A Narrow-Range Filter for Screen-Space Fluid Rendering

Nghia Truong
University of Utah
ttnghia@cs.utah.edu

Cem Yuksel
University of Utah
cem@cemyuksel.com

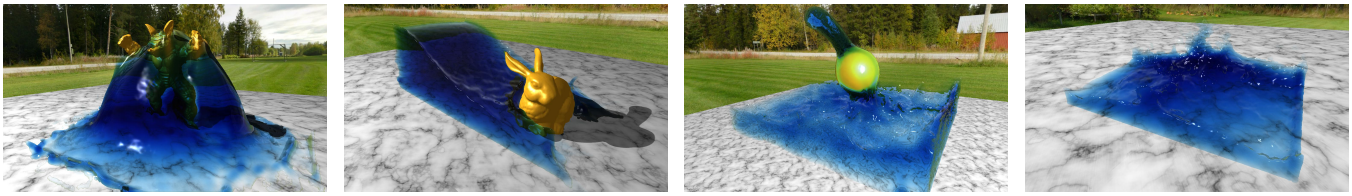


Figure 1: *Example frames from different particle-based fluid simulations rendered using our screen-space fluid rendering method.*

ABSTRACT

We introduce a simple screen-space filtering technique for real-time rendering of particle-based fluid simulations. Starting with a depth-map generated directly from the particle data, our new filter formulation smooths the depth-map by considering the depth values in a narrow range. The depth values outside of this range are carefully handled to achieve the desired surface shape near discontinuities. The simplicity of our formulation leads to a computationally efficient filter. We present examples with complex particle-based fluid simulations and provide comparisons, clearly showing that our filter provides improved surface quality in terms of surface smoothness and preserving boundaries near discontinuities, as compared to prior filtering methods.

CCS CONCEPTS

• **Computing methodologies** → **Rendering**;

KEYWORDS

fluid rendering, real-time rendering, screen-space filtering, particle-based simulation

ACM Reference Format:
Nghia Truong and Cem Yuksel. 2018. A Narrow-Range Filter for Screen-Space Fluid Rendering. In *Proceedings of Interactive 3D Graphics and Games (i3D2018).* ACM, New York, NY, USA, Article 17, 8 pages. https://doi.org/10.1145/3203201

1 INTRODUCTION

Fluid simulations are popular in computer graphics. For video games and other real-time graphics applications, particle-based simulation methods are often preferred, as they are fast and flexible. While the computation powers of current CPUs and GPUs allow simulating fluids with a relatively large number of Lagrangian particles at high frame-rates for producing visually complex animations, there is still room for improvement when it comes to efficiently rendering the fluids represented by the simulated particles.

The main difficulty in rendering the results of particle-based fluid simulations has been mesh generation, which can be computationally expensive. Therefore, recent real-time rendering methods for particle-based fluids use screen-space filtering techniques that begin with directly rendering the particle data. The resulting surface is then filtered to produce the desired smooth fluid surface. It is the properties of the screen-space filter that predominantly determines the quality of the final rendering result.

In this paper, we introduce a novel filtering technique for rendering particle-based fluid simulations on the GPU. Our filter directly uses the depth values within a narrow range and carefully handles the depth values outside of this narrow range to produce a smooth surface with a desired curvature near discontinuities. Example frames from different fluid simulations rendered using our method are shown in Figure 1. Our filter formulation not only leads to improved surface quality but also provides improved performance as compared to prior screen-space fluid rendering methods. It is also simple and easy to implement. We provide detailed comparisons that highlight the qualitative improvements provided by our method.

2 PRIOR WORK

While there is a large body of work in computer graphics on particle-based fluids simulations [Desbrun and Gascuel 1996; Macklin and Müller 2013; Müller et al. 2007; Weiler et al. 2016], real-time fluid rendering methods, particularly screen-space techniques, have received relatively little attention.

Earlier methods for real-time particle-based fluid rendering generate polygonal meshes from the particle data using screen-space techniques [Müller et al. 2007] or marching cubes [Rosenberg and Birdwell 2008]. As mesh generation can be computationally expensive, these methods do not provide the most efficient alternatives for real-time fluid rendering. Using metaballs with ray-isosurface intersection [Zhang et al. 2008] eliminates the need for explicitly generating a surface mesh and it also avoids the grid discretization

i3D2018, May 2018, Montreal, Quebec, Canada

ACM ISBN 2577-6193/2018/5-ART17…$15.00
https://doi.org/10.1145/3203201

artifacts for mesh generation, but produces thick surfaces because of the large metaball size needed for achieving surface smoothness.

More recent methods avoid surface mesh generation using an entirely different screen-space rendering approach: screen-space filtering of rendered particle data. These methods begin with creating a depth-map for the front surface of the fluid by directly rendering fluid particles as spheres or ellipsoids, using either rasterization-based techniques [Cords and Staadt 2009; Green 2010; Imai et al. 2016; van der Laan et al. 2009] or GPU-based ray-casting [Goswami et al. 2010; Reichl et al. 2014; Xiao et al. 2017]. Anisotropic kernels [Yu and Turk 2013] can be used for improving the final surface quality near thin regions by rendering each particle as elongated ellipsoids. Then, the resulting depth-map is smoothed via different types of image-based filtering operations to achieve a more fluid-like surface appearance. Finally, the filtered depth-map is used for rendering the fluid with reflections and refractions. Obviously, the properties of the filtering technique is the primary factor in the resulting quality of the final image produced by these techniques.

Depending on the position of the camera, the screen-space size of a fluid particle can be large. Therefore, screen-space filtering may require large filter sizes and multiple iterations to produce high-quality results. Unfortunately, computing large 2D filters can be expensive. Therefore, it is common practice to use separable filters that can be represented as a product of two 1D filters. Separable filters require two 1D filter passes to perform a single 2D filter operation, but since large 1D filters can be computed more efficiently, they lead to improved performance. That is why, even when the filter kernel is not separable, it is common practice to use a separable filter approximation, which can lead to visual artifacts.

Binomial filters [Cords and Staadt 2009; Müller et al. 2007] and standard Gaussian filters are fast, since they can be implemented as separable filters; however, they over-smooth the fluid surface and fail to preserve sharp boundaries. Depending on the camera angle, entirely unrelated parts of the fluid can be filtered together, based on their screen-space positions, resulting fluid surface appearance that can be a poor representation of the underlying particle-based fluid data. Bilateral Gaussian filters [Green 2010] can preserve the fluid boundaries, but they lead to excessive flattening near discontinuities. Furthermore, approximately flat fluid surfaces appear rough and noisy when viewed from near grazing angles. Moreover, bilateral Gaussian filters are not separable and using separable filter approximations with bilateral Gaussian filters lead to severe visual artifacts in the form of axis-aligned streaks near most fluid surfaces details. Still, because of the cost of computing large 2D bilateral Gaussian filters, the separable filter approximation is often preferred, even with its visual artifacts [Green 2010].

Screen-space curvature flow (SSCF) [van der Laan et al. 2009] provides superior surface quality as compared to bilateral Gaussian filters. SSCF iteratively solves a high order curvature flow PDE that is used for smoothing the depth-map. However, it typically requires numerous iterations (~100 or even more) to produce a smooth surface. Yet, since SSCF uses a small 3×3 filter, each pass can be computed efficiently. On the other hand, the level of smoothness depends on the number of passes and it can vary depending on the distance of the particles to the camera. Therefore, achieving a consistent surface smoothness requires dynamically adjusting the iteration count per pixel [Bagar et al. 2010]. Nonetheless, the need for a large number of filter iterations makes SSCF relatively expensive and the explicit integration scheme of the high order PDE may cause numerical instability, leading to some minor visual artifacts in the form of apparent noise near discontinuities.

More recently, the total-variation-based image de-noising method has been adapted to screen-space filtering for rendering particle-based fluids [Reichl et al. 2014]. This approach also uses a large number of iterations, so its performance is similar to SSCF, but it provides a better preservation of sharp fluid details as compared to SSCF. Furthermore, the amount of smoothing does not depend on the number of iterations. Another recent work includes an alternative approach using plane fitting for filtering [Imai et al. 2016]. Plane fitting can produce relatively faster results with only a few iterations and provides improved surface quality as compared to bilateral Gaussian filters by avoiding flattening near discontinuities. On the other hand, the surfaces produced by plane fitting tend to be noisier than SSCF.

The filtering method we introduce in this paper is computationally efficient, as it works with few iterations, and it provides smoother surfaces with well-preserved details near discontinuities.

3 NARROW-RANGE FILTER FORMULATION

Similar to other screen-space fluid rendering methods, we begin with generating a depth-map from the particle data. We rasterize the particles as spheres or ellipsoids (using anisotropic kernels [Yu and Turk 2013]) and store a (negative) eye-space depth value z_i for each pixel i on the screen (i.e. smaller z_i values correspond to points further away from the camera). The depth-map is then smoothed using our narrow-range filter, which is typically applied a few times.

We use a Gaussian filter at the core of our narrow-range filter, though it is possible to use a different smoothing filter kernel. Similar to a typical convolution filter, our narrow-range filter updates the depth values based on the values of the neighboring pixels for smoothing the screen-space surface representation in the depth-map. The filter kernel size is defined in world-space, so the screen-space kernel size is determined from the depth of each pixel. For each pixel only the neighboring depth values within a narrow range are directly used. The values outside of this narrow range are carefully handled to achieve the desired surface smoothness and curvature near discontinuities. This treatment eliminates the problems of simple Gaussian filters and bilateral Gaussian filters.

3.1 Limiting the Depth Range

The output depth value z'_i for pixel i computed using our narrow-range filter can be written as

$$z'_i = \frac{\sum_j \omega_{ij} \, f(z_i, z_j)}{\sum_j \omega_{ij}} \,, \tag{1}$$

where ω_{ij} is the filter weight and f is our *clamping function* that is defined as

$$f(z_i, z_j) = \begin{cases} z_j, & \text{if } z_j \geq z_i - \delta \\ z_i - \mu, & \text{otherwise,} \end{cases} \tag{2}$$

where δ and μ are user-defined parameters, such that $\mu \leq \delta$. This clamping function merely returns the depth value z_j, if it is within the permitted range; otherwise, it returns a clamped value $z_i - \mu$.

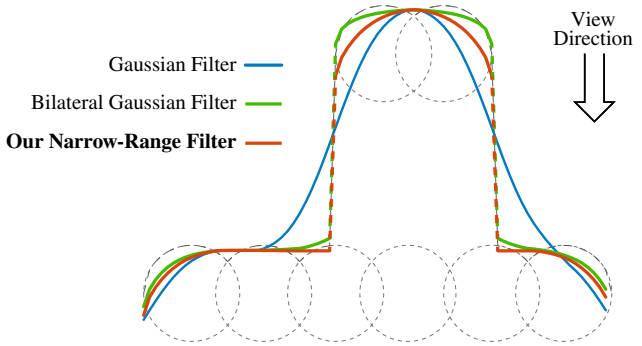


Figure 2: *A simple 2D example comparing Gaussian, bilateral Gaussian, and our narrow-range filters that are used for smoothing the surface defined by eight particles (circles) and the vertical view direction. Notice that the Gaussian filter generates a smooth surface, but filters across these two distinct groups of particles, effectively ignoring the discontinuities and producing a view-dependent surface that is not faithful to the underlying particle representation. Bilateral Gaussian filter captures the sharp change in depth, but it flattens the surface formed by the two particles at the top. Our narrow-range filter, on the other hand, produces natural curved edges for the top surface, thanks to the clamping function. Bilateral Gaussian filter also distorts the surface for the particles at the bottom. Notice that the green bilateral Gaussian filter curve bends upwards near the discontinuities. Our narrow-range filter, however, produces the desirable flat surface near the discontinuities for the particles at the bottom.*

Its purpose is to prevent blending the filtered pixel depth with surfaces that are too far behind (determined by δ). However, unlike a bilateral Gaussian filter that would effectively ignore pixels j with depth values $z_j \ll z_i$, which causes excessive flattening near discontinuities, our clamping function takes into account all pixels j with $z_j < z_i$, but clamps their depth values used in filtering. This makes the filter react to depth discontinuities and produce desirable curved edges. The desired curvature near discontinuities is controlled by the μ parameter.

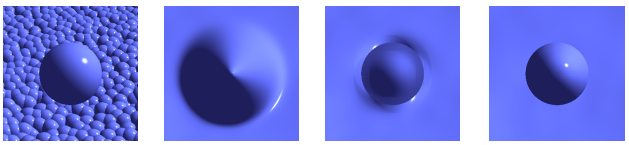
The importance of clamping is demonstrated with a simple 2D example shown in Figure 2 and a simple 3D example in Figure 3. In both examples Gaussian filter ignores discontinuities and bilateral Gaussian filter flattens the edges of the foreground particles. The clamping function of our narrow-range filter avoids flattening and produces desirable curved boundaries for the foreground particles.

We must also prevent the foreground particles from distorting the surface behind them, as shown in Figures 2 and 3. We achieve this by eliminating the contributions of pixels j that belong to a surface that is much closer to the camera than the filtered pixel (i.e. $z_j \gg z_i$). Thus, we simply ignore pixels j with $z_j > z_i + \delta$ by computing the filter weights ω_{ij} using

$$\omega_{ij} = \begin{cases} 0, & \text{if } z_j > z_i + \delta \\ G(\mathbf{p}_i, \mathbf{p}_j, \sigma_i), & \text{otherwise,} \end{cases} \tag{3}$$

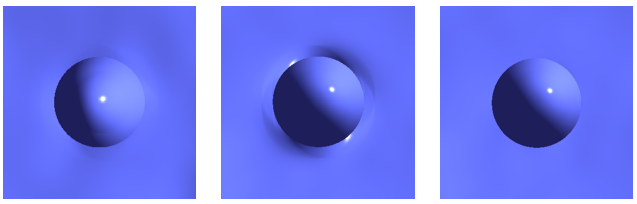
where $\mathbf{p}_i$ and $\mathbf{p}_j$ are the 2D pixel coordinates, G is the Gaussian function, and σ_i is its standard deviation parameter, such that

$$G(\mathbf{p}_i, \mathbf{p}_j, \sigma_i) = e^{-|\mathbf{p}_j - \mathbf{p}_i|^2 / 2\sigma_i^2}. \tag{4}$$



Surface Before Filtering | Gaussian Filter | Bilateral Gaussian Filter | **Narrow-Range Filter**

Figure 3: *A simple 3D example comparing Gaussian, bilateral Gaussian, and our narrow-range filters. The particle data forms a flat surface in the background and a drop in the foreground. Notice that using a simple Gaussian filter forms a smooth but incorrect surface by connecting the drop in the foreground to the surface behind it. Bilateral Gaussian filter preserves the discontinuities but causes excessive flattening for the drop in the foreground near its boundaries. Our narrow-range filter, however, produces a desirable curved boundary for the drop and preserves discontinuities. Furthermore, bilateral Gaussian filter also distorts the background surface near the discontinuities. Our narrow-range filter, on the other hand, is able to produce a flat surface for the background particles near the discontinuities.*



(a) Perpendicular Surface | (b) Inclined Surface | (c) Bias Correction

Figure 4: *An example with a drop in the foreground and an approximately flat surface in the background, showing the importance of bias correction: (a) background surface is perpendicular to the view direction, so no bias correction is needed, (b) background surface is inclined, causing deformations near discontinuities, and (c) bias correction fixes the distortion on the inclined surface near discontinuities.*

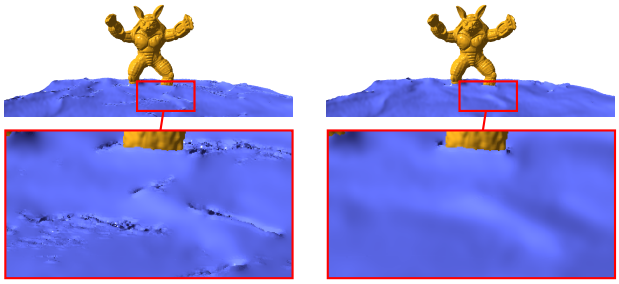
The screen-space filter kernel size at pixel i is taken as $3\sigma_i$, which is computed using

$$\sigma_i = \left\lceil \frac{H\,\sigma}{2\,|z_i|\tan(\alpha/2)} \right\rceil, \tag{5}$$

where H is the vertical resolution of the screen, σ is the world space filter size (fixed value), and α is the camera's field of view angle.

3.2 Bias Correction

The depth range limiting with Equation 3 works well for flat background surfaces that are perpendicular to the view direction, as shown in Figure 4a. However, if the background surface is viewed from a different angle, as in Figure 4b, depth range limiting with Equation 3 leads to distortion near discontinuities. The reason for this distortion is that Equation 3 simply ignores the part of the surface that is occluded by a foreground surface, which causes bias



(a) Static Range (δ) (b) Dynamic Range ($\delta_{low,high}$)

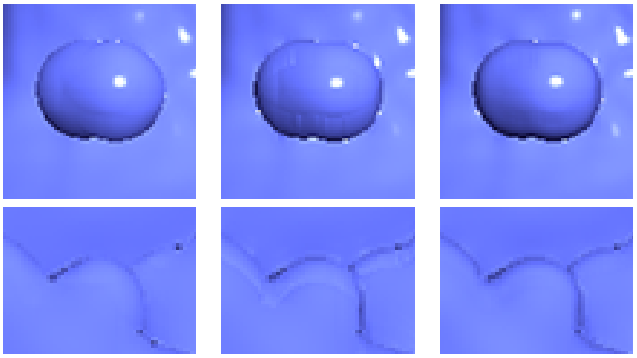
Figure 5: *An example frame showing the impact of using (a) static range and (b) dynamic range. Notice that the fluid surface with static range contains many discontinuities, while dynamic range leads to smooth fluid surface viewed at grazing angles.*

in filtering by effectively eliminating parts of the filter kernel. Consider a background pixel near the discontinuity. Filtering this pixel would only consider the depth values that are not occluded, which are on one side of the filter kernel. Thus, the values used in filtering come from the other side of the filter kernel alone, leading to bias in filtering. If the background surface is viewed at an angle, this bias would cause the resulting surface to bend, following the surface gradient.

We avoid this problem using *bias correction*. Our bias correction ensures that the filter kernel is always symmetrical. We achieve this by considering two opposing points on the filter kernel together. Let pixel j be on one side of the pixel i that is being filtered. We consider the pixel k on the opposing side of the filter kernel, such that $\mathbf{p}_k = \mathbf{p}_i + (\mathbf{p}_i - \mathbf{p}_j)$. Note that if pixel indices use scanline order, we can write $k = i + (i - j)$. By construction, $G(\mathbf{p}_i, \mathbf{p}_j, \sigma_i) = G(\mathbf{p}_i, \mathbf{p}_k, \sigma_i)$; however, the weights computed using Equation 3 can vary depending on whether z_j and z_k are within the depth range. Our bias correction simply considers these two pixels j and k together. If either one of them are out of range for filtering pixel i, such that $z_j > z_i + \delta$ or $z_k > z_i + \delta$, *both* of them are ignored during filtering. We can achieve this by simply rewriting Equation 3 as

$$\omega_{ij} = \begin{cases} 0, & \text{if } z_j > z_i + \delta \text{ or } z_k > z_i + \delta \\ G(\mathbf{p}_i, \mathbf{p}_j, \sigma_i), & \text{otherwise.} \end{cases} \tag{6}$$

The result of bias correction with this formulation is shown in Figure 4c. Notice that bias correction eliminates the distortion of the background surface near discontinuities with inclined surfaces. Note that this formulation does not assume that the background surface is flat, but it merely ensures that the filter kernel is symmetrical. Nonetheless, the visual artifacts of the biased filter (without bias correction) become more obvious when the background surface is nearly flat. When the background surface has sharp details, the bias artifacts are more difficult to notice due to the complexity of the background surface. However, when the background is nearly flat (after filtering), the distortion around foreground surfaces because of filter bias can be obvious and such artifacts can be fixed using our bias correction.



(a) 2D Filter Only (b) 1D Filters Only (c) 1D Filters + Clean-up

Figure 6: *Close-up views from an example fluid surface rendered using our narrow-range filter: (a) using the 2D filter without separable filter approximation, (b) separable filter approximation using 1D filters, and (c) separable filter approximation followed by the final clean-up pass with a small 2D filter. Notice that 1D filters result in vertical streaks that are visible near discontinuities and using a clean-up pass removes those streaks, producing an image very similar to the 2D filter result.*

3.3 Dynamic Range Adjustment

One difficulty with our narrow-range filter formulation is setting the right δ parameter for the desired result. In particular, with small δ values, when an approximately flat fluid surface is viewed from grazing angles, the steep change in surface depth can be mistaken as depth discontinuity. This happens when the nearby depth values z_j within the filter kernel range fall out of the depth range defined by $z_i + \delta \geq z_j \geq z_i - \delta$. Larger δ values alleviate this issue, but they can cause filtering across actual discontinuities.

We provide a simple solution for this by adjusting the depth range dynamically using

$$z_i + \delta_{high} \geq z_j \geq z_i - \delta_{low} \,, \tag{7}$$

where δ_{low} and δ_{high} are *dynamic* threshold values that are initially set as $\delta_{low} = \delta_{high} = \delta$. While computing the filtered depth value for pixel i, starting with the closest neighboring pixels, we dynamically adjust δ_{low} and δ_{high}, such that if the neighboring pixel depth z_j is within acceptable range $z_i - \delta_{low} \leq z_j \leq z_i + \delta_{high}$, we adjust the thresholds using

$$\delta_{low} \leftarrow \max(\, \delta_{low},\ z_i - z_j + \delta \,) \,, \tag{8}$$

$$\delta_{high} \leftarrow \max(\, \delta_{high},\ z_j - z_i + \delta \,) \,. \tag{9}$$

This effectively expands the acceptable depth range along the current surface. Thus, our δ parameter effectively controls the acceptable depth difference between neighboring pixels. By replacing δ with δ_{low} and δ_{high}, we can produce smooth surfaces for flat regions at grazing angles. Figure 5 shows an example with an approximately flat surface rendered with and without dynamic range adjustment, showing that using dynamic ranges with δ_{low} and δ_{high} can significantly improve the surface quality.

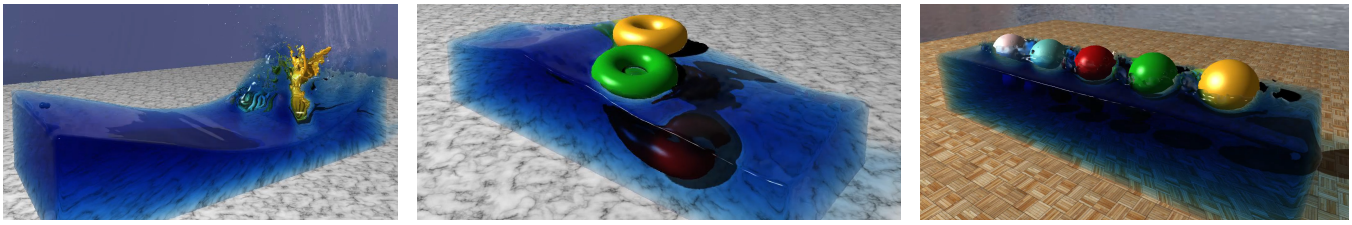
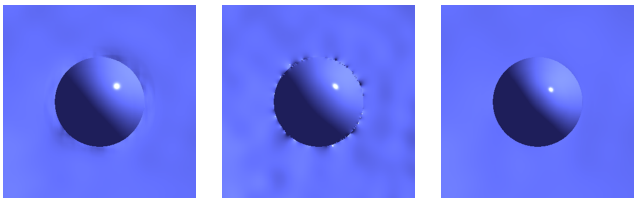


Figure 7: *Example frames from different particle-based fluid simulations rendered using our narrow-range filter.*



Plane Fitting | Screen-Space Curvature Flow | **Narrow-Range Filter**

Figure 8: *The same example in Figure 3 with a drop occluding an approximately flat fluid surface in the background, comparing plane fitting, SSCF, and our narrow-range filters. Plane fitting leads to some minor distortion on the background surface near discontinuities. SSCF requires a large number of passes to smooth the background surface and numerical instabilities lead to minor distortion near discontinuities. Our narrow-range filter produces an artifact-free image in this example, using the same particle data.*

3.4 Separable Filter Approximation

Unfortunately, our narrow-range filter formulation is not separable. Yet, it is still possible to use a separable filter approximation by applying our filter as two 1D filter passes with alternating directions. This leads to substantial performance improvement, especially when rendering extreme close-ups that use large filter sizes, but it also produces visible artifacts in the form of axis-aligned streaks. An example of these artifacts are shown in Figure 6b.

We noticed that these artifacts mostly disappear with an additional filter pass using our 2D filter after multiple passes with the separable filter approximation. Therefore, as a remedy for these artifacts, we introduce an additional *clean-up* pass, after all 1D filter passes are completed. This final clean-up pass uses our 2D filter with a small fixed filter size. Its purpose is to hide the streaks produced by the last 1D filter pass, rather than filtering the depth values. Using a small filter size (5×5 in our implementation) provides a minor performance overhead, but can hide majority of the artifacts. This way, we can benefit from the performance improvement of the separable filter approximation without introducing objectionable artifacts.

Figure 6c shows an example of the impact of our final clean-up pass. Notice that the final result is similar to using the 2D filter without separable filter approximation (Figure 6a).

4 RESULTS

We have tested our fluid rendering method with our narrow-range filter using various particle-based fluid simulations. Example frames from some of these simulations are shown in Figures 1 and 7. We also compare the results of our method to bilateral Gaussian filter, plane fitting [Imai et al. 2016], and SSCF [van der Laan et al. 2009]. In all examples we use the same number of iterations for bilateral Gaussian filter, plane fitting, and our narrow-range filter, but more iterations are used for SSCF. The filter size for bilateral Gaussian and plane fitting are dynamically computed similar to ours, using Equations 5 with $\sigma = 0.7r$, where r is the radius of a particle. For our narrow-range filter we use $\delta = 10r$ and $\mu = r$ for all examples in this paper unless otherwise specified.

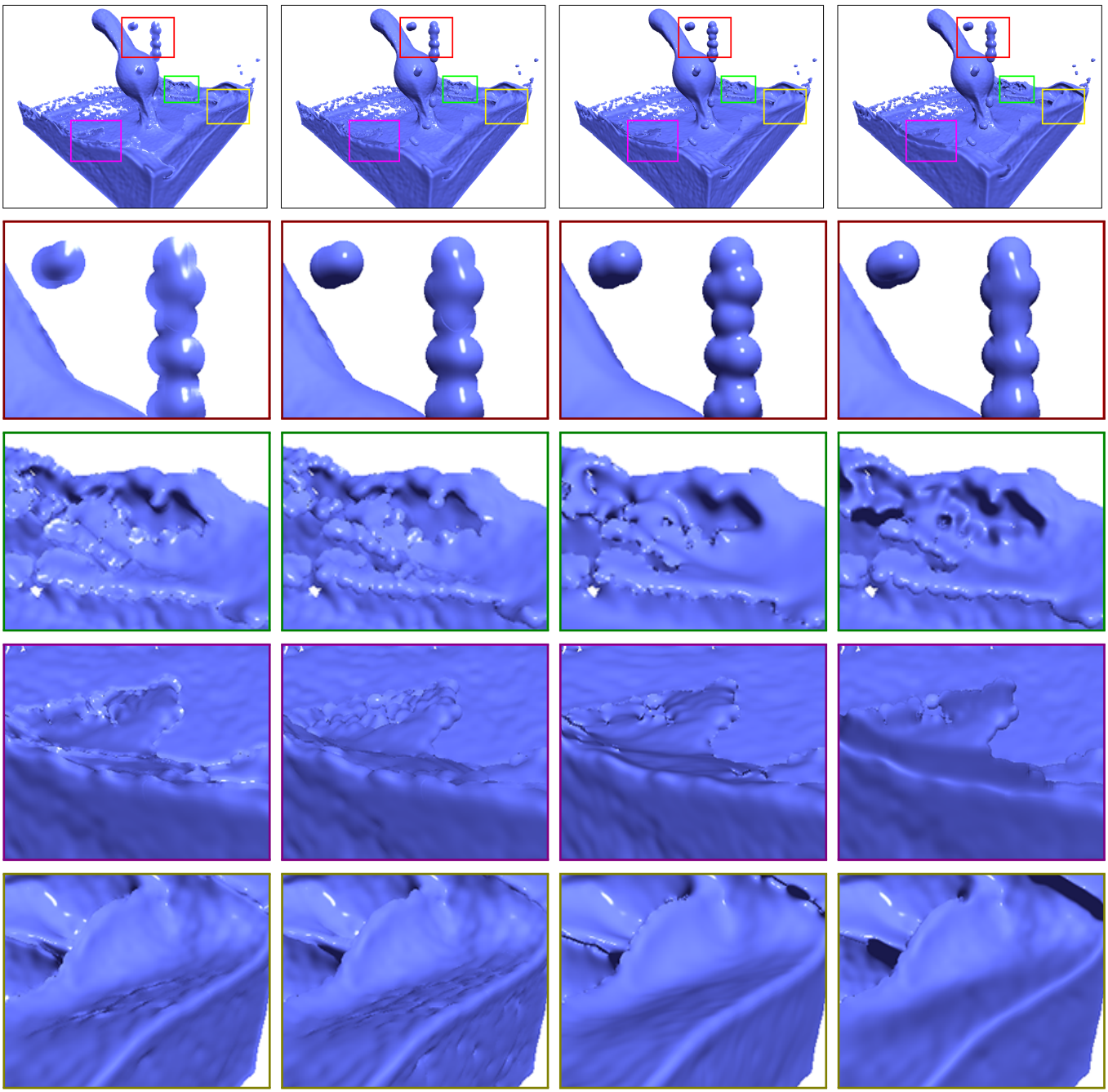
Figure 8 shows the same example in Figure 3, where an approximately flat fluid surface is occluded by a drop in the foreground. Bilateral Gaussian filter (Figure 3) not only distorts the background surface but also flattens the drop in the foreground. Both plane fitting and SSCF produce a desirable curved boundary for the drop in the foreground, but they lead to some minor distortion on the background surface near the discontinuities. Using the same particle data, our narrow-range filter produces a smooth surface for the background without any visible distortion near discontinuities.

A more comprehensive comparison is provided in Figure 9, showing a challenging frame that includes various surface features. Some of these surface features are highlighted, including isolated drops, complex details, thin surfaces with sharp discontinuities, and smooth areas. All methods use three iterations, except for SSCF, which requires a large number of iterations to produce smooth surfaces.

As apparent in previous comparisons as well, bilateral Gaussian filter flattens isolated drops (Figure 9a). A similar form of flattening is also apparent near all other surface details involving discontinuities. Furthermore, the fluctuations in the input depth values lead to extensive noise on approximately flat parts of the surface.

Plane fitting (Figure 9b) provides some improvements over bilateral Gaussian filter. The surfaces of the isolated particles are not completely flattened with plane fitting and sharp discontinuities are properly handled. However, it fails to produce a smooth surface for complex features, thin surface details, and approximately flat regions. The smooth parts of the fluid surface are less noisy than the bilateral Gaussian filter results, but they still contain considerable amount of noise.

SSCF (Figure 9c) produces high-quality surface details in most areas. However, even after 100 iterations, parts of the surface still lack a desirable level of smoothness. Isolated particles are not flattened, but particles that are relatively closer to the camera require more iterations to form smooth surface details. On the other hand, parts of the fluid surface that are further away from the camera have excessive smoothness. This could be resolved by limiting the iteration count for those regions [Bagar et al. 2010]. However, SSCF



(a) Bilateral Gaussian (3 iterations)

(b) Plane Fitting (3 iterations)

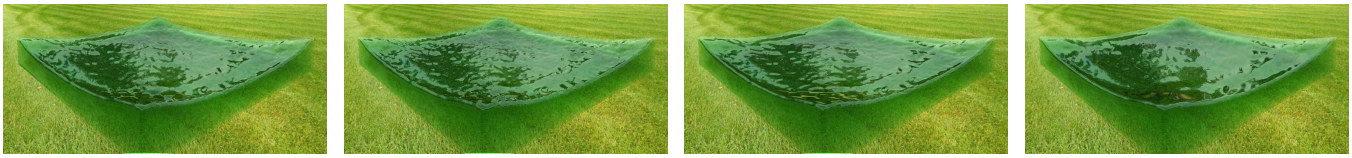
(c) Screen-Space Curvature Flow (100 iterations)

(d) **Our Narrow-Range Filter** (1D + clean-up, 2+1 iterations)

Figure 9: *Comparison of different filtering methods: (a) bilateral Gaussian filter, (b) plane fitting, (c) screen-space curvature flow, and (d) our narrow-range filter using separable filter approximation with an additional clean-up pass.*

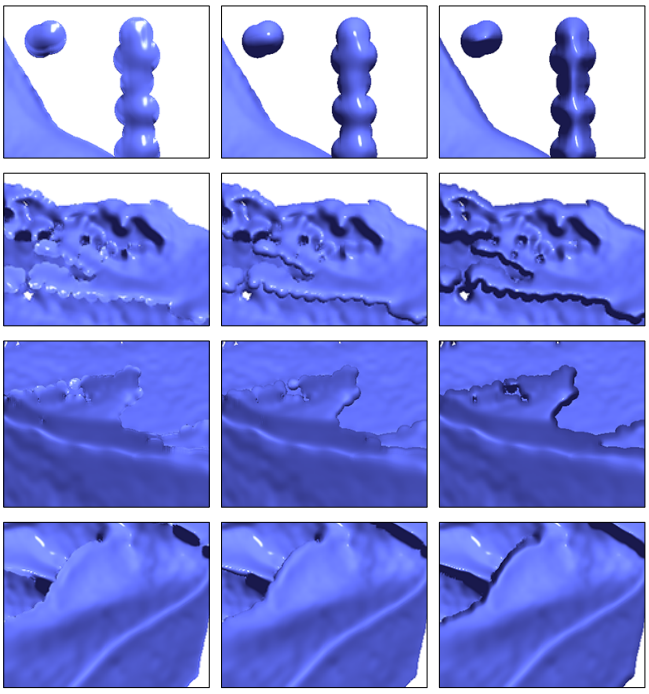
encounters some numerical instability near sharp discontinuities, producing a few pixel-wide high-frequency noise around them.

In comparison, our narrow-range filter (Figure 9d) can produce smooth surfaces using only two iterations (a total of four 1D filter passes) and a final clean-up pass. Isolated particles join together to form smooth surfaces with a desirable curvature near discontinuities. The surface has uniform smoothness, regardless of the camera distance. All discontinuities are preserved with curved boundaries for the foreground surfaces and smooth surfaces without distortion for the partially-occluded background surfaces.



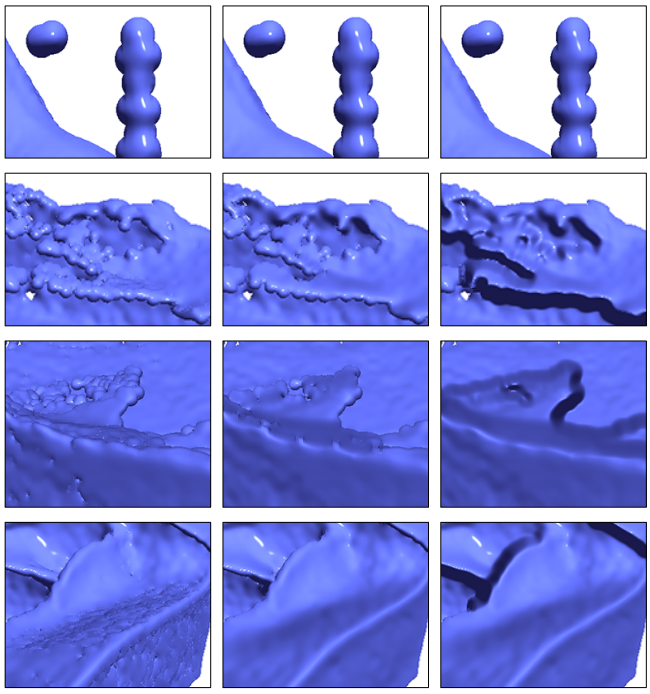
(a) Bilateral Gaussian (3 iterations) (b) Plane Fitting (3 iterations) (c) Screen-Space Curvature Flow (100 iterations) (d) **Our Narrow-Range Filter** (1D + Clean-up, 2+1 iterations)

Figure 10: *Comparison of reflections and refractions on surfaces generated using different filtering methods: (a) bilateral Gaussian filter, (b) plane fitting, (c) screen-space curvature flow, and (d) our narrow-range filter using separable filter approximation with an additional clean-up pass.*



(a) $\mu = 0$, $\delta = 10r$ (b) $\mu = r$, $\delta = 10r$ (c) $\mu = 10r$, $\delta = 10r$

Figure 11: *Surfaces generated with different μ parameter values using our narrow-range filter (1D + clean-up, 2+1 iterations).*



(a) $\mu = r$, $\delta = r$ (b) $\mu = r$, $\delta = 5r$ (c) $\mu = r$, $\delta = 50r$

Figure 12: *Surfaces generated with different δ parameter values using our narrow-range filter (1D + clean-up, 2+1 iterations).*

The smoothness of the surface is especially important when rendering the surface with reflections and refractions, as shown in Figure 10. Notice that our narrow-range filter can produce a smooth surface with smooth reflections and refractions, while other methods, including curvature flow with 100 iterations, contain considerable amount of noise.

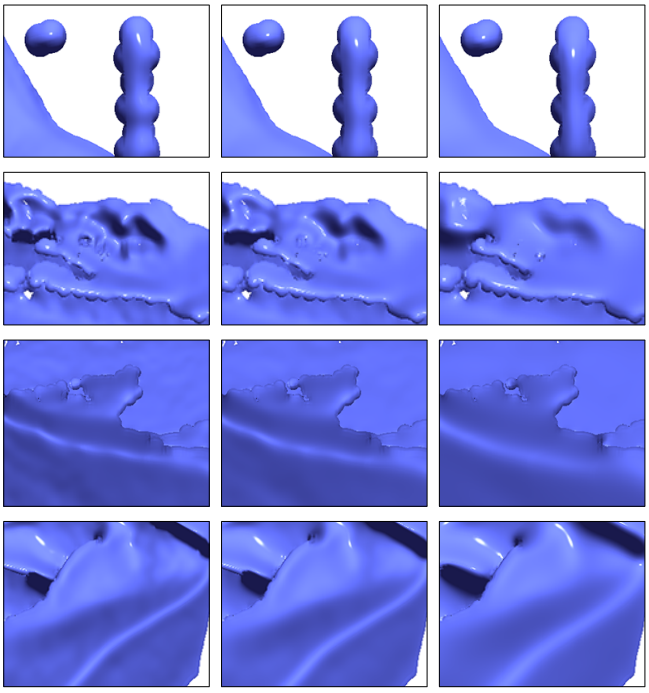
The μ parameter of our narrow-range filter controls the curvature of the surface near discontinuities. Figure 11 shows results with different μ values. Using extremely small μ values, such as $\mu = 0$, make the edge of the fluid appear flat (Figure 11a), similar to bilateral Gaussian filter results. Larger μ values form a curved border (Figure 11b). This effect is exaggerated when the μ value is too large (Figure 11c).

The δ parameter of our narrow-range filter controls the depth range that is considered when filtering. Results with different δ values are shown in Figure 12. When δ is too small, filtering is limited to a very narrow depth range (Figure 12a). Consequently, surfaces viewed at an angle, which have relatively large variations in the depth values of neighboring pixels, may not be filtered properly. When neighboring depth values fall out of the depth range, they are treated as discontinuities, resulting in cracks in surface appearance. Simply using larger δ values resolves this issue (Figure 12b). However, if the δ value is too large, actual discontinuities may be missed and separate parts of the surface may get filtered together (Figure 12c). In other words, the δ parameter determines the magnitude of depth variance that is treated as a surface discontinuity and the range within which depth values are filtered together.

Like other screen-space filtering methods, the number of iterations control the effective filter size. Figure 13 show results with different numbers of iterations. Notice that using more iterations make the surfaces smoother by filtering out small-scale details, but the same surface discontinuities remain.

Table 1: *Performance Results*

Number of Particles	Figure 1 Armadillo 368K	Figure 1 Bunny 333K	Figure 1 Emitter 350K	Figure 1 Drop 65K	Figure 7 Lucy 135K	Figure 7 Tori 135K	Figure 7 Balls 180K
Bilateral Gaussian Filter	0.40 ms	0.46 ms	1.03 ms	0.68 ms	0.48 ms	0.38 ms	0.62 ms
Plane Fitting	0.70 ms	0.90 ms	1.95 ms	1.51 ms	0.93 ms	0.72 ms	1.32 ms
Screen-Space Curvature Flow	5.02 ms	5.50 ms	6.05 ms	5.30 ms	4.34 ms	4.82 ms	5.80 ms
Our Narrow-Range Filter 2D	0.53 ms	0.68 ms	1.58 ms	1.23 ms	0.72 ms	0.53 ms	1.04 ms
Our Narrow-Range Filter 1D	0.51 ms	0.58 ms	1.25 ms	0.62 ms	0.62 ms	0.50 ms	0.67 ms
Our Narrow-Range Filter 1D & Clean-up	0.46 ms	0.52 ms	1.21 ms	0.54 ms	0.56 ms	0.46 ms	0.58 ms



(a) 2 + 1 iterations (b) 4 + 1 iterations (c) 14 + 1 iterations

Figure 13: *Surfaces generated with different number of filter iterations using our narrow-range filter (1D + clean-up).*

Table 1 provides the performance results of the filtering operations for our test scenes. All timings are computed on an NVIDIA GTX 1080 GPU. Notice that our narrow-range filtering methods with and without separable filter approximation provide a similar performance as the bilateral Gaussian filter results. In these examples separable filter approximation with our method leads to only a minor performance improvement, though the performance difference can be more substantial if the rendered scene includes extreme close-ups that would lead to large screen-space filter sizes. In most cases, our 1D filter with clean-up is faster than the 1D and 2D variants, because the last clean-up pass performs filtering with very small fixed filter size (5×5). Plane fitting is considerably slower than our method and SSCF takes about an order of magnitude longer to compute because of the large number of iterations needed.

5 CONCLUSION

We have introduced a narrow-range filter for rendering particle-based fluid simulations. Our filter formulation is simple, easy to implement, and fast to compute. Our test results show that this new filter provides improved surface quality, as compared to prior screen-space fluid rendering methods.

REFERENCES

Florian Bagar, Daniel Scherzer, and Michael Wimmer. 2010. A Layered Particle-Based Fluid Model for Real-Time Rendering of Water. *Computer Graphics Forum* 29, 4 (2010), 1383–1389.

Hilko Cords and Oliver G. Staadt. 2009. Interactive Screen-Space Surface Rendering of Dynamic Particle Clouds. *Journal of Graphics, GPU, and Game Tools* 14, 3 (2009), 1–19.

Mathieu Desbrun and Marie-Paule Gascuel. 1996. Smoothed Particles: A New Paradigm for Animating Highly Deformable Bodies. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96*. Springer-Verlag New York, Inc., New York, NY, USA, 61–76.

Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola. 2010. Interactive SPH Simulation and Rendering on the GPU. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '10)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 55–64.

Simon Green. 2010. Screen Space Fluid Rendering for Games. In *Game Developers Conference*.

Takuya Imai, Yoshihiro Kanamori, and Jun Mitani. 2016. Real-time screen-space liquid rendering with complex refractions. *Computer Animation and Virtual Worlds* 27, 3-4 (2016), 425–434. cav.1707.

Miles Macklin and Matthias Müller. 2013. Position Based Fluids. *ACM Trans. Graph.* 32, 4, Article 104 (July 2013), 12 pages.

Matthias Müller, Simon Schirm, and Stephan Duthaler. 2007. Screen Space Meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '07)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 9–15.

F. Reichl, M. G. Chajdas, J. Schneider, and R. Westermann. 2014. Interactive Rendering of Giga-particle Fluid Simulations. In *Proceedings of High Performance Graphics (HPG '14)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 105–116.

Ilya D. Rosenberg and Ken Birdwell. 2008. Real-time Particle Isosurface Extraction. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games (I3D '08)*. ACM, New York, NY, USA, 35–43.

Wladimir J. van der Laan, Simon Green, and Miguel Sainz. 2009. Screen Space Fluid Rendering with Curvature Flow. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (I3D '09)*. ACM, New York, NY, USA, 91–98.

Marcel Weiler, Dan Koschier, and Jan Bender. 2016. Projective Fluids. In *Proceedings of the 9th International Conference on Motion in Games (MIG '16)*. ACM, New York, NY, USA, 79–84.

Xiangyun Xiao, Shuai Zhang, and Xubo Yang. 2017. Real-time High-quality Surface Rendering for Large Scale Particle-based Fluids. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '17)*. ACM, New York, NY, USA, Article 12, 8 pages.

Jihun Yu and Greg Turk. 2013. Reconstructing Surfaces of Particle-based Fluids Using Anisotropic Kernels. *ACM Trans. Graph.* 32, 1, Article 5 (Feb. 2013), 12 pages.

Yanci Zhang, Barbara Solenthaler, and Renato Pajarola. 2008. Adaptive Sampling and Rendering of Fluids on the GPU. In *Proceedings of the Fifth Eurographics / IEEE VGTC Conference on Point-Based Graphics (SPBG'08)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 137–146.