources on a narrow band near the iso-surface 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 system easily. Finally, our approach is a general iso-surface extraction technique for particle set which is not limited in fluid simulation. It can be especially used as both preview rendering and real-time iso-surface visualization for large scale particle sets in a wide range of applications.

There are several limitations of our method and possible improvements in the future. Firstly, our method is view-dependent and only renders the surface nearest to the camera. It does not support the visualization for the fluid internal volume [Fraedrich et al. 2010] such as holes. Secondly, we only use single layer refraction which is not physically accurate. However, our result is close to two interface refraction result 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]. Thirdly, we currently use the simplest ray casting model 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.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China (No. 61173105, 61373085), and the National High Technology Research and Development Program of China (No. 2015AA016404).

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.

ADAMS, B., PAULY, M., KEISER, R., AND GUIBAS, L. J. 2007. Adaptively sampled particle fluids. In *ACM Transactions on Graphics (TOG)*, vol. 26, ACM, 48.

AKINCI, N., AKINCI, G., AND TESCHNER, M. 2013. Versatile surface tension and adhesion for sph fluids. *ACM Transactions on Graphics (TOG) 32*, 6, 182.

AURICH, V., AND WEULE, J. 1995. Non-linear gaussian filters performing edge preserving diffusion. In *Mustererkennung 1995*. Springer, 538–545.

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, 17–24.

BLINN, J. F. 1982. A generalization of algebraic surface drawing. *ACM transactions on graphics (TOG) 1*, 3, 235–256.

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.

DESBRUN, M., AND GASCUEL, M.-P. 1996. *Smoothed particles: A new paradigm for animating highly deformable bodies*. Springer.

DREBIN, R. A., CARPENTER, L., AND HANRAHAN, P. 1988. Volume rendering. In *ACM Siggraph Computer Graphics*, vol. 22, ACM, 65–74.

FRAEDRICH, R., AUER, S., AND WESTERMANN, R. 2010. Efficient high-quality volume rendering of sph data. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6, 1533–1540.

GREEN, S. 2010. Screen space fluid rendering for games. In *Proceedings for the Game Developers Conference*.

GUENNEBAUD, G., AND GROSS, M. 2007. Algebraic point set surfaces. In *ACM Transactions on Graphics (TOG)*, vol. 26, ACM, 23.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J., AND STUETZLE, W. 1992. *Surface reconstruction from unorganized points*, vol. 26. ACM.

HUANG, H., LI, D., ZHANG, H., ASCHER, U., AND COHEN-OR, D. 2009. Consolidation of unorganized point clouds for surface reconstruction. In *ACM transactions on graphics (TOG)*, vol. 28, ACM, 176.

IHMSEN, M., ORTHMANN, J., SOLENTHALER, B., KOLB, A., AND TESCHNER, M. 2014. Sph fluids in computer graphics.

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.

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.

KANAMORI, Y., SZEGO, Z., AND NISHITA, T. 2008. Gpu-based fast ray casting for a large number of metaballs. In *Computer Graphics Forum*, vol. 27, Wiley Online Library, 351–360.

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, 38.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, vol. 21, ACM, 163–169.

MACKLIN, M., AND MÜLLER, M. 2013. Position based fluids. *ACM Transactions on Graphics (TOG) 32*, 4, 104.

MACKLIN, M., MÜLLER, M., CHENTANEZ, N., AND KIM, T.-Y. 2014. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG) 33*, 4, 153.

MONAGHAN, J. J. 1992. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics 30*, 543–574.

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, 154–159.

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, 9–15.

NAVRATIL, P. A., JOHNSON, J., AND BROMM, V. 2007. Visualization of cosmological particle-based datasets. *Visualization and Computer Graphics, IEEE Transactions on 13*, 6, 1712–1718.

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.

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, 105–116.

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, 35–43.

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, 247–255.

SOLENTHALER, B., AND PAJAROLA, R. 2009. Predictive-corrective incompressible sph. In *ACM transactions on graphics (TOG)*, vol. 28, ACM, 40.

SOLENTHALER, B., SCHLÄFLI, J., AND PAJAROLA, R. 2007. A unified particle model for fluid–solid interactions. *Computer Animation and Virtual Worlds 18*, 1, 69–82.

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

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, 91–98.

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, 9–16.

WILLIAMS, B. W. 2008. *Fluid surface reconstruction from particles*. PhD thesis, Citeseer.

WYMAN, C. 2005. An approximate image-space approach for interactive refraction. In *ACM Transactions on Graphics (TOG)*, vol. 24, ACM, 1050–1053.

YU, J., AND TURK, G. 2013. Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Transactions on Graphics (TOG) 32*, 1, 5.

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, 137–146.

ZHU, Y., AND BRIDSON, R. 2005. Animating sand as a fluid. In *ACM Transactions on Graphics (TOG)*, vol. 24, ACM, 965–972.

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, 371–378.