Cartoon Water Rendering with Foam and Surface Smoothing

Liordino dos S. Rocha Neto
Antonio L. Apolinário Jr.
Federal University of Bahia

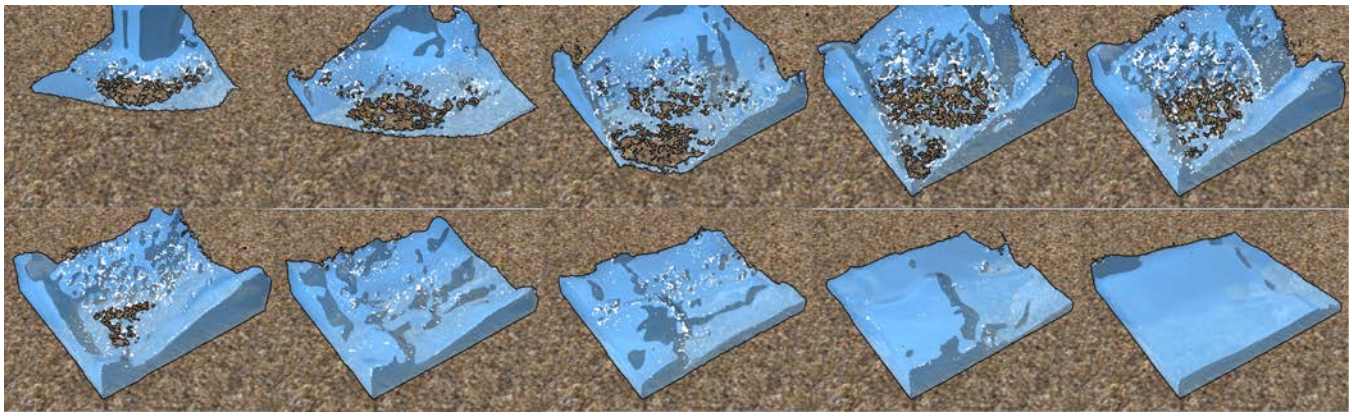


Figure 1: *Sequence of images showing an example of the results achieved with our method for a falling water scene simulated through smoothed particle hidrodynamics. Notice the presence of silhouette edges and foam, as well as the absence of striped artifacts common to the separated bilateral filter smoothing results.*

Abstract

We present an extension to a previously developed method for rendering smooth surfaces of particle-based liquid simulations that is suitable for use in real-time interactive environments such as video games. Our extension uses a version of the bilateral filter to smooth the fluid's surface and introduce a new method for generating foam that is appropriate for non-photorealistic styles. The method occurs in screen space, which avoids the usual artifacts of polygonization techniques and all the steps are implemented directly in the graphics hardware. Performance and visual analysis are realized to demonstrate the applicability of the approach.

Keywords: Non-photorealistic Rendering, Screen Space Rendering, Smoothed Particle Hydrodynamics Simulation

Author's Contact:

liordino.neto@ufba.br.
apolinario@dcc.ufba.br

1 Introduction

Rendering physically simulated fluids has been a topic of much research nowadays, thanks to the great demand for realistic simulations in computer graphics [Ojeda Contreras et al. 2013]. Although this is true for environments like movies, it represents a greater challenge in interactive real-time applications such as games, where the dynamic behavior of fluids can be a desirable gameplay element [Kellomäki 2012]. Three basic steps must be performed to obtain a graphical representation from physical simulations like these: simulating the fluid, extracting a renderable representation of it and finally performing the rendering itself.

There are two main broad categories that fluid simulations can be divided into: eulerian (grid-based) and Lagrangian (particle-based) [Williams 2008]. While grid-based approaches have the advantage of a high-quality surface straight-forward extraction, they are usually more costly than particle-based ones in both memory and computation, what makes the latter preferred in real-time interactive environments [van der Laan et al. 2009].

Several approaches have been proposed recently to extract and render smooth surfaces from particle-based fluids, but the majority of these methods aim only at photorealistic renditions [van der Laan et al. 2009] [Fraedrich et al. 2010] [Bagar et al. 2010]. The surface extraction and smoothing can be done either through object-space polygonization methods like Marching Cubes or screen-space approaches. Due to the computation and memory intensive behavior of the former, screen-space methods are prefered in real-time environments like games [van der Laan et al. 2009]. Methods to render non-photorealistic water were recently developed, but they are either not meant for real-time applications [Eden et al. 2007] [You et al. 2009], don't use fluid simulation [Yu et al. 2007] or don't take into account effects like foam and spray [Neto et al. 2013]. Although techniques to generate foam and spray specifically for cartoon water were developed recently, they rely on pre-loaded textures and procedural methods instead of on the simulation itself [Yingst et al. 2011] [Liao et al. 2011].

Our method's rendering style is inspired by modern cartoon animations like *One Piece*[1] (Figure 2(a)) and *Avatar: The Last Airbender*[2] (Figure 2(b)), and further enhances the visual style observed in these examples with optical effects and silhouette edges. This paper presents an approach to render a cartoon-style SPH simulation that extends the work of Neto *et al.* [Neto et al. 2013] with the following contributions:

- A surface smoothing algorithm that makes artifacts generated by the separated version of the bilateral filter virtually unnoticeable while having a low performance impact;
- A neighborhood-based technique to hide stray particles and generate foam and spray suitable for non photorealistic rendering;

This paper is structured as follows: section 2 presents related works, highlighting their contribution to our method, which section 3 describes. Section 4 explains the conducted experiments and discuss their results. Finally, section 5 presents our conclusions, along with suggestions for future works.

2 Related Work

SPH as means to simulate water in computer animation was first introduced by Müller *et al.* [Müller et al. 2003], where it was successfully used in interactive real-time applications. Improvements over the initial approach were developed, first being fully implemented

[1] ©Toei Animation, http://www.toei-anim.co.jp/tv/onep/
[2] ©Nickelodeon Animation Studios, http://www.nick.co.uk/shows/avatar/

(a) (b)

Figure 2: *Two examples of modern hand drawn cartoons that inspired our work: (a) One Piece, from Toei Animation and (b) Avatar: The Last Airbender, from Nickelodeon Animation Studios.*

in CPU [Adams et al. 2007] [Solenthaler and Pajarola 2009], and later in GPU [Hoetzlein 2012] [Macklin and Müller 2013]. Our focus on this our work is on rendering SPH-based water simulations with a traditional cartoon style, and it is assumed that an SPH simulation is already carried out. The following subsections lists works related to ours in different categories.

2.1 Surface Extraction and Smoothing

Müller's work, while introducing the use SPH for real-time fluid simulations, also successfully performed it's surface extraction with both marching cubes [Lorensen and Cline 1987] and surface splatting (also called point splatting) [Zwicker et al. 2001] methods. While marching cubes extracts a simulation iso-surface, the point splatting objective is to render surfaces from point clouds without any connectivity. In a recent work, Van der Laan *et al.* [van der Laan et al. 2009] proposes a point splatting based screen space technique that renders particles as screen oriented point sprites through several steps performed in the graphics hardware, reducing geometric computations. The same framework was used by Neto *et al.* [Neto et al. 2013] to render cartoon water. Our work uses the same workflow, extending it with a better surface smoothing algorithm, foam rendering and stray particles removal.

Many fluid rendering methods employee smoothing algorithms to prevent a blobby appearance when rendering surfaces extracted from particle-based simulations. An iterative curvature flow [Desbrun et al. 1999] is used in Van der Laan's work [van der Laan et al. 2009]. This technique consists of repeatedly shifting a surface along it's normal vector depending on it's mean curvature. An adaptive version of this filter was used in the work by Bagar *et al.* [Bagar et al. 2010], varying the number of iterations in order to produce a consistent fluid surface independent of the view distance. Green's work [Green 2010] shows the same workflow, but instead of curvature flow it uses a separable bilateral filter [Pham and Vliet 2005] to smooth the fluid surface. It greatly improves performance over the full kernel bilateral filter [Tomasi and Manduchi 1998], but generates striped artifacts over the smoothed surface. Posterior shading steps hides these artifacts in photorealistic renderings [Green 2010], but that's not enough if an NPR style is chosen [Neto et al. 2013]. Gastal and Oliveira [Gastal and Oliveira 2011] shows an approach to make these artifacts virtually unnoticeable by iteratively applying a separated filter reducing it's kernel at each iteration. This idea was adapted to the bilateral filter, successfully removing striped artifacts while having a low performance impact.

2.2 NPR Rendering

Both Selle *et al.* [Selle et al. 2004] and McGuire and Fein [McGuire and Fein 2006] presents methods to create cartoon style renderings with information gathered from particle-based fluid simulations, being the later in real-time. Unfortunately these approaches are used to render smoke, what makes them not ideal for our purposes, since smoke don't have a free surface that makes it's interface with air, like liquids [Eden et al. 2007]. Winemoller introduced the XDoG operator [Winnemöller 2011], an extended difference of gaussians filter that produce smooth edges and stylized images. Other approaches for image stylization are the anisotropic Kuwahara [Kyprianidis et al. 2009] and coherence-enhancing filters [Kyprianidis and Kang 2011]. Images can also be stylized through bilateral filter's iterative application [Kyprianidis et al. 2013]. Although these filters can successfully create NPR style renderings, they show visual results that differ from our traditional cartoon approach.

Contributions by Eden *et al.* [Eden et al. 2007] and Yu *et al.* [Yu et al. 2007] propose cartoon water rendering techniques, but they ignore water's optical characteristics. You *et al.* [You et al. 2009], on the other hand, takes into account optical properties like transparency, reflection and refraction, but don't achieve real-time performance. A surface generated by a physically based fluid simulation is used as input, and several shading steps are combined to compose it's result, including an automatic control of reflection and refraction also used in Neto's work [Neto et al. 2013].

2.3 Spray and Foam Generation

Foam rendering has been the object of study of some recent works. Yingst *et al.* method [Yingst et al. 2011] emulate sea foam dissipation with a precomputed dither array, on a method suitable for real-time environments, but produces pixelation as the camera approaches the liquid surface. Liao *et al.* [Liao et al. 2011] presents another method, consisting of procedural modeling of water caustics and foam employing a texture generated with Voronoi diagrams. Both methods don't require a physical simulation. Zhang *et al.* work [ZHANG et al. 2014], on the contrary, uses the Weber number to simulate foam and sprays generated by particle-based fluid motion. Each particle Weber number is calculated and compared to critical thresholds that divides them into water, transition and foam. Each group are them rendered in a different way to compose the method's result, also achieving real-time performance. Bagar's work [Bagar et al. 2010] also rely on Weber number to generate particle-based foam. In our method, particles are divided into groups by comparing their neighbor count. This way we avoid doing any extra calculations, since the neighbor counting is a process

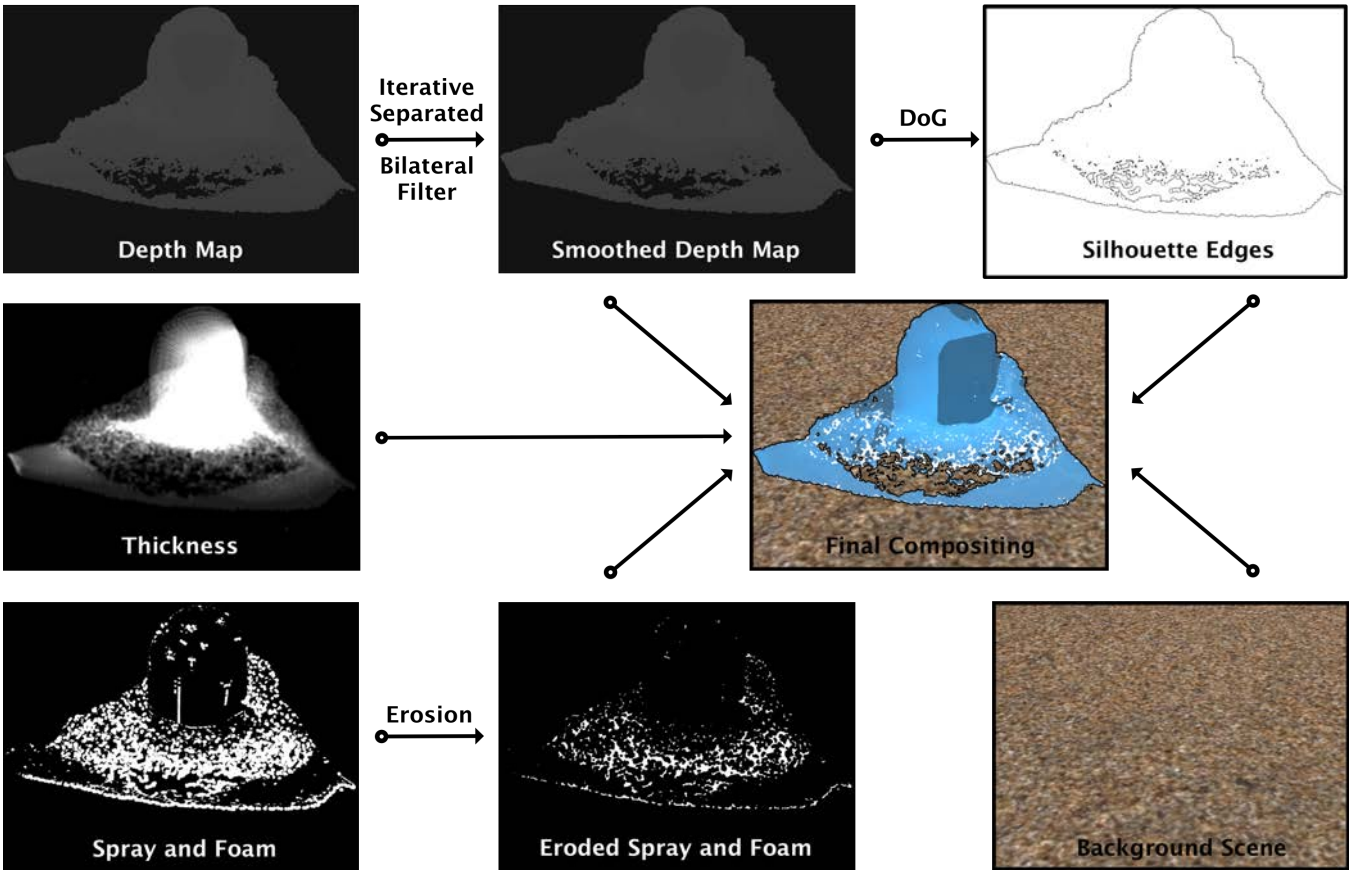


Figure 3: *Overview of the intermediate steps of our method. From the particles position a depth map is obtained and smoothed out. Next, silhouette edges are retrieved from the smoothed depth map. Then water thickness is obtained, along with the foam/spray mapping, which is morphologically eroded to prevent a blobby aspect. Finally, the intermediate steps are blended with the background scene texture to generate the final compositing.*

already done in SPH simulations.

3 Technique Overview

Our technique is built on top of the cartoon water rendering presented in a previous work [Neto et al. 2013]. We assume that an SPH simulation is already carried out, and use it's particles positions as input. Figure 3 shows a step by step overview of the technique. The process start by obtaining the fluids frontmost surface depth, thickness and grouping with a point splatting technique and storing each one in an offscreen buffer. Next, a smoothing step is performed on the depth map in order to prevent a blobby aspect on the fluid's surface. Two versions of the bilateral filter were used for smoothing the depth map of in a previous work [Neto et al. 2013]: a simple bilateral filter and it's separated version. The problem with these two ways to apply the bilateral filter were performance and the generation of striped artifacts, respectively. In this work an iterative version of the separated bilateral filter is used in the smoothing step to remove the artifacts created by it's separated version while having a low performance impact (section 3.1). Stray particles are removed while creating the particle grouping texture, that is morphologically eroded in order to remove unwanted details (section 3.2). These steps are then incorporated into the final composite step (section 3.3). Simple silhouette edges are also added using the difference of Gaussians (DoG) operator [Marr and Hildreth 1980], to further enhance the cartoon style.

3.1 Surface Smoothing

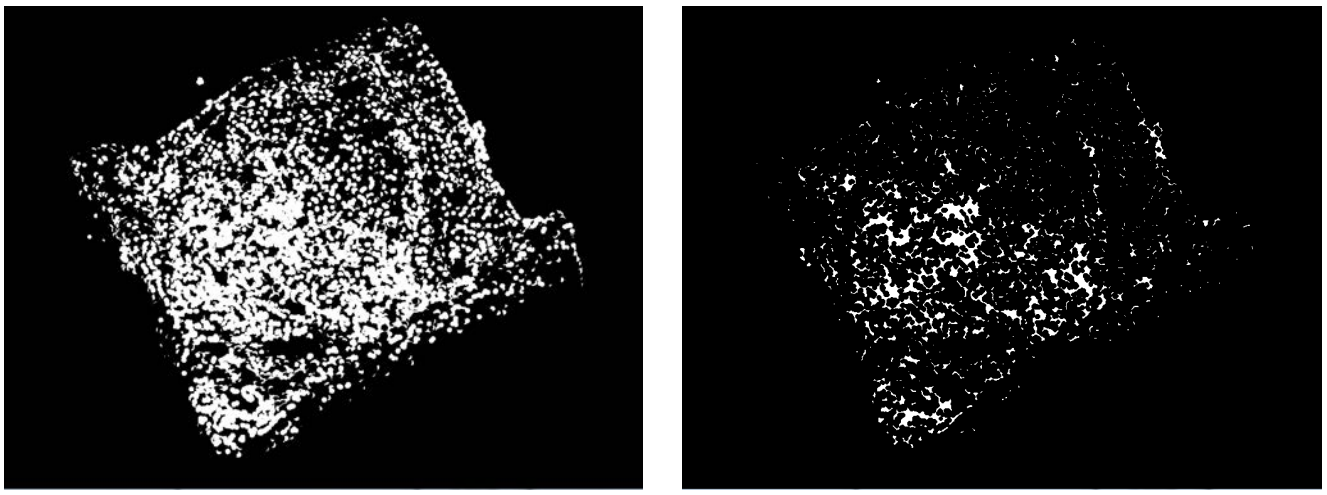
After obtaining a fluid surface from particle positions, a smoothing process is needed to prevent a blobby-like appearance in the final rendering. A simple Gaussian blur is not suitable for this task, since it would blur the fluid's silhouette edges, blending the particles with background surfaces [Green 2010]. Usually a bilateral filter is used to overcome this limitation [Tomasi and Manduchi 1998], employing a regular Gaussian filter with a spatial kernel f and a function g on the intensity domain to determine a pixel's weight. This way two Gaussian filters are combined, one in the spatial domain and another in intensity domain, what makes the value of a certain output pixel s more influenced by pixels close to it in both domains [Durand and Dorsey 2002]. Its output is:

$$J_s = \frac{1}{k} \sum_{p \in \Omega} f(p - s) g(Ip - Is) Ip \quad (1)$$

where p is a pixel on the image Ω, Ip and Is are the values of pixels p and s on the intensity domain, and k is a normalization term:

$$k_s = f(p - s) g(Ip - Is) \quad (2)$$

Bilateral filter iterates through both width and height of its spatial kernel at the same time, making it expensive as the kernel size grows. For this reason, a faster approximation version of this filter that still satisfies the noise reduction and edge preservation requirements can be used for the sake of performance. This version first applies a one-dimensional filter to one dimension of the image, and then filters this intermediate result in the subsequent dimension [Pham and Vliet 2005]. This way the computational complexity of the separated Bilateral Filter becomes equals $O(p)$, faster than the $O(p^2)$ full kernel approach, where p is the number of pixels in the image [Paris et al. 2007]. Although this complexity reduction is obtained at the expense of the generation of some artifacts, this strategy was applied successfully to the photo-realistic water rendering method presented in [Green 2010], since this artifacts was masked by it's shading steps. In a cartoon style rendition, though, the shading steps aren't enough to hide the artifacts [Neto et al. 2013].



(a) **(b)**

Figure 4: *Results of the morphological erosion process over a particle grouping map: (a) Before the erosion, (b) after the erosion.*

Gastal and Oliveira [Gastal and Oliveira 2011] shows that these striped artifacts come from the fact that the filtering of a 2D signal using a 1D transform is not a separable operation. This occurs because with a single iteration of a 1D filtering (a horizontal pass followed by a vertical pass, or vice-versa) some pixels that belong to the same region, affected by the filter's kernel, may end not being combined. To solve this problem, a simpler version of their technique was implemented, relying on two key observations: every 1D step removes artifacts introduced by the previous, what makes the stripes present only along the last filtered dimension, and that their length is proportional to the dimensions of the kernel used on the last pass. This way a sequence of vertical and horizontal passes are interleaved, reducing it's kernel dimensions in half at every new iteration and progressively reducing the extension of the striped artifacts, making them virtually unnoticeable. According to the authors three iterations are usually enough to achieve good results, what we confirmed in our experiments (section 4).

3.2 Particle Grouping

Artifacts generated by the presence of stray particles and the absence of spray and foam are the main limitations pointed by [Neto et al. 2013]. Both of these limitations are addressed with the separation of the particle set in three groups: water, spray or foam and stray particles. Assuming that an SPH simulation has been already carried out and that it has a neighbor counting (N_{count}) process for each particle, this value is compared with two different thresholds T_{stray} and T_{foam} to set each particle group P_{group}:

$$P_{group} = \begin{cases} Stray & \text{if } N_{count} <= T_{stray} \\ Spray/Foam & \text{if } T_{stray} < N_{count} <= T_{foam} \\ Water & \text{Otherwise} \end{cases} \tag{3}$$

After setting the group of an particle it is drawn: stray particles are ignored, spray or foam particles are painted white, and finally, the group of particles are painted black. This way a binary texture that tells how to render each particle on the final rendering is generated and stored in an offscreen buffer.

Since the particles are first rendered as spherical point sprites, the obtained foam and spray map has a blobby appearance. To smooth it, we could use the same iterative separated bilateral filter, but it would have no effect on a binary image. A morphological erosion [Serra 1982] was an adequate approach, as it can be used to remove unwanted details on binary images. This operation performs a convolution where pixels are removed from object boundaries, turning white pixels (i.e. the object) into black. This is done by searching for the minimum value of the neighborhood (defined by the kernel) and setting it to the current pixel. Figure 4 shows the result of a erosion applied over the particle grouping map.

3.3 Final Compositing

With all the intermediary textures available, they are finally combined to render the cartoon water. To determine how to render each particle the grouping texture (generated in section 3.2) is used, and the final illumination of the Cartoon Water Shader I_{wcs} is obtained as follows:

$$I_{wcs} = \begin{cases} a + bF_r & \text{if } P_{group} = Water \\ F(s) & \text{if } P_{group} = Spray/Foam \end{cases} \tag{4}$$

where a is the refracted fluid color, b is the reflected scene color and F_r is the reflection factor. The reflection effect, represented by b, is produced with the sampling of a cubemap texture. $F(s)$ represents the outcome of a erosion function to that particular pixel, and is computed only for spray and foam, as explained in section 2.3.

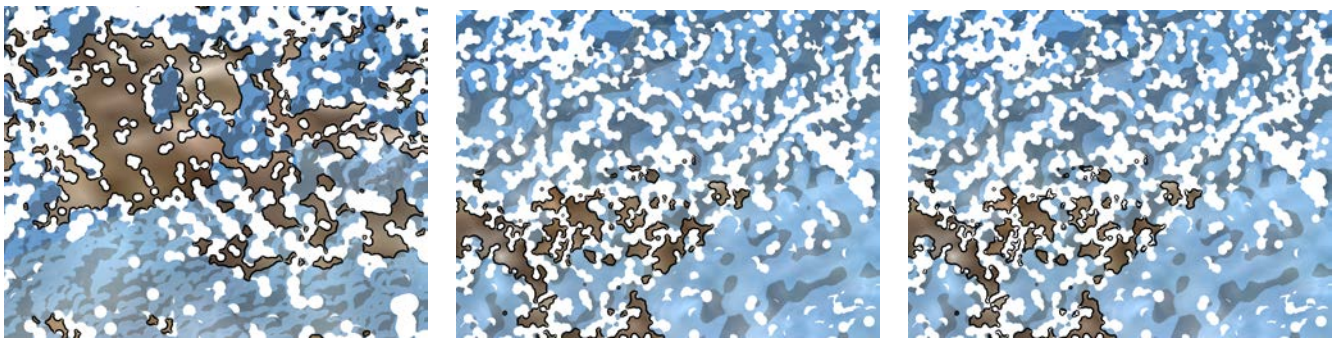
First the reflection factor F_r is obtained by a function that interpolates between reflection and refraction according to the viewers position in relation to the fluid's surface, depicting these two effects separately like traditional hand drawn animations. Next, the diffuse intensity I_d with quantized colors is obtained by a modification of the photorealistic shading model, giving the fluid a cartoon appearance. After that the refracted fluid color a is obtained by the blending of the fluid's refracted color with the background using the obtained thickness as a threshold:

$$a = lerp(I_d C_f, C_s, e^{-T(x,y)} F_r) \tag{5}$$

Finally, all acquired values are combined with the equation 4, generating the I_{wcs} value for each fragment, which are then mixed with the silhouette edges obtained using the DoG operator. For more details about this process see [Neto et al. 2013].

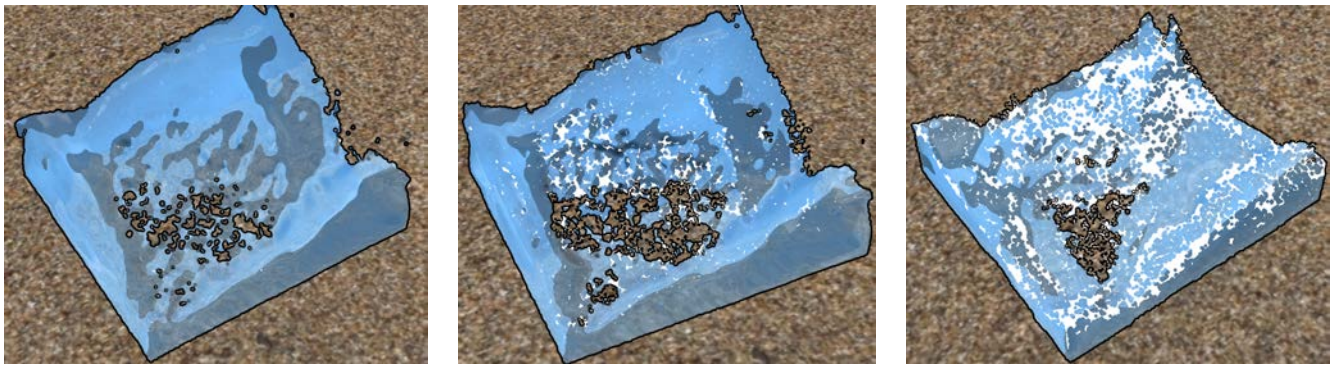
4 Results and Discussion

To evaluate the technique a series of experiments were conducted using a machine with a 3.4 GHz Intel Core i7 4770 processor, 16 GB DDR3 RAM and an NVIDIA GeForce GTX 680 in 1024 x 768 resolution. This video card has a 2048MB GDDR5 memory and 1536 CUDA cores. The development environment consisted in Windows 7, CUDA, OpenGL and GLSL. Fluid was simulated using Fluids v3 [Hoetzlein 2012], a real time SPH fluid simulator. For the tests we used two scenes: the Large Ocean Waves, composed of 262144 particles and part of the Fluids v3 environment, and the Falling Water, composed of 127832 particles and created for the tests. All parameter values used in the experiments were



(a) (b) (c)

Figure 5: *Results for different smoothing parameter values: (a) for domain σ = 0.01 and spatial σ = 2.0 a blobby aspect is clearly visible on water's surface, (b) domain σ = 0.05, spatial σ = 10.0 smooths the surface and (c) with domain σ = 0.23, spatial σ = 15.0 the high value of domain σ cause the fluid blending with the background.*



(a) (b) (c)

Figure 6: *Results for different foam and spray generation parameter values: (a) with T_{stray} = 1 and T_{foam} = 6 some stray particles are still visible and almost no foam is generated, (b) T_{stray} = 2 and T_{foam} = 10 effectively removes stray particles and creates a good amount of foam and (c) T_{stray} = 4 and T_{foam} = 15 hides particles that should have been foam or water, and generates foam where there should be water.*

obtained empirically. Figure 5 displays the visual results with different parameter values for the smoothing process, showing that a small value of spatial σ leave a blobby aspect that gradually decreases with the increase of this parameter. Increasing the value of domain σ too much, on the other hand, increases the influence of pixels that have distant depth values, causing the fluid to blend with the background. Figure 6 shows the variation of results for the particle grouping process with different parameter values. In this case, while a low value of T_{stray} keep stray particles in the final render, a value that is too high can hide more particles than needed. A similar behavior occurs with T_{foam}, but in this case increasing it's value continuously increases the amount of foam.

Performance values, including the SPH simulation cost are provided in table 1, comparing the separated bilateral filter, which can be interpreted as a single iteration, with it's iterative version with several numbers of iterations. A performance comparison between the full kernel and single iteration separated versions of the bilateral filter is shown in [Neto et al. 2013]. All the methods are compared with different kernel sizes, to check it's impact on the performance.

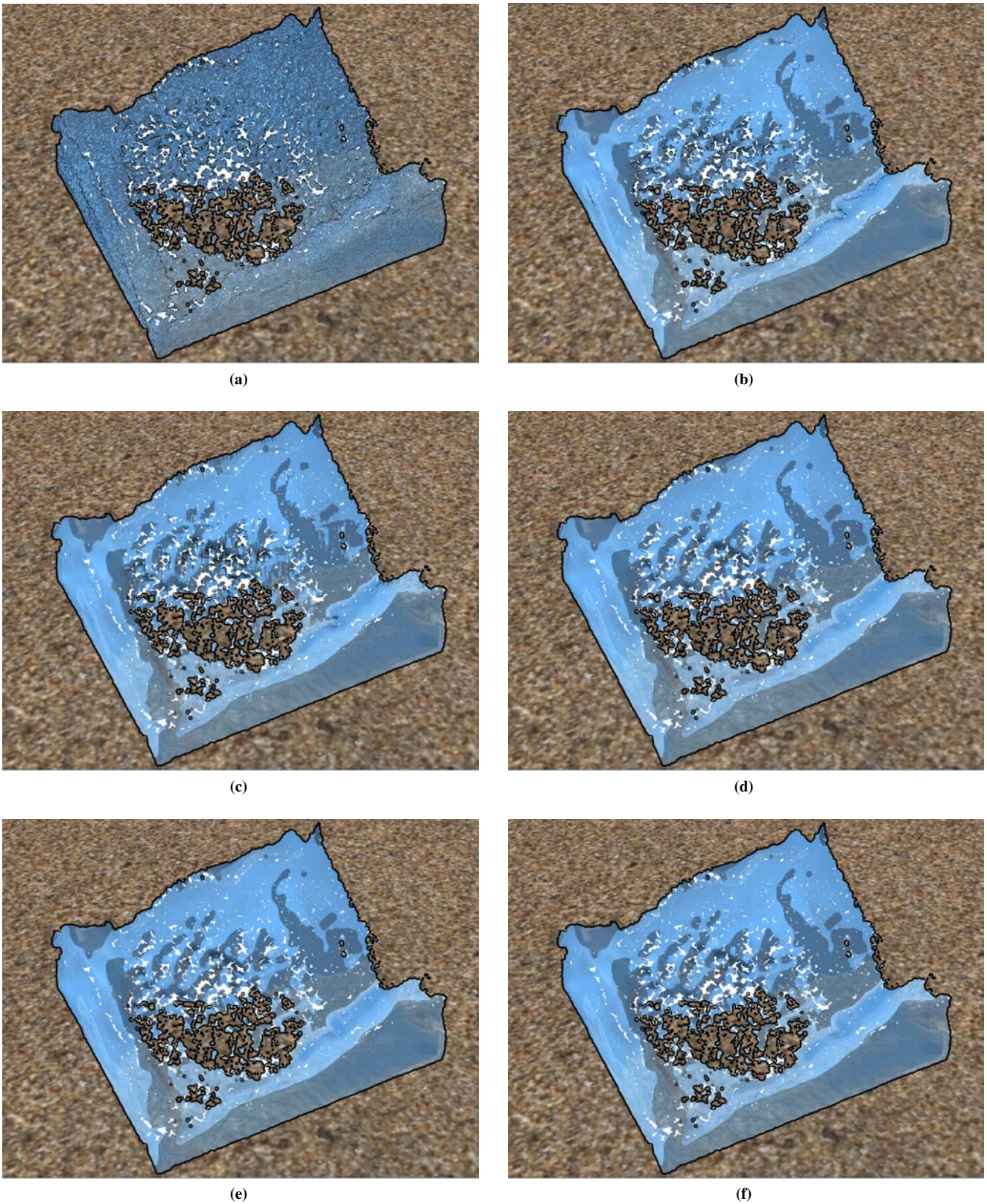
Table 1: *Performances results: expressed in FPS.*

Scene	Kernel Size	1x	2x	3x	4x
Large Ocean Waves	10x10	37	36	35	34
	20x20	36	35	34	34
	30x30	35	35	34	33
Falling Water	10x10	42	42	40	40
	20x20	41	41	40	39
	30x30	41	40	39	39

Employing multiple iterations with reduced kernel sizes in the separated version of the bilateral filter doesn't have a high impact in performance when compared with the single iteration separated bilateral filter, making this method the most appropriate for cartoon rendering. This low performance impact is explained by the linear behavior of the algorithm and the kernel reduction performed on each new iteration, making it faster than the previous one.

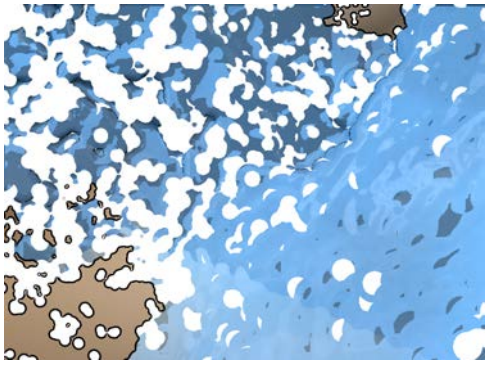
To assess the visual quality of the iterative method, it is compared with the results obtained with the full kernel and separated versions of the bilateral filter, although the performance of the full kernel filter and the artifacts generated by its single iteration separated version makes their use in practical situations unfeasible. Figure 7(a) shows the Falling Water scene without depth smoothing, while Figure 7(b)-(f) shows the same dataset rendered with different smoothing methods: full kernel, single, 2, 3 and 4 iterations separated bilateral filter, respectively, all with the same parameters. Striped artifacts generated by the single iteration separated bilateral filter can be clearly seen in Figure 7-(c), while Figure 7-(d)-(f) shows that they are gradually removed if the iterative filter is applied. Two iterations are applied to obtain the results shown in Figure 7-(d), that shows smaller artifacts when compared with the single iteration algorithm. If 3 iterations are applied (Figure 7-(e)) these artifacts become virtually unnoticeable, what suggests that the application of a fourth iteration (Figure 7-(f)) isn't necessary. A close-up of the water surface is shown on Figure 8(a)-(c), rendered with the full kernel, single iteration and three iteration methods respectively, with a kernel size of 30x30. Figure 9 shows how the surface becomes smoother as the kernel size grows when applying the algorithm, also with three iterations.

Regarding the foam and spray generation it's possible to notice that the blobby appearance of this element is seen if the viewer is close

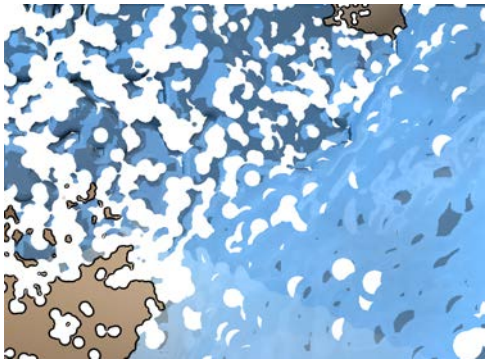


(a) **(b)** **(c)** **(d)** **(e)** **(f)**

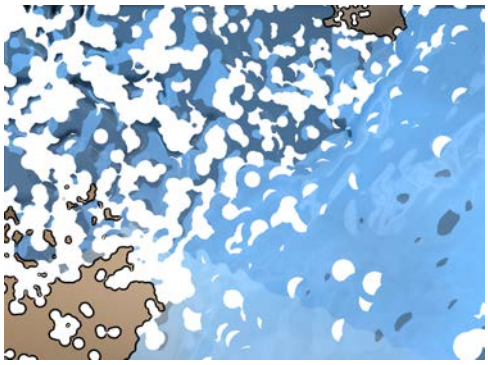
Figure 7: *The Falling Water scene rendered with different methods: (a) Without smoothing, (b) Full kernel bilateral filter, (c) Single iteration separated bilateral filter, (d) the same filter with 2 iterations, (e) 3 iterations, and (f) 4 iterations. The same parameters where used for all the methods: spatial kernel dimensions of 30x30, domain σ = 0.01, spatial σ = 10.0, T_{stray} = 2 and T_{foam} = 10.*



(a)

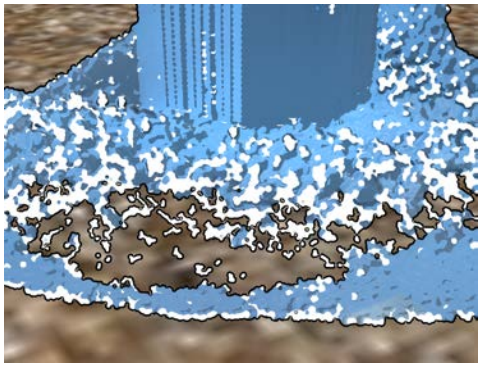


(b)

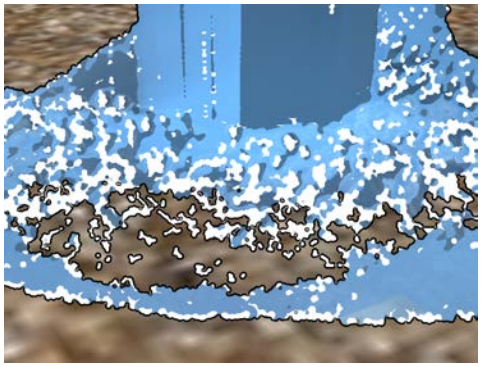


(c)

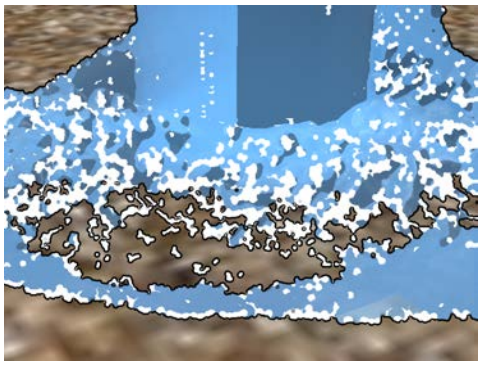
Figure 8: *A close-up on the fluid surface rendered with a kernel size of 30x30 and different methods: (a) Full Kernel Bilateral Filter, (b) Separated Bilateral Filter and (c) Multi Iteration Separated Bilateral Filter. Domain σ = 0.01, spatial σ = 10.0, T_{stray} = 2 and T_{foam} = 10.*



(a)



(b)



(c)

Figure 9: *The same scene smoothed with the iterative separated bilateral filter and different dimensions for the kernel: (a) 10x10, (b) 20x20 and (c) 30x30. Domain σ = 0.01, spatial σ = 10.0, T_{stray} = 2 and T_{foam} = 10.*

to the fluid's surface (Figure 8), suggesting that the erosion algorithm applied is not sufficient to create a natural looking foam. Another limitation of our method is the flickering caused by the foam generation. It can be explained by the separation of the particles performed with simple thresholds, causing a rough transition between water and foam. Animation results can be seen in the accompanying video.

By replacing the interpolation function from which reflection and refraction are obtained with a Fresnel effect and removing the silhouette edges, our method can also generate a photorealistic rendering, as shown on Figure 10(a)-(b). While Figure 10(a) shows that the cartoony features of our method's foam effect doesn't generate a consistent look with a photorealistic water, on Figure 10(b) we can see that this kind of rendering can be obtained with small changes on the approach.

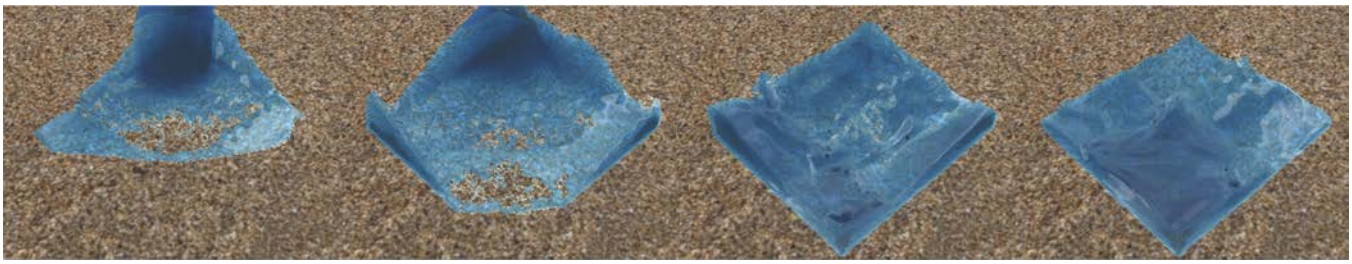
5 Conclusion and Future Works

In this paper a method that extends the real-time screen space rendering of cartoon water [Neto et al. 2013] and renders a visualization based on an SPH fluid simulation that achieves real-time performance while removing stray particles and adding spray and foam effects to the final rendering was presented. The iterative separated bilateral filter algorithm used to smooth the fluid surface is capable of generating visual results almost identical to it's full kernel version while successfully removing striped artifacts and having a low performance impact when compared with it's single iteration version. A reduction of the blobby aspect of the fluid's surface even when the viewer is close to it is also obtained with this filter. The proposed method should be able to be integrated in any game engine that employees SPH simulations to obtain the results here presented, and tests are in progress to apply it on PhysX[3] based engines.

Although the neighbor counting efficiently separates the fluid body

[3] www.geforce.com/hardware/technology/physx

(a)



(b)

Figure 10: *Two different photorealistic renderings of the Falling Water scene with reflection and refraction obtained through a fresnel effect and no silhouette edges: (a) with foam and (b) without foam.*

from spray, foam and stray particles, the method still has some limitations: the transition between fluid and foam is currently not smooth and cause flickering. Occasionally a group of particles can detach from the system and only after breaking up into smaller groups be considered stray, what makes them suddenly disappear. Investigating a way to make the particle/foam transition smoother, as well as tracking stray particles to prevent their suddenly disappearance could improve the visual results of the method. Replacing the simple neighbor counting particle separation process with a physically based approach, like the one presented by Ihmsen *et al.* [Ihmsen et al. 2012], could also benefit our method visual results. Different from our method, modern cartoons like the examples shown in Figure 2(a)-(b) depict the foam with more than one color. We leave the implementation of this behavior for future works.

Acknowledgements

The authors would like to thank Rama Hoetzlein, author of the Fluids v3 project [Hoetzlein 2012], for his support to our project since it's conception, and FAPESB, for financing the machines used in the development and tests of this work.

References

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.

BAGAR, F., SCHERZER, D., AND WIMMER, M. 2010. A layered particle-based fluid model for real-time rendering of water. In *Computer Graphics Forum*, vol. 29, Wiley Online Library, 1383–1389.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 317–324.

DURAND, F., AND DORSEY, J., 2002. Fast bilateral filtering for the display of high-dynamic-range images.

EDEN, A. M., BARGTEIL, A. W., GOKTEKIN, T. G., EISINGER, S. B., AND O'BRIEN, J. F. 2007. A method for cartoon-style rendering of liquid animations. In *Proceedings of Graphics Interface 2007*, ACM, 51–55.

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.

GASTAL, E. S. L., AND OLIVEIRA, M. M. 2011. Domain transform for edge-aware image and video processing. *ACM TOG 30*, 4, 69:1–69:12. Proceedings of SIGGRAPH 2011.

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

HOETZLEIN, R. C., 2012. Fluids v.3 - a large-scale, open source fluid simulator. Released under Z-lib license.

IHMSEN, M., AKINCI, N., AKINCI, G., AND TESCHNER, M. 2012. Unified spray, foam and air bubbles for particle-based fluids. *The Visual Computer 28*, 6-8, 669–677.

KELLOMÄKI, T. 2012. Water simulation methods for games: a comparison. In *Proceeding of the 16th International Academic MindTrek Conference*, ACM, 10–14.

KYPRIANIDIS, J. E., AND KANG, H. 2011. Image and video abstraction by coherence-enhancing filtering. In *Computer Graphics Forum*, vol. 30, Wiley Online Library, 593–602.

KYPRIANIDIS, J. E., KANG, H., AND DÖLLNER, J. 2009. Image and video abstraction by anisotropic kuwahara filtering. In *Computer Graphics Forum*, vol. 28, Wiley Online Library, 1955–1963.

KYPRIANIDIS, J., COLLOMOSSE, J., WANG, T., AND ISENBERG, T. 2013. State of the" art": a taxonomy of artistic stylization techniques for images and video. *IEEE transactions on visualization and computer graphics 19*, 5, 866.

LIAO, J., YU, J.-H., AND JIA, L. 2011. Procedural modeling of water caustics and foamy water for cartoon animation. *Journal of Zhejiang University SCIENCE C 12*, 7, 533–541.

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.

MARR, D., AND HILDRETH, E. 1980. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences 207*, 1167, 187–217.

MCGUIRE, M., AND FEIN, A. 2006. Real-time rendering of cartoon smoke and clouds. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, ACM, 21–26.

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.

NETO, L., DEÓ, F., APOLINÁRIO, A., AND MELLO, V. 2013. Real-time screen space rendering of cartoon water. In *SBGames 2013 - Trilha de Computação*.

OJEDA CONTRERAS, J., ET AL. 2013. Efficient algorithms for the realistic simulation of fluids.

PARIS, S., KORNPROBST, P., TUMBLIN, J., AND DURAND, F. 2007. A gentle introduction to bilateral filtering and its applications. In *ACM SIGGRAPH 2007 courses*, ACM, 1.

PHAM, T. Q., AND VLIET, L. J. 2005. Separable bilateral filtering for fast video preprocessing. In *In IEEE Internat. Conf. on Multimedia & Expo, CD14*, IEEE, 1–4.

SELLE, A., MOHR, A., AND CHENNEY, S. 2004. Cartoon rendering of smoke animations. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, ACM, 57–60.

SERRA, J. 1982. *Image analysis and mathematical morphology, v. 1*. Academic press.

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

TOMASI, C., AND MANDUCHI, R. 1998. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, IEEE, 839–846.

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.

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

WINNEMÖLLER, H. 2011. Xdog: advanced image stylization with extended difference-of-gaussians. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*, ACM, 147–156.

YINGST, M., ALFORD, J. R., AND PARBERRY, I. 2011. Very fast real-time ocean wave foam rendering using halftoning. *Proceedings of the 6th International North American Conference on Intelligent Games and Simulation (GAMEON-NA)*, 27–34.

YOU, M., PARK, J., CHOI, B., AND NOH, J. 2009. Cartoon animation style rendering of water. In *Advances in Visual Computing*. Springer, 67–78.

YU, J., JIANG, X., CHEN, H., AND YAO, C. 2007. Real-time cartoon water animation. *Computer Animation and Virtual Worlds 18*, 4-5, 405–414.

ZHANG, L., GUO, C., TANG, Y., LV, M., LI, Y., AND ZHAO, J. 2014. Real-time and realistic simulation of large-scale deep ocean wave foams based on gpu. *Journal of Computational Information Systems 10*, 7, 28212828.

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.