Eurographics Symposium on Rendering 2010
Jason Lawrence and Marc Stamminger
(Guest Editors)

Volume 29 (2010), Number 4

A Layered Particle-Based Fluid Model for Real-Time Rendering of Water

Florian Bagar †, Daniel Scherzer and Michael Wimmer ‡

Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria

Abstract

We present a physically based real-time water simulation and rendering method that brings volumetric foam to the real-time domain, significantly increasing the realism of dynamic fluids. We do this by combining a particle-based fluid model that is capable of accounting for the formation of foam with a layered rendering approach that is able to account for the volumetric properties of water and foam. Foam formation is simulated through Weber number thresholding. For rendering, we approximate the resulting water and foam volumes by storing their respective boundary surfaces in depth maps. This allows us to calculate the attenuation of light rays that pass through these volumes very efficiently. We also introduce an adaptive curvature flow filter that produces consistent fluid surfaces from particles independent of the viewing distance.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Dynamic fluids are a desirable element of many real-time applications like games. So far, the mathematical complexity of realistically simulating and rendering the behavior and interaction of fluids with the environment has hindered their widespread use. One promising approach would be to render the results of smoothed particle hydrodynamics (SPH) simulations using splatting, but the locally high curvature of spherical splatting primitives results in an unrealistic jelly-like appearance.

Only recently, van der Laan et al. [vdLGS09] proposed curvature-based screen space filtering for rendering the result of SPH simulations. The approach alleviates sudden changes in curvature between the particles and creates a continuous and smooth surface. While this method is a significant step towards realistic fluid rendering in real time, there is room for improvement. First, the screen-space curvature flow formulation is highly dependent on viewer distance. While fluids farther away from the viewer are overly smoothed, fluids near the viewer almost completely retain the undesirable spherical particle structure. Second, there exists as yet no realistic real-time method to create foam, which is an important visual element in most situations where real-time fluids are used (see Figure 1).

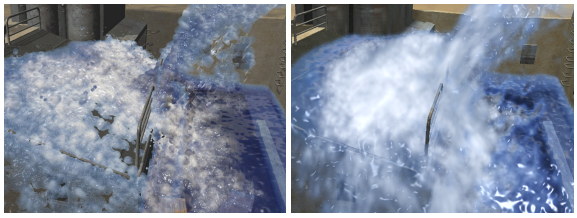

Figure 1: *A scene rendered with simple noise-based foam [vdLGS09]* (left) *and with our new method* (right)*;*

This paper presents a real-time fluid simulation and rendering system that overcomes these drawbacks:

- We introduce an *adaptive curvature flow* filtering algorithm for SPH rendering which accounts for perspective.
- We introduce a *physically based foam rendering* method using Weber number thresholding and a volumetric layer-based rendering system (see Figure 2). Foam can appear

† e-mail: florian.bagar@aon.at
‡ e-mail: scherzer | wimmer@cg.tuwien.ac.at

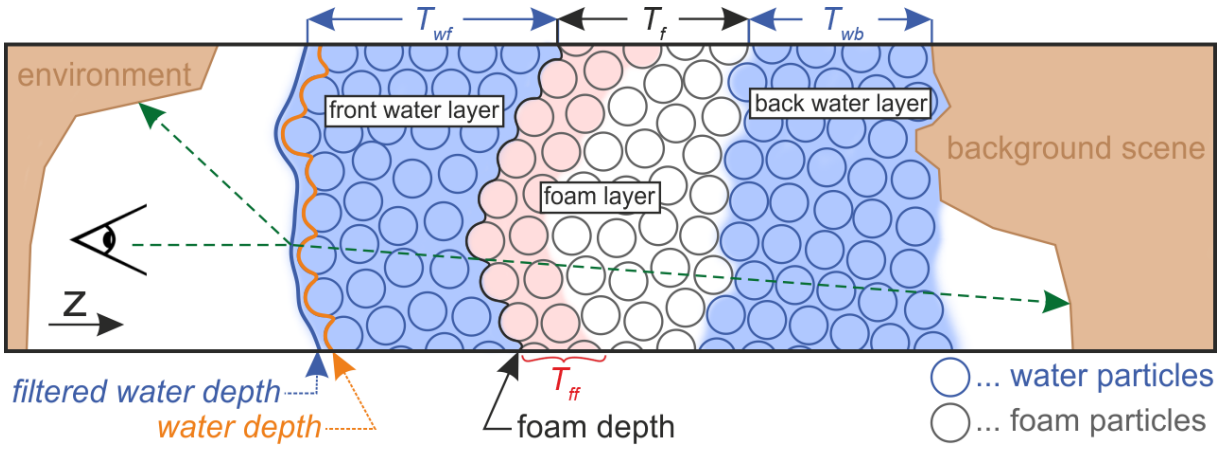

Figure 2: *A cross-section of our layered water model: The volumetric appearance of the result is achieved by not only accounting for the water thickness T_{wb} at each pixel as previous approaches [vdLGS09], but also for the foam thickness T_f and the thickness of water in front of the foam T_{wf}. We also partition foam into two differently colored layers (T_{ff}) to achieve more interesting foam.*

as the top-most layer or between two water layers, as in a waterfall.

- Our method is almost as fast as previous approaches, while providing higher image quality.
- In addition, the algorithm is simple to implement and integrate into existing rendering engines.

2. Previous Work

Simulation of liquids like water can be classified into Eulerian- and Lagrangian-approaches. The former build on a fixed grid in space, using finite element techniques [Sta99] to solve the Navier Stokes Equations. However, these approaches are not intuitive for flows because they limit the simulation to the space where the grid is defined. Lagrangian-approaches, like Smoothed Particle Hydrodynamics (SPH), introduced for computer graphics by [DG96], simulate a fluid by moving discrete volume elements, and are therefore not restricted concerning the simulation space.

Offline methods include effects like foam, bubbles and spray, and we adapt some of their elements for our real-time method. Losasso et al. [LTKF08] mix the Eulerian and Lagrangian approaches to generate realistic fluids including spray and foam. Mihalef et al. [MMS09] adapt this method and replace the Lagrangian SPH approach by a simple particle system to include droplets and bubbles. The so-called Weber number, as defined in [Sir99], is used to control the generation of droplets and bubbles, which we adapt for creating foam in real time. Takahashi et al. [TFK*03] use a particle-based approach to model splashes and foam. The generation and transition of foam is controlled using state change rules, which work in a similar way as our separation of the particles into water- and foam-particles. Cleary et al. [CPPK07] extend SPH by considering the dissolved gas within the fluid. Similar to our work they use a layered representation where the different parts of the fluid volume are separately rendered and composed into the final image.

Current *real-time approaches* are usually limited in the number of particles they can handle, and do not include realistic foam [MCG03]. One way to render the water surface from the results of the particle simulation are Müller et al.'s [MSD07] *screen space meshes*, created using a marching squares technique on the particle depth map. Although the algorithm provides view-dependent level of detail and filtering in screen space, rendering foam with this approach is prohibitive, because of the large amount of geometry that needs to be generated. Thürey et al. [TSS*07] present a shallow water-based particle model that is coupled with a SPH simulation to simulate bubbles and foam effects, but the method simulates individual foam particles, which is expensive.

van der Laan et al. [vdLGS09] present an approach for rendering particle-based fluids directly using splatting instead of performing a polygonization. They use screen space curvature flow filtering to conceal the sphere geometry of the particles and to prevent the fluid from looking jelly-like. However, the curvature flow filtering is dependent on the view distance, and the proposed simple noise-based surface foam effect does not have a volumetric appearance (see Figure 1).

3. Overview

Our method builds on the screen space fluid rendering approach with curvature flow [vdLGS09]. Similar to this method, we start from an SPH simulation calculated using

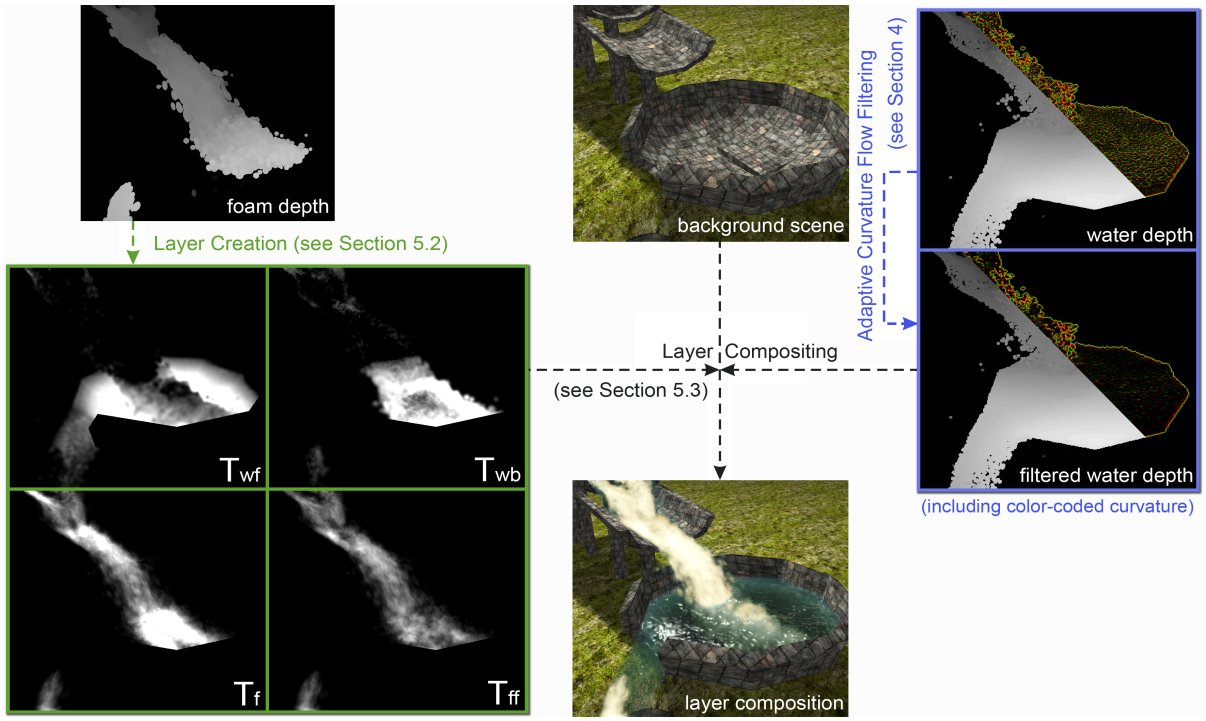

Figure 3: *Overview of the buffers used in our method. T_{wf}: thickness of the water in front of the foam; T_{wb}: thickness of the water behind the foam; T_f: foam thickness; T_{ff}: thickness of the front foam layer;.*

a hardware physics engine (PhysX), which provides a non-sorted 3D point cloud as input. Apart from the particle's position x we will also use the density ρ and velocity v for foam thresholding and the lifetime for varying the Perlin noise on a foam particle.

The original algorithm calculates the water depth by splatting the particles, then smooths the depth buffer using curvature flow filtering, then calculates water thickness by accumulating particle depths in a separate thickness buffer, and finally composites the results. Our algorithm extends this by adapting the curvature flow filter for the viewer distance, and by adding a foam layer that can lie between two water layers. Our algorithm then performs the following steps once per frame after the scene has been rendered into a texture (see Figure 3):

1. Calculate water and foam depth (Section 5.2)
2. Smooth the water depth using the new adaptive curvature flow algorithm (Section 4)
3. Calculate water and foam thickness (Section 5.2)
4. Composite water and foam layers using intermediate results (Section 5.3)

4. Adaptive Curvature Flow

The first step in rendering a fluid using particles is to create the fluid surface. This is done by splatting the particles into the depth buffer using point sprites, with the depth values replaced by the geometry of a sphere. In order to avoid a "jelly-like" appearance due to the spherical particles, it is important to *smooth* the depth surface. van der Laan et al. [vdLGS09] argue that (bilateral) Gaussian filters are too expensive because they are not separable. Instead, they use *curvature flow* [MS97], which is a method that repeatedly shifts a surface along its normal vector depending on the mean curvature of the surface. This process corresponds to an Euler integration, and causes smoothing, as it tries to minimize curvature.

Under certain assumptions and simplifications, van der Laan et al. derive a formulation of curvature flow in *screen space* that moves the z-value according to a simple Euler integration of the differential equation

$$\frac{\partial z}{\partial t} = H, \tag{1}$$

where H is the mean curvature of a pixel, which is a function of the partial derivatives at the pixel and the depth value itself (for details see [vdLGS09]):

$$H\left(\frac{\partial z}{\partial x_s}, \frac{\partial z}{\partial y_s}, \frac{\partial^2 z}{\partial x_s^2}, \frac{\partial^2 z}{\partial y_s^2}, z\right) \tag{2}$$

where x_s, y_s are the *window coordinates* of the current pixel. The partial derivatives are calculated using simple finite dif-

ferencing, so the discrete form of the mean curvature H_s depends on a 4-neighborhood around the current pixel:

$$H_s(z(x_s,y_s),z(x_s-1,y_s),z(x_s+1,y_s),z(x_s,y_s-1),z(x_s,y_s+1)) \quad (3)$$

Each iteration of the Euler integration simply renders a full-screen pass with $z_{i+1} = z_i + \Delta t H_s$. As proposed by [vdLGS09], the number of iterations are chosen depending on the smoothness that is desired.

However, close inspection of the screen-space curvature formulation reveals that the reference coordinate system for calculating the curvature is the window coordinate system. This means that, given equal iteration sizes, a particle that appears larger on screen (because it is closer due to perspective) will have a significantly larger radius in this coordinate system and therefore significantly lower curvature than a particle that is farther away. The resulting artifact is that smoothing will have a lower effect on closer particles, which therefore retain the unwanted spherical appearance, whereas particles far from the viewer will be overly smoothed, so that the fluid surface loses its defining characteristics such as highlights. This can be observed in Figure 4, left.

Figure 4: *In [vdLGS09]* (left)*, distant water is over-smoothed (top) and near water is under-smoothed (bottom). Our new method* (right) *maintains the same amount of smoothing regardless of the distance.*

One possible solution would be to remap the curvatures into a common reference coordinate system, for example by dividing H_s by z for each evaluation of H_s. However, our experiments have shown that this makes the integration very unstable, because the screen-space evaluation for larger particles is very noisy due to depth quantization. On the other hand, depth correction would make the resulting curvatures large in magnitude, leading to oscillation.

Therefore, we approach the problem from a different direction and interpret each integration step as a filtering step with a 3x3 kernel. Obviously, repeated filtering leads to an increased screen-space kernel radius r_s of the hypothetical overall filter – in fact, the number of iterations corresponds exactly to r_s. The main idea is now to vary the number of iterations depending on the view space distance z in order to obtain a roughly equal overall filter kernel size r_w in world space, making sure that a similar world-space neighborhood is taken into account when calculating the curvature flow. So r_s can be calculated from a desired world-space kernel radius r_w through

$$r_s = \frac{r_w}{z}\frac{FV}{2} \quad (4)$$

where F is the focal length, V is the viewport width in pixels, and z is the eye-space z-distance. So in iteration i, an Euler iteration step is only applied to a pixel if $r_s < i$. For optimization, the user can specify a maximum iteration count. Furthermore, an occlusion query is issued for each iteration to check whether any depth value was actually modified. If that is not the case, all pixels are already converged and no further iteration is necessary.

5. Real-Time Foam

In this section we describe how to incorporate foam into real-time fluid rendering. We define water foam as a substance that is formed by trapping air bubbles in the liquid. Form is usually observed as spray or bubbles above the surface of a turbulent water stream. However, we also observed that a significant visual effect is caused by foam that occurs behind a water surface, usually due to a turbulent water stream that immerges into resting water with high impact (see Figure 5).

In order to capture these two main effects in a real-time setting, we separate foam particles from water particles and arrange the resulting foam and water particles in separate layers and render them using volumetric back-to-front compositing. Although our layered representation does not account for discontinuity in the fluid volume which occurs if there are several layers of water and foam, the two most common cases mentioned above are covered by this model.

5.1. Foam Formation

First, we classify particles as water or foam. Following [MMS09], we base the classification on the Weber number [Sir99], which is a dimensionless physical quantity that describes the relative influence of the inertia of a fluid to its surface tension. The Weber number is defined as the ratio of the kinetic energy to the surface energy:

$$We = \frac{\rho v^2 l}{\sigma}, \quad (5)$$

where ρ is the density, v is the relative velocity between the liquid and the surrounding gas, l is the characteristic length, and σ is the surface tension. For larger We, the kinetic energy of the water is greater than the surface energy, causing water to intermix with the surrounding air, which results in foam formation. Thus, we separate particles into water and foam particles by thresholding the Weber number. In practice, we use a linear transition area where the particle is counted both as water and foam particle to ensure a smooth emergence and disappearance of foam. The new foam particle starts out as

a point and expands, while the corresponding water particle shrinks.

Similar to [MMS09], we assume that the surface tension and the characteristic length are fixed for the SPH simulation. We also assume that the characteristic length l is the particle diameter, which is a simplification for our real-time purposes, and that the surrounding air is not moving, and therefore the relative velocity v is the velocity of the particles. The velocity v and the density ρ are obtained from the physics simulation package.

5.2. Layer Creation

Now that particles have been classified as either particle or foam, we partition the fluid into *layers*, as shown in Figure 2.

By using two water layers, one in front and one behind the foam layer, we can simulate foam inside water, as happens at the end of a waterfall (see for instance Figure 8, *middle* or Figure 5). We first determine the front water surface and the front foam surface by splatting water and foam particles into separate depth buffers (the splatting step was described in Section 4). Curvature flow is only applied to the front water surface.

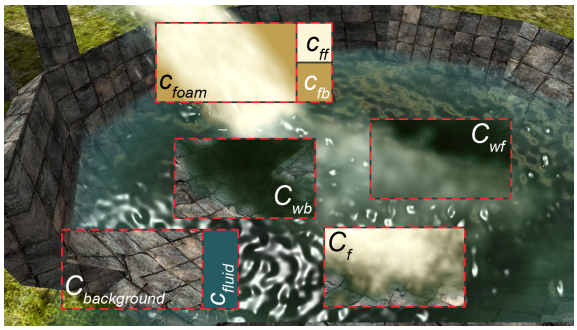

Figure 5: *User defined colors (c_{fluid}, c_{ff}, c_{fb}) and resulting colors from the compositing steps ($C_{background}$, C_{wb}, C_f, C_{wf}).*

Since water and foam are volumetric phenomena, the amount of water respectively foam between two layer surfaces needs to be determined in order to allow correct compositing and attenuation. Similar to [vdLGS09], the thickness of a layer is determined by additively splatting every particle belonging to the volume into a buffer. In contrast to the depth surface calculation, the splat kernel gives the thickness of the particle at each particle sampling point. Accumulating particle thicknesses is a reasonable approximation because particles from the physics simulation can be assumed to be largely non-overlapping.

$$T(x,y) = \sum_{i=0}^{n} t\left(\frac{x-x_i}{\sigma_i}, \frac{y-y_i}{\sigma_i}\right) \tag{6}$$

where t is the particle thickness function, x_i, y_i are the projected position of the particle, x and y are screen coordinates and σ_i is the projected size. In comparison to [vdLGS09], we not only calculate the water thickness, but also:

- the foam thickness T_f, by considering only foam particles. For the foam particles, the splat kernel is also multiplied with a 3D Perlin noise function (*noise*3), which is varied with the lifetime of the particle, to add sub particle detail, giving $noise3(x, y, lifetime)$.
- the front water thickness T_{wf}, by considering only water particles that are in front of the foam layer (by comparing the particle depth with the depth of the front foam surface).
- the back water thickness T_{wb}, by considering the other water particles.
- the thickness in a constant range behind the foam surface T_{ff} (also multiplied with *noise*3), to allow blending of two different foam colors.

5.3. Layer Compositing

Finally, to account for the attenuation caused by the previously calculated layers, the actual pixel color is calculated by volumetric compositing. Figure 3 gives an overview of the buffers that are used for the compositing.

Compositing along a viewing back to front, we have (see Figure 2):

$$C_{wb} = \text{lerp}(c_{fluid}, C_{background}, e^{-T_{wb}}) \tag{7}$$
$$C_f = \text{lerp}(c_{foam}, C_{wb}, e^{-T_f}) \tag{8}$$
$$C_{wf} = \text{lerp}(c_{fluid}, C_f, e^{-T_{wf}}) \tag{9}$$

where c_{foam} and c_{fluid} are the colors of the medium. Figure 5 shows the individual steps and colors used in the compositing.

In addition to attenuation, we also calculate reflection with a Fresnel Term, and a highlight at the front water surface, as well as refraction, similar to [vdLGS09]. For reflection and highlight, care needs to be taken because the front water surface might be behind the foam. So if $T_{wf} = 0$, we have $C_{wb} = illuminate(C_{wb})$, otherwise $C_{wf} = illuminate(C_{wf})$, where

$$illuminate(x) = x(1 - F(\mathbf{nv})) + rF(\mathbf{nv}) + k_s(\mathbf{nh})^{\alpha}, \tag{10}$$

i.e., the standard Fresnel (F) reflection calculation (r is a lookup into an environment map) and a Phong term. We also carry out refraction for the whole water surface, so $C_{background}$ is sampled from the scene background texture perturbed along the normal vector, scaled using $T_{wb} + T_{wf}$ (for details see [vdLGS09]).

Finally, we have found that foam can be made to look more realistic by blending two different user defined colors,

c_{ff} and c_{fb}. The thickness T_{ff} is calculated along with the foam thicknesses T_f by accumulating just the foam particles within a user defined constant range δ_{foam}. So c_{foam} is actually calculated as $c_{foam} = \text{lerp}(c_{fb},\ c_{ff}, e^{-T_{ff}})$. By considering only foam particles which are close behind the foam surface, we obtain a foam pattern that has a controllable thickness, achieving the benefit that the pattern moves along with the particles and exhibits fine micro-structures in its visual appearance.

6. Results

We have tested our approach in three scenes (see Figure 8): *Corridor* has many obstacles and therefore creates a turbulent water flow with a lot of foam and spray. *Waterfall* is less turbulent, but due to its simplicity, artifacts are easily detected by visual inspection. Here, rendering of foam is essential for realistic results. *Bamboo* has dynamic elements that interact with the water. The bamboo is slowly filled with water, till the water weights it down and is emptied again. Please see the video for more details.

We have used an Intel Q9450 CPU with a GeForce GTX 280 graphics card. The SPH simulation was done with NVIDIA PhysX. Particle counts range from 20k to 64k, depending on the scene. All images were taken at 1280 x 720 resolution. The curvature flow filtering step was done at half resolution. We use off-screen buffers to store our various intermediate results: 32 bit float for the water depth, 16 bit float for the foam depth, and 16 bit each for T_{wb}, T_{wf}, T_f and T_{ff}. This results in a total of 112 bit per pixel.

Scene	[vdLGS09]	without foam	with foam
Waterfall	14 ms	14 ms	16 ms
Corridor	11 ms	12 ms	15 ms
Bamboo	12 ms	13 ms	17 ms

Table 1: *Performance comparison between [vdLGS09] (without foam) and our method with and without foam.*

Table 1 compares the *computational cost* of [vdLGS09] with our method (SPH simulation time not included). The indicated running times are an average for a default camera movement. Our method has comparable performance with the benefit of improved image quality especially at near or far viewpoints. Even foam does not significantly increase running time for our method. Figure 6 presents the *computational cost* of [vdLGS09] and our method using the example of a camera zoom movement in the waterfall scene (like the one of the filter comparison shown in the accompanying video). It takes on average 23.12% of the computation time to render the water and foam depth, 24.4% for the thickness passes, 27.43% for the adaptive curvature flow filtering and 25.05% for the composition (including update of data structures). This measurement represents the mean breakdown of 6k frames using different viewpoints. Figure 1

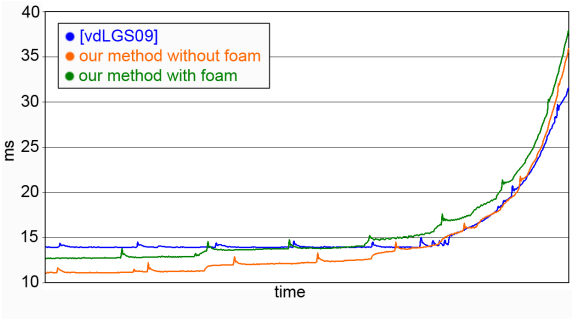

Figure 6: *Performance comparison of a camera zoom movement in the waterfall scene.*

shows the benefit of our physically guided foam generation over simple noise-based foam [vdLGS09]. Figure 7 demonstrates that foam is an important visual element when rendering fluids. Figure 9 compares a photograph of a real waterfall with our method. As one can observe, the foam is visible below the surface when a turbulent water stream immerges into resting water. Dynamic visual results can be observed in the accompanying video.

7. Conclusions and Future Work

We presented a new method for rendering particle-based fluids with foam in real time. The first contribution is an adaptive curvature flow smoothing method that avoids over- or under-smoothing as present in previous methods. Our second contribution is a fast physically guided foam rendering algorithm based on Weber number thresholding and a layered compositing algorithm. Our approach provides more realistic fluid rendering at comparable cost to previous methods, and is simple to implement and integrate into existing engines. In future work, we plan to use the volumetric information available in the layers to generate soft shadows. We will also investigate whether situations that require more than 3 layers are likely to appear.

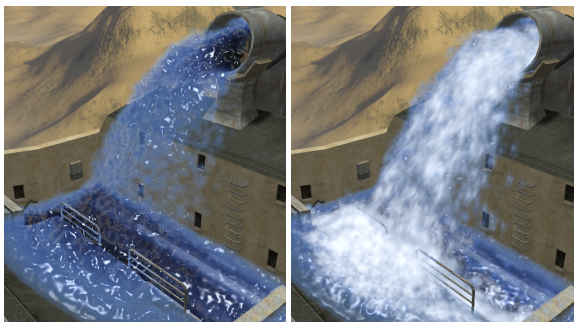

Figure 7: *Corridor scene without/with foam (26–50 iterations).*

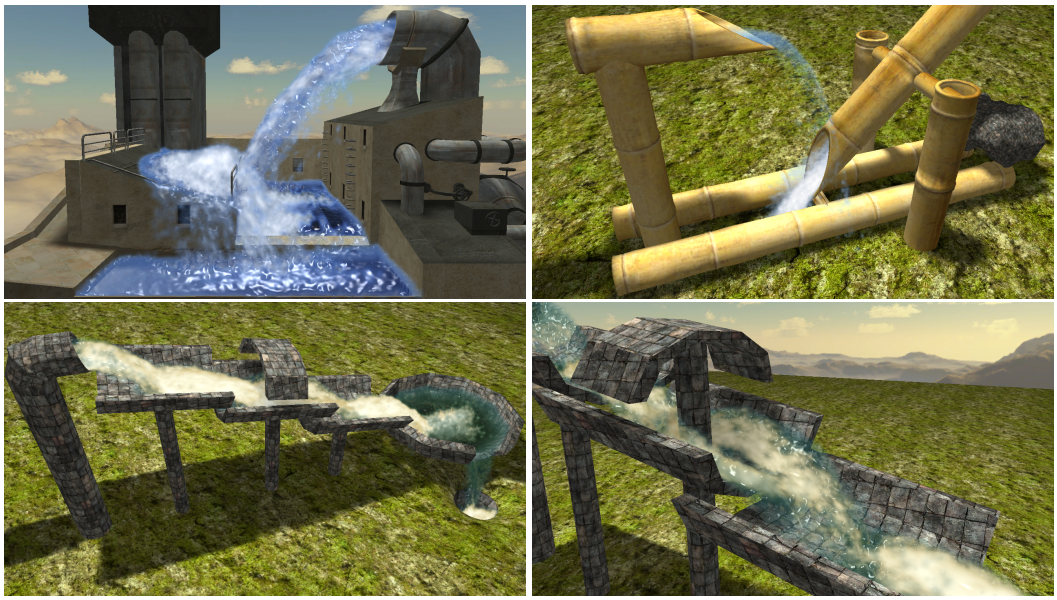

Figure 8: *Our three test scenes: at the* top-left: *Corridor (27–52 iterations) ;* right: *Bamboo (22–40 iterations); and at the* bottom: *Waterfall (*left: *15–20 iterations;* right: *20–44 iterations).*

Figure 9: *Comparison between a photograph of a real waterfall (left) and our new method (right). The rectangle marks an area where foam occurs below the water surface.*

References

[CPPK07] CLEARY P. W., PYO S. H., PRAKASH M., KOO B. K.: Bubbling and frothing liquids. *ACM Trans. Graph. 26*, 3 (2007), 97. 2

[DG96] DESBRUN M., GASCUEL M.-P.: Smoothed particles: a new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96* (1996), Springer-Verlag New York, Inc., pp. 61–76. 2

[LTKF08] LOSASSO F., TALTON J., KWATRA N., FEDKIW R.: Two-way coupled sph and particle level set fluid simulation. *IEEE Transactions on Visualization and Computer Graphics 14*, 4 (2008), 797–804. 2

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2003), Eurographics Association, pp. 154–159. 2

[MMS09] MIHALEF V., METAXAS D. N., SUSSMAN M.: Simulation of two-phase flow with sub-scale droplet and bubble effects. *Comput. Graph. Forum 28*, 2 (2009), 229–238. 2, 4, 5

[MS97] MALLADI R., SETHIAN J. A.: Level set methods for curvature flow, image enhancement, and shape recovery in medical images. 329–ff. 3

[MSD07] MÜLLER M., SCHIRM S., DUTHALER S.: Screen space meshes. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2007), Eurographics Association, pp. 9–15. 2

[Sir99] SIRIGNANO W. A.: Fluid dynamics and transport of droplets and sprays, 1999. Incluye referencias bibliográficas (p. 287-307) e índice (p. 309-311). 2, 4

[Sta99] STAM J.: Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (1999), ACM Press/Addison-Wesley Publishing Co., pp. 121–128. 2

[TFK*03] TAKAHASHI T., FUJII H., KUNIMATSU A., HIWADA K., SAITO T., TANAKA K., UEKI H.: Realistic animation of fluid with splash and foam. *Comput. Graph. Forum 22*, 3 (2003), 391–400. 2

[TSS*07] THÜREY N., SADLO F., SCHIRM S., MÜLLER-FISCHER M., GROSS M.: Real-time simulations of bubbles and foam within a shallow water framework. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2007), Eurographics Association, pp. 191–198. 2

[vdLGS09] VAN DER LAAN W. J., GREEN S., SAINZ M.: Screen space fluid rendering with curvature flow. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games* (2009), ACM, pp. 91–98. 1, 2, 3, 4, 5, 6