Eurographics Symposium on Rendering (2022)
A. Ghosh and L.-Y. Wei (Editors)

A Real-Time Adaptive Ray Marching Method for Particle-Based Fluid Surface Reconstruction

Tong Wu, Zhiqiang Zhou, Anlan Wang, Yuning Gong and Yanci Zhang

National Key Laboratory of Fundamental Science on Synthetic Vision

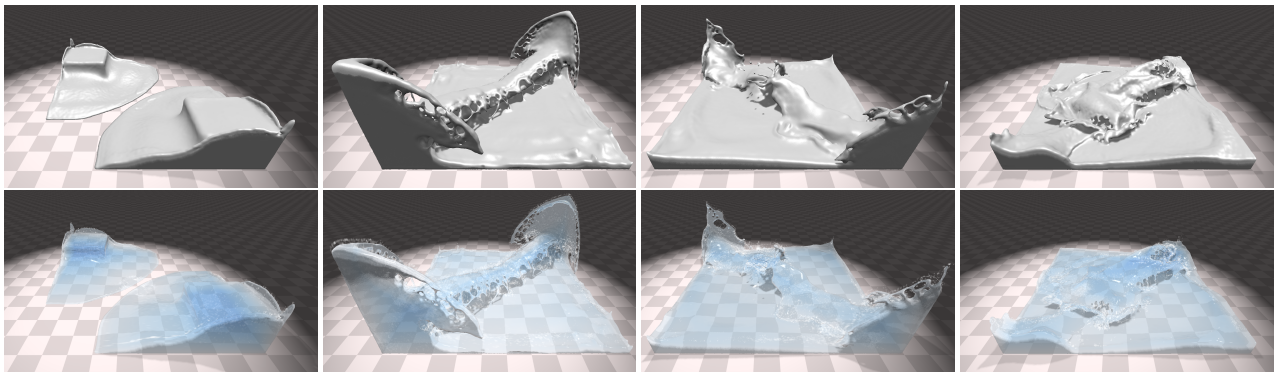


Figure 1: *A double dam break scenario.*

Abstract

In the rendering of particle-based fluids, the surfaces reconstructed by ray marching techniques contain more details than screen space filtering methods. However, the ray marching process is quite time-consuming because it needs a large number of steps for each ray. In this paper, we introduce an adaptive ray marching method to construct high-quality fluid surfaces in real-time. In order to reduce the number of ray marching steps, we propose a new data structure called binary density grid so that our ray marching method is capable of adaptively adjusting the step length. We also classify the fluid particles into two categories, i.e. high-density aggregations and low-density splashes. Based on this classification, two depth maps are generated to quickly provide the accurate start and approximated stop points of ray marching. In addition to reduce the number of marching steps, we also propose a method to adaptively determine the number of rays cast for different screen regions. And finally, in order to improve the quality of reconstructed surfaces, we present a method to adaptively blending the normal vectors computed from screen and object space. With the various adaptive optimizations mentioned above, our method can reconstruct high-quality fluid surfaces in real time.

CCS Concepts

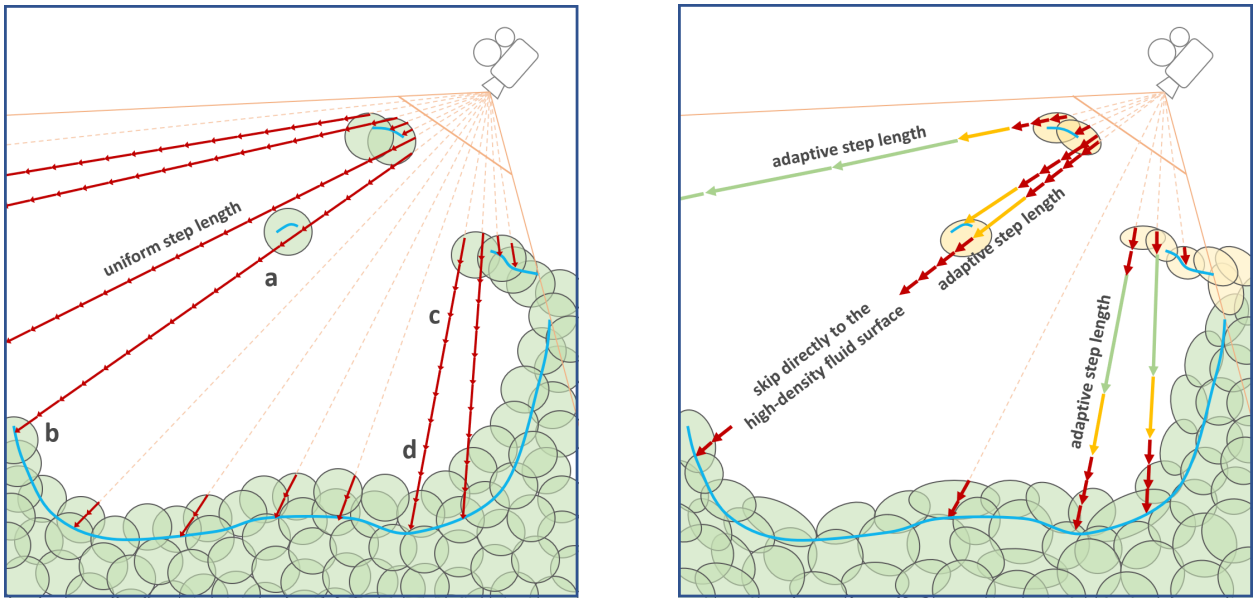
• ***Computing methodologies*** → ***Rendering;*** *Massively parallel algorithms;*

1. Introduction

Particle-based simulation methods like Smoothed Particle Hydrodynamics (SPH)[DG96] are widely adopted for video games and other real-time graphics applications. Even though the performance of CPU and GPU increases largely in recent years, reconstructing and rendering high quality fluids surface in real-time is still very challenging.

In real-time applications, fluid surfaces only need to be rendered from one viewpoint per frame. So it is not worthwhile to reconstruct the entire mesh for fluids. This is the reason why screen space methods are now the mainstream in the field of real-time fluid reconstruction. Screen space methods can be divided into two main categories depending on the space where the computations are performed: one[MSD07; Gre10; vdLGS09; IKM16; TY18] is to execute the computations in image space by smoothing the depth map of particles with some filters, and the other[XZY17] is to cast a ray per pixel to evaluate the iso-surface in object space. With respect

DOI: 10.2312/sr.20221157



(a) original algorithm[XZY17] (b) our adaptive algorithm

Figure 2: *A comparison between [XZY17] and our adaptive fluid reconstruction algorithm. The blue lines represent the final reconstructed surfaces. In our method, low-density splash particles are drawn as yellow ellipsoids and high-density particles are drawn as green ellipsoids.*

to the quality of reconstructed surfaces, the latter one can produce more realistic and natural-looking details. But it needs more computation overhead due to a large number of steps required in the ray marching process.

In this paper, we present a novel method to improve the performance and quality of the ray marching based algorithm. Three adaptive mechanisms are designed to fulfill the target. First of all, we generate a compact 3D density mask map and two depth maps. Based on these three maps, our ray marching can use an adaptive step length to quickly skip empty spaces between splashes and aggregations quickly. Secondly, we divide the screen into multiple equal sized tiles and use different resolutions to cast rays for different tiles depending on their depth variances. And finally, we adaptively blend the normal vectors computed from screen and object space to achieve a better reconstruction quality.

2. Related Work

In computer graphics, particle-based fluids simulation has became a popular method to simulate complex and diverse fluid effects[AIA*12; MM13; KS14; BK15]. However, there are relatively few methods in high-quality fluid reconstruction, particularly in real-time.

Iso-surfaces extraction. Traditionally, particle-based fluid was represented as blobby spheres or metaballs. Then iso-surfaces are extracted as polygonal meshes using marching cubes. Early in 1982, the classic metaballs approach had been introduced by Blinn[Bli82]. At the center of each particle, radial basis functions are placed and accumulated to construct a scalar field where iso-surfaces are then extracted. Zhu and Bridson[ZB05] improved this algorithm and obtain smoother surfaces. They performed smoothing pass over a dicrete grid that is constructed by sampling scalar field functions. Adams et al.[APKG07] tracked the particle-to-surface distances over time to generate smooth surfaces. Yu and Turk[YT13] used anisotropic kernels to stretch spheres into ellipsoids to produce flat surfaces and sharp features that are difficult to generate with spherical smooth kernels. Recently, Biedert et al.[BSS*18] improved on Yu and Turk's surface definition by presenting a novel direct raytracing scheme.

Screen-space Filters. Although the approaches of iso-surfaces extraction can generate accurate surfaces, they are computationally too expensive to provide the most efficient alternatives for real-time fluid reconstruction. Therefore, many screen-space filtering methods have been proposed to avoid polygonal mesh generation. In these methods, particles are first drawn as spheres or ellipsoids to create a depth map. And the filtering used to smooth this depth map has been the focus of many researches. Binomial filters[MSD07] and Gaussian filters can be implemented as separable filters, which relieve the computational pressure of large filter sizes. The bilateral Gaussian filter[Gre10] preserves the boundaries of the fluid and avoids completely unrelated parts being filtered. And van der Laan et al.[vdLGS09] also smoothed the depth map using curvature flow to provide outstanding fluid surface quality. Reichl et al.[RCSW14] proposed a total-variation-based image de-noising method to generate sharp fluid details. More recently, Truong and Yuksel[TY18] considered the depth values in a narrow range. The depth values outside of this range are carefully handled to provide smoother surfaces with well-preserved details near discontinuities.

Other screen-space approaches. The screen-space filtering approaches are capable of achieving real-time requirements, but still

have many shortcomings in terms of quality. To achieve more complex refractions, Imai et al.[IKM16] reconstructed the front- and back-facing surfaces of the splashes and aggregations separately. Xiao et al.[XZY17] only use the depth map as the entrance to the rays that are cast towards the fluid. For each step of the ray, a neighbor search is performed to evaluate whether the density iso-surface has been reached. Our method is also based on the ray marching to reconstruct the visible surface, but implements adaptive step lengths to improve marching efficiency.

3. Algorithm Overview

In order to keep the details of tiny droplets and splashes and make the fluid surfaces reconstructed more realistic, [XZY17] defines fluid surfaces as iso-surfaces for some specific density value. And a ray marching algorithm is executed to extract iso-surfaces by computing the density of samples on the ray. But this method needs a large amount of steps to find iso-surfaces which greatly hurts the performance. As illustrated in Figure 2(a), Xiao et al. generate a depth map from all fluid particles by rasterization, which provides the starting points of ray marching. Once the ray marching begins, it uses uniform step lengths to compute the fluid density from surrounding particles until the density threshold is satisfied. This mechanism may introduce many useless computations in empty spaces (like *ab* and *cd* in Figure 2(a)).

In this paper, we propose an adaptive ray marching method to reconstruct and render particle-based fluid surfaces. Actually three adaptive methods are employed in our algorithm: the adaptive ray marching step length provides the ability to reach the iso-surfaces in fewer steps; the adaptive reconstruction resolution can reduce the number of rays cast; and the adaptive normal blending weights help us to obtain better smooth fluid surfaces with less computation in all camera views.

The basic idea of our algorithm is illustrated in Figure 2(b).

- **Adaptive ray marching step lengths:** In order to allow the rays to dynamically adjust step lengths in marching, a binary density grid $\mathbf{M}$ is used to represent the rough distributions of fluid particles in the whole simulation domain. $\mathbf{M}$ divides the simulation domain into two classes based on the density of particles. With $\mathbf{M}$, rays can avoid meaningless neighbor searches when passing through low-density areas (See Section 4.1). We also classify particles into splashes and aggregations according to their anisotropic density. Based on this classification, two depth maps $\mathbf{D}_{all}$ and $\mathbf{D}_{agg}$ are generated from all particles and only high-density surface particles respectively. $\mathbf{D}_{all}$ defines the entrances of all rays, while $\mathbf{D}_{agg}$ provides a rough approximation where ray marching is supposed to stop. Based on $\mathbf{D}_{agg}$, rays can skip directly to aggregations when step counts exceed the threshold (See Section 4.2).
- **Adaptive ray casting resolution:** The screen is divided into tiles and the reconstruction resolution of each tile is evaluated based on the particles' information inside it, such as the variance of the depth $\mathbf{D}_{all}$. Fewer rays will be cast in those tiles containing flat surfaces, where the reconstruction result of pixels casting no ray will be obtained by interpolation (See Section 4.3).
- **Adaptively blending normal vectors:** To obtain smooth normals at a small cost, we use screen space filtering to smooth $\mathbf{D}_{all}$ and then reconstruct the normals $\mathbf{N}^{Screen}$ from it. However, $\mathbf{D}_{all}$ is only the entrance to the rays and is likely to be far from the final iso-surface. On the other hand, particles closer to the camera will leave spherical artifacts that are difficult to remove with filtering. To correct these two possible errors, we also estimate another normal $\mathbf{N}^{Object}$ by searching for neighboring particles in the object space and design two adaptive weights to blend these two normals (See Section 4.4).

Our overall process is shown in algorithm 1:

Algorithm 1: Algorithm overview.

```
M ← generateDensityMask();
D_agg ← generateAggregationDepth();
D_all ← generateAllParticlesDepth();
D_smooth ← smoothDepth(D_all);
N_screen ← generateNormals(D_smooth);
foreach tile t ∈ tiles do
    // Which pixels need to cast the ray
       depends on each tile's resolution.
    foreach pixel x ∈ pixelsCastingRays[t] do
        P ← getEntrance(D_all[x]);
        for i ← 0 to maxStepCounts do
            if i > maxSkipCounts and !hasSkipped then
                P ← getExit(D_agg[x]);
                hasSkipped ← true;
            end
            L ← getStepLength(M[P]);
            P ← P + L;
            D ← CalculateDensity(P);
            if D > iso then
                N_object ← getObjectNormal(P);
                N ← blendNormals(N_screen[x], N_object);
                Result[x] ← (P, N);
            end
        end
        Result[x] ← (NULL, NULL);
    end
end
```

4. Fluid Surfaces Reconstruction via Adaptive Ray Marching

4.1. Binary Density Grid

A time-consuming neighbor search is required to estimate the density at each step, which is the main cause of performance bottlenecks. It is crucial to enable the ability to skip those low-density areas to avoid meaningless neighbor searches.

In order to fulfill this target, we propose a new data structure called binary density grid $\mathbf{M}$. Instead of "gathering" neighbor particles at each step, we "splatting" densities carried by particles to a uniform grid before marching. And at each cell of this uniform grid $\mathbf{M}$, we only record a boolean value indicating whether an iso-surface might be extracted inside this cell as shown in Figure 3. Based on this data structure, our ray marching is capable of skipping the low-density regions very fast.

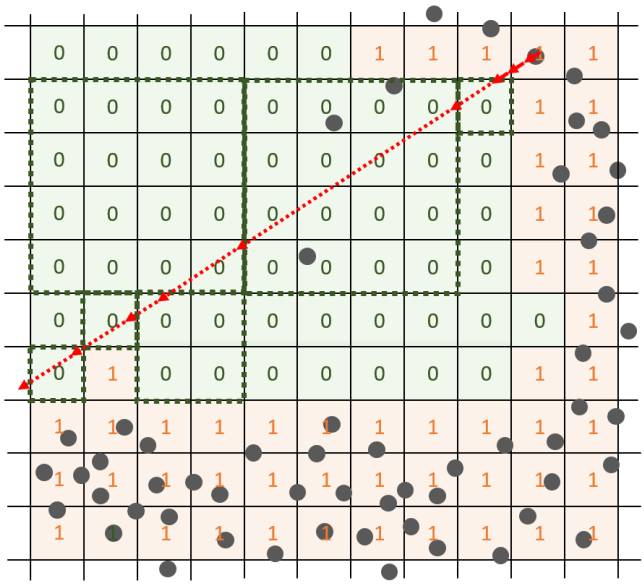


Figure 3: *The figure is a 2d cross-section of our 3d grid. The number in each cell indicates whether it is possible to contain an iso-surface, where 0 indicates that the cell's density is too low to find an iso-surface. With the assistance of the binary density grid, our algorithm is capable of using adaptive step lengths to quickly skip the empty spaces.*

To further improve the efficiency of querying the grid when marching, we compact multiple cells into one texel to generate a compact 3D density mask map for $\mathbf{M}$, which means only one texel-fetching instruction can return multiple boolean values. Furthermore, if the returned texel value is 0, all the cells covered by this texel can be quickly skipped.

Based on the above idea, in order to determine whether it is possible to extract a density iso-surface for each cell, the key issue is to quickly estimate the maximum density ρ_c^{max} of the cell c. And then the corresponding boolean value M_c of cell c in $\mathbf{M}$ can be defined as:

$$M_c = \begin{cases} 1, & \text{if } \rho_c^{max} \geq \sigma \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where σ is the iso-value defining fluid surfaces.

In our algorithm, ρ_c^{max} is defined as the maximum value of all $\rho_{\mathbf{p}}$ where $\mathbf{p}$ is a position located inside c. The density $\rho_{\mathbf{p}}$ can be computed from surrounding particles inside radius r:

$$\rho_{\mathbf{p}} = m \sum_j W\left(\mathbf{p} - \mathbf{x}_j, r\right), \tag{2}$$

where m is the particle mass, $\mathbf{x}_j$ is the position of surrounding particle j. It can be seen from Equation 2, $\rho_{\mathbf{p}}$ depends on two factors: the number of surrounding particles and their relative positions to p.

Computing ρ_c^{max} is not easy because it depends on too many variables. In our algorithm, we propose to use $\tilde{\rho}_c^{max}$ computed by Equation 3 as a conservative estimation for ρ_c^{max}:

$$\tilde{\rho}_c^{max} = m N_c W(0, r) \tag{3}$$

where N_c is a fixed value of cell c to represent the number of surrounding particles. Comparing with Equation 2, the ideas behind Equation 3 are: 1. Use 0 as all the relative positions of surrounding particles, which actually maximizes the value of $W()$; 2. Use a fixed number N_c to replace the varying number of surrounding particles inside radius r.

Now the only undetermined factor in Equation 3 is N_c. We set the size of cells to r so that for any location $\mathbf{p} \in c$, its surrounding particles involved in the computation of $\rho_{\mathbf{p}}$ must locate in the $3 \times 3 \times 3$ neighbor cells of c. Based on this, the easiest way to compute N_c is to access all the $3 \times 3 \times 3$ cells and sum the particle count in each cell together. But this mechanism is quite time consuming because each cell needs to access its 26 neighbors.

In this paper, we propose a fast way to compute N_c. The basic idea is to convert the sum operation to convolution. Let's take 2D as an example to illustrate this conversion:

$$N_c = \sum_x \sum_y N_{(x,y)} = \begin{bmatrix} N_{(x-1,y+1)} & N_{(x,y+1)} & N_{(x+1,y+1)} \\ N_{(x-1,y)} & N_{(x,y)} & N_{(x+1,y)} \\ N_{(x-1,y-1)} & N_{(x,y-1)} & N_{(x+1,y-1)} \end{bmatrix} \star \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \tag{4}$$

where N is an array which stores particle counts per cell. In fact, $\tilde{\rho}_c^{max}$ obtained using the above method is too conservative though it is absolutely correct. We can replace the convolution kernel in Equation 4 with an isotropic spatial separable convolution kernel K:

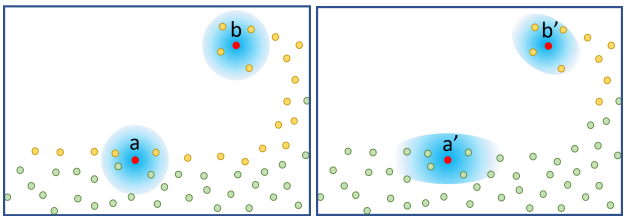
$$K = \begin{bmatrix} b^2 & ab & b^2 \\ ab & a^2 & ab \\ b^2 & ab & b^2 \end{bmatrix} = \begin{bmatrix} b \\ a \\ b \end{bmatrix} \times \begin{bmatrix} b & a & b \end{bmatrix}, \tag{5}$$

which means Equation 4 can be calculated in two steps. Since particles in the center cell have a higher probability of being searched, we set $b < a < 1$ to achieve higher marching efficiency.

4.2. Particles Classification

To reduce unnecessary computations in empty-space between the camera and the fluid region, we generate the depth map $\mathbf{D}_{all}$ of all particles in advance as the entry point for each ray, just like [XZY17]. However, the entrance represented by $\mathbf{D}_{all}$ may be very far from the final iso-surfaces when there are a lot of splash particles in the scene, which means unnecessary computations in a lot of low-density regions are still unavoidable. In particular, when our binary density grid is so conservative that most of these regions is marked as 1.

In order to maintain a more stable performance, we also generate approximate exits where the ray can quickly escape when the number of steps exceeds a threshold. The basic idea is to classify the fluid particles into splash (low density) and aggregated (high density) particles according to their densities. Then a depth map $\mathbf{D}_{agg}$ is generated from aggregated particles. $\mathbf{D}_{agg}$ provides approximate potential stop positions because the region where the aggregated particles are located is more likely to have a higher density.



(a) Isotropic (b) Anisotropic

Figure 4: *Comparison between density computed by isotropic (a) and anisotropic kernel function (b). Particles in yellow and green indicate splash and aggregated particles respectively.*

Based on the above analysis, classifying the particles accurately is the key for the algorithm to work. We first try to classify particles simply according to their density. In SPH, the density ρ_i of particle i with mass m_i at location $\mathbf{x}_i$ can be computed as:

$$\rho_i = \sum_j m_j W\left(\mathbf{x}_i - \mathbf{x}_j, r\right), \tag{6}$$

where $\mathbf{x}_j$ is the position of neighbor particle j and W is the smoothing kernel function:

$$W\left(\mathbf{x}_i - \mathbf{x}_j, r\right) = \frac{s}{r^3} P\left(\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|}{r}\right), \tag{7}$$

where s is a scaling factor, r is the smoothing radius, and P is a symmetric decaying spline with finite support.

However, as illustrated in Figure 4, particle a on the fluid surface has a great chance to be incorrectly tagged as splash particle instead of an aggregated particle. Because Equation 7 actually is an isotropic kernel function, from which the density value computed might be too low.

In order to address the above issue, we analyzed the different characteristics of particle a and splash particle b. The distribution of aggregated particles located on the fluid surface is more consistent (all along the fluid surface) than the disordered splash particles. Therefore, we replace W with the anisotropic kernel function proposed in [YT13], whose basic idea is that neighbor particles with more consistent orientation should be given higher weights. The anisotropic kernel function is expressed as:

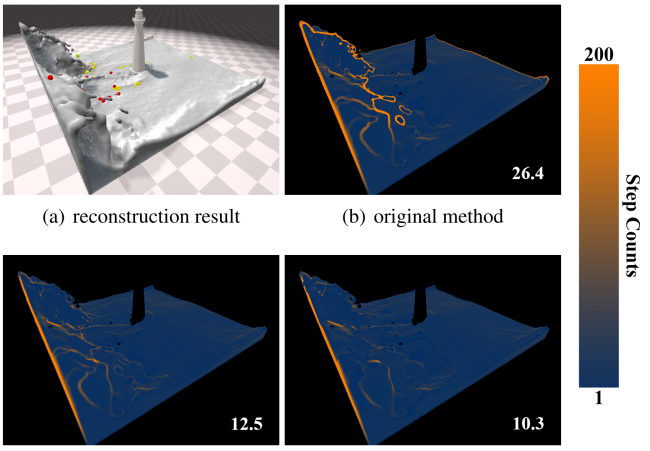
$$W\left(\mathbf{x}_i - \mathbf{x}_j, \mathbf{G}_j\right) = s\left|\det\left(\mathbf{G}_j\right)\right| P\left(\left\|\mathbf{G}_j(\mathbf{x}_i - \mathbf{x}_j)\right\|\right); \tag{8}$$

$$\mathbf{G}_j = \frac{1}{r}\mathbf{R}\widetilde{\Sigma}^{-1}\mathbf{R}^T. \tag{9}$$

where the linear transform matrix $\mathbf{G}$ of neighbor particle j, for rotating and stretching $\mathbf{x}_i - \mathbf{x}_j$, is constructed by applying the weighted version of Principal Component Analysis[KC03] to $\mathbf{x}_j$. $\mathbf{R}$ is the rotation matrix and Σ is the diagonal matrix obtained by the singular value decomposition (SVD) of the covariance matrix. We used Yu and Turk's method to obtain modified matrix $\widetilde{\Sigma}$ to prevent extreme deformations.

Since $\mathbf{G}$ is only used to calculate the determinant and L1 Norm, Equation 9 can be further simplified:

$$\mathbf{G}_j = \frac{1}{r}\widetilde{\Sigma}^{-1}\mathbf{R}^T. \tag{10}$$



(a) reconstruction result (b) original method

(c) mask (d) mask + skip

Figure 5: *A comparison of step counts required for different algorithm. (a) Reference image. (b) Step counts required by [XZY17]. (c) Step counts required by our method presented in Section 4.1. (d) Step counts required by our method presented in Section 4.1 and Section 4.2.*

As shown in the Figure 4, the anisotropic kernel function will produce a larger density value for a', and thus the possibility to incorrectly tag a' as splash particle is reduced.

4.3. Adaptive Ray Casting Resolution

In addition to reducing marching step counts of each ray, we also tried to reduce the number of rays cast. Note an important fact that it is not necessary to cast a ray for each pixel if it belongs to some screen areas covered by flat fluid surfaces. We can execute ray marching on low resolution for these regions so as to reduce the number of rays cast, and then use interpolation to up-sampling back to full resolution.

Based on the above idea, we divide the screen into equal-sized tiles. Before ray marching, we calculate the variance of depths $\mathbf{Var}(\mathbf{D}_{all})$ for each tile to determine whether it is flat enough. We use two variance thresholds ε_1 and ε_2 ($\varepsilon_1 < \varepsilon_2$) to divide the flatness into three levels. More specifically, if $\mathbf{Var}(\mathbf{D}_{all}) > \varepsilon_2$ then rays in the tile are cast in full resolution, if $\varepsilon_1 < \mathbf{Var}(\mathbf{D}_{all}) \leq \varepsilon_2$ then rays in the tile are cast in half resolution, and $\mathbf{Var}(\mathbf{D}_{all}) \leq \varepsilon_1$ means quarter resolution.

4.4. Adaptively Blending Normals

For pixel(i, j), the ray marching method can help us attain more precise position $\mathbf{x}_{ij}$ by finding iso-surfaces. And the normal $\mathbf{N}_{ij}$ on $\mathbf{x}_{ij}$ is also critical to the fluid reconstruction quality. To get the normal quickly, we first calculate normal $\mathbf{N}_{ij}^{Screen}$ from the depth map $\mathbf{D}_{all}^{smooth}$ that is generated by smoothing $\mathbf{D}_{all}$ with Bilateral Gaussian Filter.

And there are two issues that need to be addressed. Firstly, the position $\mathbf{p}_{ij}$ from depth map $\mathbf{D}_{all}$ is just an entrance of the ray and

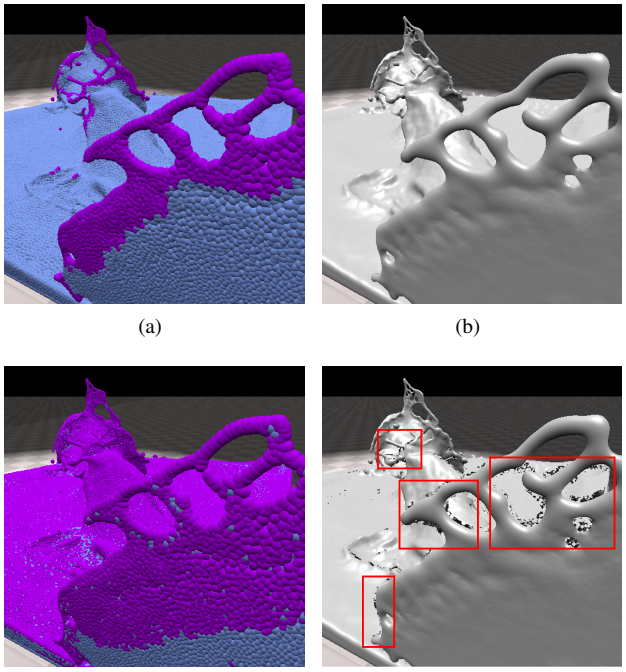


Figure 6: *A comparison between particle classification results generated by isotropic and anisotropic kernel function. (a) Classification results by anisotropic function. (b) Surfaces reconstructed by anisotropic function. (c) Classification results by isotropic function. (d) Surfaces reconstructed by isotropic function.*

maybe far from $\mathbf{x}_{ij}$, especially in those areas penetrated by the ray. Secondly, screen space filtering can be affected by camera distance. When particles are too close to the camera, noticeable spherical artifacts will be difficult to be smoothed by a tolerable filter radius.

To solve the above two problems of screen space normals, we also estimate the normal $\mathbf{N}_{ij}^{Object}$ by searching for neighbor particles in object space.

$$\mathbf{N}_{ij}^{Object} = normalize\left(\sum_k \nabla W\left(\mathbf{x}_{ij} - \mathbf{x}_k, r\right)\right), \tag{11}$$

where $\mathbf{x}_k$ is the position of the neighbor particles centered at the iso-surface position $\mathbf{x}_{ij}$. And $\mathbf{N}_{ij}^{Object}$ is only used to correct $\mathbf{N}_{ij}^{Screen}$, so only a small search radius is needed without a large performance overhead.

For the two problems, we design two corresponding adaptive weights w_1 and w_2 to blend $\mathbf{N}_{ij}^{Screen}$ and $\mathbf{N}_{ij}^{Object}$:

$$w_1 = A\left\|\mathbf{x}_{ij} - \mathbf{p}_{ij}\right\|; \tag{12}$$

$$w_2 = e^{BR_{ij}}; \tag{13}$$

$$w = min(w_1 w_2, 1); \tag{14}$$

$$\mathbf{N}_{ij} = w\mathbf{N}_{ij}^{Object} + (1-w)\mathbf{N}_{ij}^{Screen}, \tag{15}$$

where A and B are the factors to adjust these two weights respectively. And R_{ij} is the screen-space size of the particle in the pixel(i, i), which can be derived from eye-space depth value $z_{ij}^{(Eye)}$ with the parameters of the camera:

$$R_{ij} = \frac{hd}{2\left|z_{ij}^{(Eye)}\right|\tan(\alpha/2)}, \tag{16}$$

where h is the vertical resolution of the screen, d is particle's size in object space and α is the camera's field of view angle. With our adaptive normal blending weight, we can correct the position mismatch and close particle spherical artifacts while obtaining smooth surface reconstruction results at a small cost.

	Cam Break 1024×1024	Cam Break 1920×1920	Still Water 1920×1920
Full Resolution	8.98ms	19.80ms	17.47ms
Half Resolution	5.91ms	15.29ms	6.31ms
Quarter Resolution	4.11ms	10.78ms	5.38ms
Adaptive Resolutionn	5.10ms	13.42ms	4.55ms

Table 1: *A performance comparison between different ray casting resolutions.*

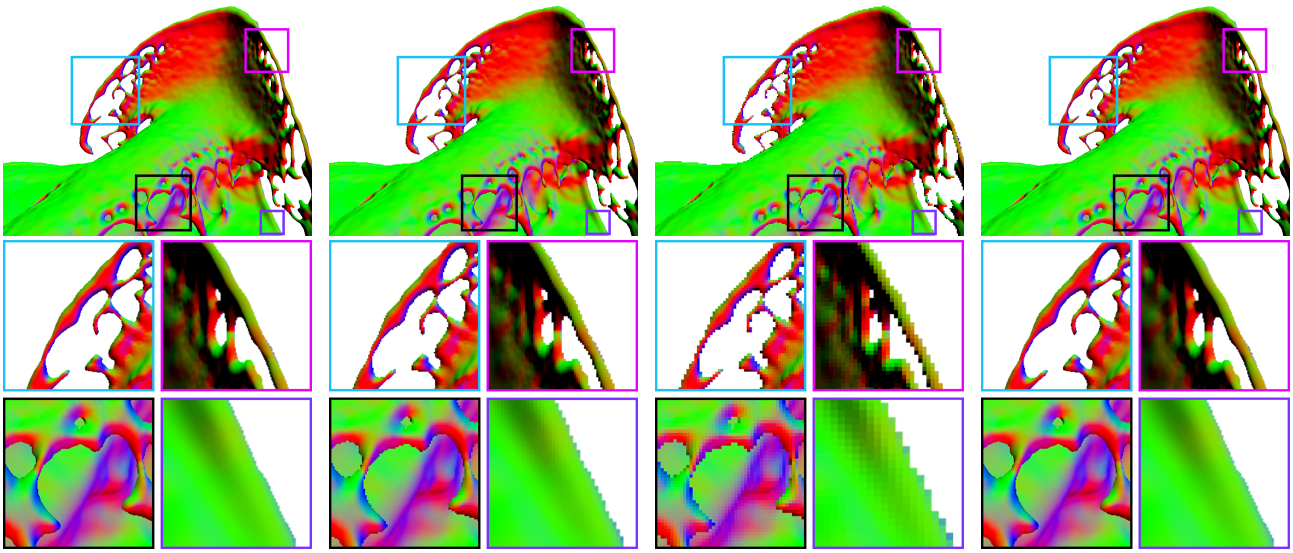
5. Experimental Results

Our adaptive reconstruction method is implemented on a PC with a NVIDIA GeForce RTX 2070 GPU. The movements of particles are generated by the algorithm presented in [MM13], whose overhead is not included in the performance results in our experiments.

In order to prove that our adaptive mechanism has the ability to skip large empty space, we compared our method with [XZY17]. Figure 5 illustrates marching step counts required by [XZY17] and our method. It can be seen that [XZY17] (Figure 5(b)) needs more steps than our methods (Figure 5(c) and 5(d)), especially in the regions covered by splashes. The average number of steps are also illustrated at the right-down corn of each image. And these data suggest that our method can reduce the number of marching steps by 50%.

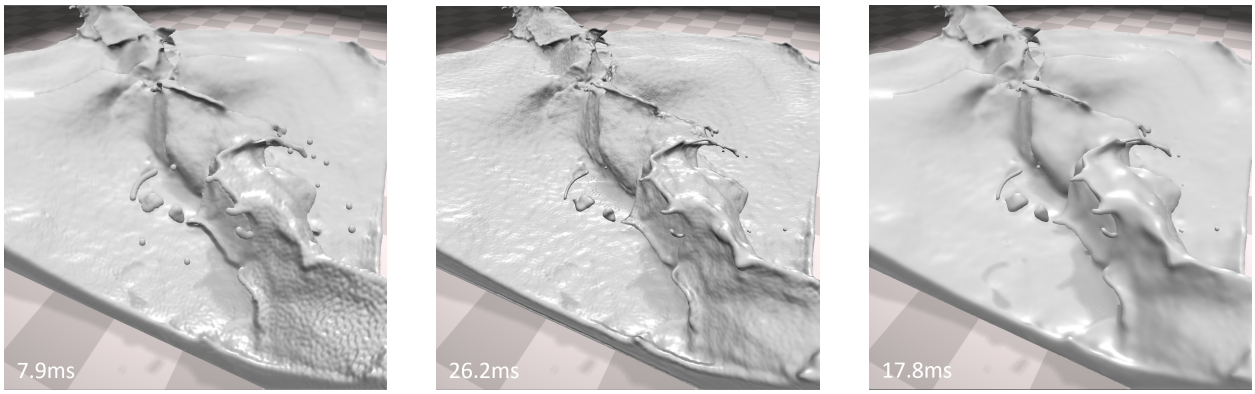
To demonstrate the necessity of using anisotropic kernel function to compute density in the process of classifying particles, we compare the classification and reconstruction results of isotropic and anisotropic kernel function in Figure 6. It can be seen from Figure 6(a) and 6(c), the anisotropic kernel function produces much better classification results than isotropic function. If the outermost particles of the fluid surfaces are not identified as aggregates, $\mathbf{D}_{all}$ will be generated by aggregated particles from below the surface. It means a wrong skip position will be provided for ray marching. Figure 6(b) and 6(d) illustrate the iso-surfaces extracted by anisotropic and isotropic kernel function respectively. It can be seen from Figure 6(d), the regions marked by red boxes present some error reconstruction results.

In addition to reducing the number of steps per ray, we also cast



(a) Full Resolution (b) Half Resolution (c) Quarter Resolution (d) Adaptive Resolution

Figure 7: *A comparison between the reconstruction quality under different ray casting resolution.*



(a) Narrow range Filter[TY18] (b) the method of Xiao et al.[XZY17] (c) Our Method

Figure 8: *Comparison between reconstruction quality.*

fewer rays in the screen areas covered by flat fluid surfaces. Unlike simply reducing the entire resolution (Figure 7(b) and 7(c)), our adaptive method can achieve a better balance between performance and quality. As illustrated in Table 1, the performance of our method is similar to half resolution, but it achieves almost the same reconstruction quality as full-resolution.

In order to test the reconstruction quality of our adaptive normal blending method, we compare our algorithm with narrow range filter method[TY18] and the method of Xiao et al.[XZY17]. As illustrated in Figure 8, [TY18] produces fairly smooth surfaces for particles not close to the eye. But for the particles close to eye, the surfaces reconstructed by [TY18] present bumpy artifacts because these areas need a quite big smoothing kernel size which is unbearable for real-time application. [XZY17] can obtain more natural droplet details. But it needs more computations to smooth the normal vectors in object space. Our method is capable of achieving smooth fluid surfaces while keeping the overhead in an acceptable level.

Finally, we present the rendering results of our method for different scenarios. Figure 9 shows an image sequence of a dam break scenario with lots of splashes. It can be seen that our algorithm can produce smooth fluid surfaces while preserving a lot of very detailed splashes and sharp corners. The average reconstruction time for this scene is about 16ms per frame for 912k particles. More sce-

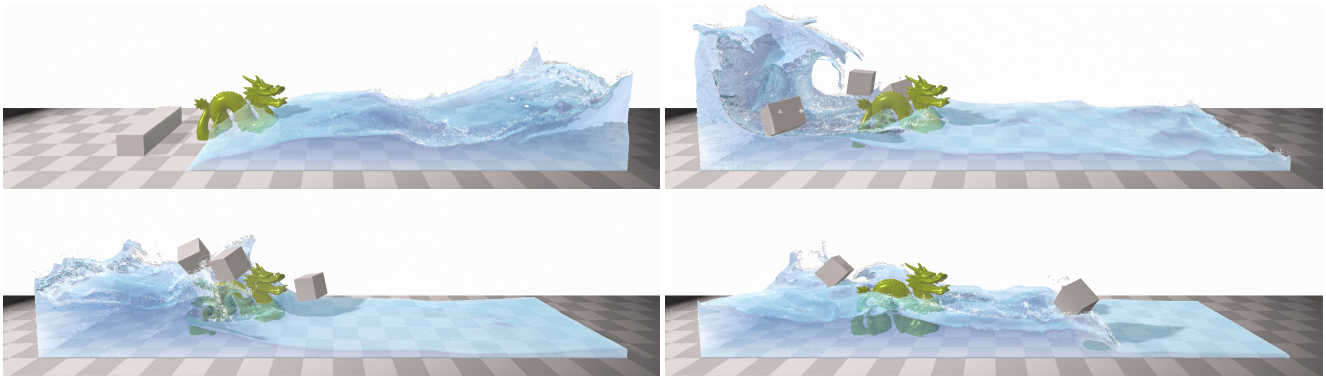
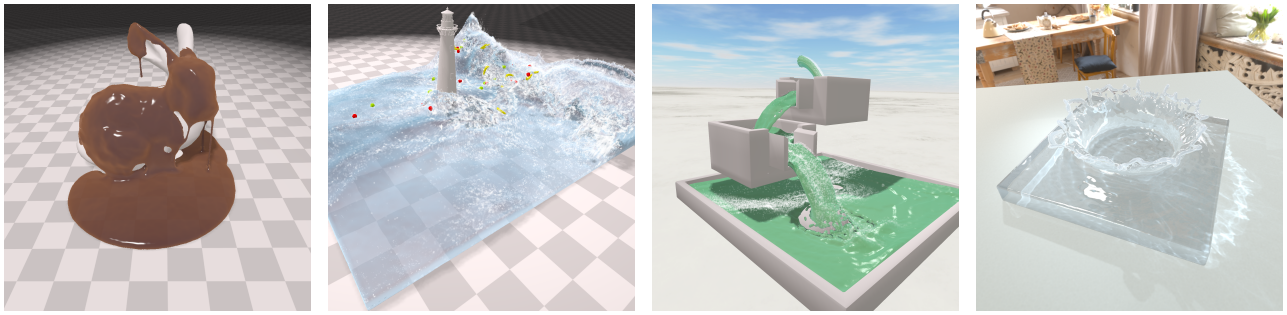


Figure 9: *A dam break scenario with 912k particles.*



(a) Melting chocolate (b) Waves with lots of splashes (c) Spring (d) Crown-like splashes

Figure 10: *Our algorithm is applicable for different kinds of fluids.*

	Stanford Dragon	Double Cam Break	Surfing
Number of Particles	912k	636k	443k
Generate Density Mask	0.031	0.011	0.023
Draw high-density Particles	2.122	1.572	1.025
Draw low-density Particles	0.225	0.432	0.106
Smooth Depth Map	3.047	3.415	3.512
Genrate Screen Space Noraml	0.059	0.040	0.064
Evaluate Tiles	0.038	0.032	0.033
Ray marching	10.766	12.333	7.560
total	**16.318**	**17.835**	**12.323**

Table 2: *Computation time (ms) for each pass in different scenes for fluid surface reconstruction.*

narios with step-by-step calculation times can be found in Table 2. In order to prove that our algorithm is applicable for different kind of fluids, Figure 10 demonstrates four different scenes rendered by our algorithm, including the viscous chocolate pulling details (Figure 10(a)), waves with lots of splashes (Figure 10(b)), spring water (Figure 10(c)) and crown-like splashes (Figure 10(d)).

6. Conclusions

In this paper, we present a set of adaptive mechanisms to reconstruct hight-quality fluid surfaces in real-time. In order to reduce the number of steps required for each ray, we propose to use adaptive step lengths computing from a binary density grid and offer an approximate exit obtained by classifying out aggregated particles. We also try to reduce the number ray cast by employing an adaptive resolution. In addition to addressing the performance issue, an adaptive normal blending method is proposed to improve the reconstruction quality.

In the future, we will also explore more potential approaches about adaptive step lengths, such as adjusting the step lengths according to the density of the previous step. On the other hand, our binary density grid might be based on the view frustum and also be used to estimate the refraction and absorption of the ray in the fluid.

7. Acknowledgement

This work was supported by the National Key Project of China (Project Number GJXM92579) and National Natural Science Foundation of China (Project Number 61972271).

References

[AIA*12] AKINCI, NADIR, IHMSEN, MARKUS, AKINCI, GIZEM, et al. "Versatile rigid-fluid coupling for incompressible SPH". *ACM Transactions on Graphics (TOG)* 31.4 (2012), 1–8 2.

[APKG07] ADAMS, BART, PAULY, MARK, KEISER, RICHARD, and GUIBAS, LEONIDAS J. "Adaptively sampled particle fluids". *ACM SIGGRAPH 2007 papers*. 2007, 48–es 2.

[BK15] BENDER, JAN and KOSCHIER, DAN. "Divergence-free smoothed particle hydrodynamics". *Proceedings of the 14th ACM SIGGRAPH/Eurographics symposium on computer animation*. 2015, 147–155 2.

[Bli82] BLINN, JAMES F. "A generalization of algebraic surface drawing". *ACM transactions on graphics (TOG)* 1.3 (1982), 235–256 2.

[BSS*18] BIEDERT, TIM, SOHNS, J-T, SCHRÖDER, SIMON, et al. "Direct raytracing of particle-based fluid surfaces using anisotropic kernels". *Proceedings of the Symposium on Parallel Graphics and Visualization*. 2018, 1–12 2.

[DG96] DESBRUN, MATHIEU and GASCUEL, MARIE-PAULE. "Smoothed particles: A new paradigm for animating highly deformable bodies". *Computer Animation and Simulation'96*. Springer, 1996, 61–76 1.

[Gre10] GREEN, SIMON. "Screen space fluid rendering for games". *Proceedings for the Game Developers Conference*. Moscone Center San Francisco, CA. 2010 1, 2.

[IKM16] IMAI, TAKUYA, KANAMORI, YOSHIHIRO, and MITANI, JUN. "Real-time screen-space liquid rendering with complex refractions". *Computer Animation and Virtual Worlds* 27.3-4 (2016), 425–434 1, 3.

[KC03] KOREN, YEHUDA and CARMEL, LIRAN. "Visualization of labeled data using linear transformations". *IEEE Symposium on Information Visualization 2003 (IEEE Cat. No. 03TH8714)*. IEEE. 2003, 121–128 5.

[KS14] KANG, NAHYUP and SAGONG, DONGHOON. "Incompressible SPH using the divergence-free condition". *Computer graphics forum*. Vol. 33. 7. Wiley Online Library. 2014, 219–228 2.

[MM13] MACKLIN, MILES and MÜLLER, MATTHIAS. "Position Based Fluids". *ACM Trans. Graph.* 32.4 (July 2013). ISSN: 0730-0301. DOI: 10.1145/2461912.2461984. URL: https://doi.org/10.1145/2461912.2461984 2, 6.

[MSD07] MÜLLER, MATTHIAS, SCHIRM, SIMON, and DUTHALER, STEPHAN. "Screen space meshes". *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 2007, 9–15 1, 2.

[RCSW14] REICHL, FLORIAN, CHAJDAS, MATTHÄUS G, SCHNEIDER, JENS, and WESTERMANN, RÜDIGER. "Interactive Rendering of Giga-Particle Fluid Simulations." *High Performance Graphics*. Citeseer. 2014, 105–116 2.

[TY18] TRUONG, NGHIA and YUKSEL, CEM. "A narrow-range filter for screen-space fluid rendering". *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1.1 (2018), 1–15 1, 2, 7.

[vdLGS09] Van der LAAN, WLADIMIR J, GREEN, SIMON, and SAINZ, MIGUEL. "Screen space fluid rendering with curvature flow". *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. 2009, 91–98 1, 2.

[XZY17] XIAO, XIANGYUN, ZHANG, SHUAI, and YANG, XUBO. "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*. 2017, 1–8 1–7.

[YT13] YU, JIHUN and TURK, GREG. "Reconstructing surfaces of particle-based fluids using anisotropic kernels". *ACM Transactions on Graphics (TOG)* 32.1 (2013), 1–12 2, 5.

[ZB05] ZHU, YONGNING and BRIDSON, ROBERT. "Animating sand as a fluid". *ACM Transactions on Graphics (TOG)* 24.3 (2005), 965–972 2.