Vol. [VOL], No. [ISS]: 1–15

Instant Liquids: Fast Screen-Space Surface Generation for Particle-Based Liquids

Hilko Cords and Oliver Staadt
University of Rostock, Germany
Institute of Computer Science
Visual Computing Group

Abstract. We present Instant Liquids, a fast GPU-based method for rendering boundary surfaces from 3D particle-based fluid simulations. Instead of extracting and tracking polygonal meshes of boundary surfaces in 3D world space, we devise a new meshless 2D screen-space approach. First, we generate a smooth view-dependent depth map from the 3D particle cloud. Then, surface normals are derived directly from the depth map and lighting calculations are carried out in screen-space. We extend this basic algorithm with efficient reflection and refraction approximations. The advantage of our method over existing rendering approaches for particle-based fluids models is the fact that we do not have to extract 3D or 2D meshes at any time. Thus, we improve visualization time for particle-based liquids to a level were it falls significantly below simulation time. Furthermore, effects such as breaking waves and splashes are supported automatically.

1. Introduction

The use of physics-based liquid simulations has become a staple of visual effects and computer animation. Complex and photo-realistic liquid effects can be simulated and rendered with high quality, but simulation and rendering

1086-7651/06 $0.50 per page

time takes many seconds if not minutes per frame. In general, this process involves three steps: (i) simulation, (ii) surface extraction, and (iii) rendering. Underlying physics is simulated by solving, or at least approximating, the Navier-Stokes equations using a grid-based Eulerian (e.g., [Stam 99]) or a 3D particle-based Lagrangian approach, e.g. Smoothed Particle Hydrodynamics (SPH, [Müller et al. 03]). SPH finds a particle-based solution to the 3D Navier-Stokes equations by approximating the local physical properties of liquids using local neighborhood comparisons. The use of GPU-based simulations have lead to substantial improvements of simulation performance. Thus, the bottleneck in fluid animation has been shifted from simulation to surface extraction and rendering. Hence, the need for fast rendering methods handling tens of thousands of particles in real-time has increased immensely over the last few years. In off-line environments, a detailed polygonal mesh is extracted from particle cloud (e.g., marching cubes [Lorensen and Cline 87]) and frequently, ray tracing is used to render the resulting surface. However, this process is too slow for interactive applications such as computer games.

Highly realistic and detailed liquid surfaces can be achieved using height field-based approaches, e.g., [Tessendorf 04]. However, more complex behavior of liquids, such as splashes and breaking waves, cannot be represented adequately as height fields. Therefore, 3D surfaces are desirable to support such effects. Recently, the concept of screen-space meshes was presented in [Müller et al. 07]. The complexity of a volume discretization in world space is reduced to an area discretization in screen-space. The liquid is sampled CPU-based by a modified marching squares algorithm with plausible results even in real-time environments. The screen-space meshes are related to our technique and the creation process of the basic depth map is similar. However, it differs completely in the surface generation, normal extraction, and rendering steps.

We present *Instant Liquids*, a view-dependent screen-space rendering method for highly-dynamic transparent liquids (e.g., water) based on 3D particle-based simulation. We use the depth map of a particle cloud to approximate the boundary surface and determine the normals in screen-space. We avoid the costly step of extracting a polygonal surface approximation for every step of the simulation and, thus, increase overall performance substantially. Using screen-space projection, a 3D liquid can be rendered including effects such as breaking waves and splashes, and approximations of reflection and refraction. It is important to note that our scheme can be implemented entirely on the GPU, freeing the CPU for numerical simulation calculations.

We demonstrate the power of our approach in a number of application examples, including examples from real-time SPH simulations, mass-point schemes, and breaking waves. As shown in Section 5, the high performance of *Instant Liquids* makes liquid simulation and rendering in today's computer games or virtual reality applications possible. A video demonstrating our technique in practice is available online.

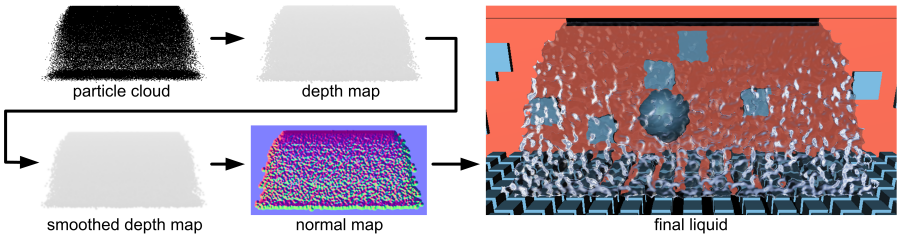


Figure 1. *Instant Liquids*: The main steps (waterfall, front view).

2. Instant Liquids – Basic Algorithm

We take as input the positions of an unstructured 3D particle cloud $\mathbf{x}_i, ..., \mathbf{x}_N \in \mathbb{R}^3$ and the homogeneous projection matrix $\mathbf{P} \in \mathbb{R}^{4\times4}$. Additionally, the point splat radius r_p and the filter kernel size k can be used to control the visual behavior of the liquid, but could also be estimated within a pre process. The outline of the basic algorithm, which is implemented as a fragment shader, is as follows (Figure 1):

1. Create depth map from 3D particle cloud
2. Smooth depth values
3. Extract normal map from smoothed depth map
4. Perform per-fragment lighting

These four steps are repeated every time a particle position or the camera position changes. Note that the smoothed depth map can also be interpreted as a dynamic height field viewed from above. Thus, the process of normal map extraction is similar to extracting a normal map from a height field. We elucidate additional view-dependent optical effects such as reflection and refraction in Section 3.

2.1. Depth Map Creation

We first generate a depth map $\mathbf{Z} \in \mathbb{R}^{W\times H}$ with depth values $z_{i,j}$ and resolution $W \times H$ as depicted in Figure 1. For each particle with position $\mathbf{x}_i$, we render a circular point splat into the frame buffer. To avoid the known non-linearity of z-values in screen-space in the case of perspective projection, we transform the homogeneous coordinates of x, y, z, w to screen-space coordinates x_p, y_p, z_p:

$$\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = \begin{pmatrix} W \cdot (\frac{1}{2} + \frac{x/w}{2}) \\ H \cdot (\frac{1}{2} + \frac{y/w}{2}) \\ z \end{pmatrix}, \tag{1}$$

were W denotes the horizontal and H the vertical resolution of the screen-space. As a result, the z-coordinate z_p is linear and describes the distance to the camera.

The point splat radius r_p influences the visual appearance of the liquid. In general, the radius can be determined according to distances of neighboring particles in screen-space, however, this would require expensive topological neighborhood searches. Since we aim at dynamic liquids, we estimate r_p within a preprocess. Particle radius r is determined in world space for the calm liquid, depending on the nearest neighbor. The projected radii r_x, r_y, r_z in screen-space of particle size r in world space can be determined as described in [Müller et al. 07]:

$$\begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} = \begin{pmatrix} rW\sqrt{p_{1,1}^2 + p_{1,2}^2 + p_{1,3}^2}/w \\ rH\sqrt{p_{2,1}^2 + p_{2,2}^2 + p_{2,3}^2}/w \\ r \end{pmatrix}, \tag{2}$$

where $p_{i,j}$ are the indices of the projection matrix $\mathbf{P}$. Thus, point splats are scaled according to their distance to camera position. If the aspect ratio of the projection is chosen by W/H, no distortion occurs and the particles appear as circles in screen-space ($r_p = r_x = r_y$). Afterwards, an empirical adaption depending on the specific data can increase details or can be used to model the liquids appearance. Rendering all particles according to these transformations into the depth buffer result in $\mathbf{Z}$.

2.2. Screen-Space Surface Smoothing

In order to obtain a smooth surface, we apply a low pass filter to all depth values $z_{i,j}$ (Figure 1, step 3). We experimented with different separable filters and decided to use a binomial filter, which delivers good results and can be implemented efficiently.

We estimate the initial kernel size k by setting it equal to the point splat radius r_p. Based on this estimate, the user can vary the kernel size to control the appearance of the liquid. If the kernel is too small, holes in the surface may appear, but if it is large, "splashes" of liquid separated from the surface are suppressed.

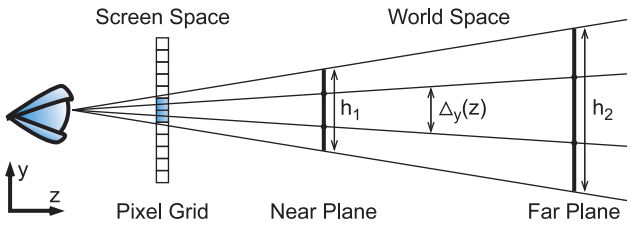


Figure 2. Calculation of pixel grid spacing in screen coordinates.

2.3. Normal Extraction

We extract a normal map $\mathbf{N} \in \mathbb{R}^{W \times H}$ by determining the partial derivations of the smoothed depth map. If we interpret the smoothed depth map as height field, its screen-space coordinates x_p, y_p, z_p sample surface coordinates in world space. Thus, the distance of surface point samples in world space depends on the camera position. In other words, neighboring elements (i. e., pixels) in the smoothed depth map can have different distance in screen-space (e.g., parallax), which is important in calculating the gradient.

Consider, for example, two objects in world space, one close to the near clipping plane and the other close to the far clipping plane. Due to perspective foreshortening, we obtain different depth values for two neighboring pixels in screen-space corresponding to both objects, respectively: neighboring pixels in screen-space corresponding to the object closer to the near plane are significantly closer to each other in world space than the two pixels corresponding to the object closer to the far plane. Therefore, we determine the actual distance of neighboring pixels at distance z by scaling the projected $x - y$-pixel-grid distance at the near plane w_1, h_1 linearly to the grid distance at the far plane w_2, h_2 (see Figure 2):

$$\triangle_x(z) = w_1 + (w_2 - w_1)z \quad (3)$$

$$\triangle_y(z) = h_1 + (h_2 - h_1)z. \quad (4)$$

Using these values, the partial derivatives of the smoothed depth map can be discretized in a straight forward fashion by forward differencing. Hence, $\mathbf{N}$ can be constructed as the normalized gradient.

Within the described method above, we assume a particle cloud without given normals. However, if the normals are given, the normals could be splat directly, to save the depth map creation process.

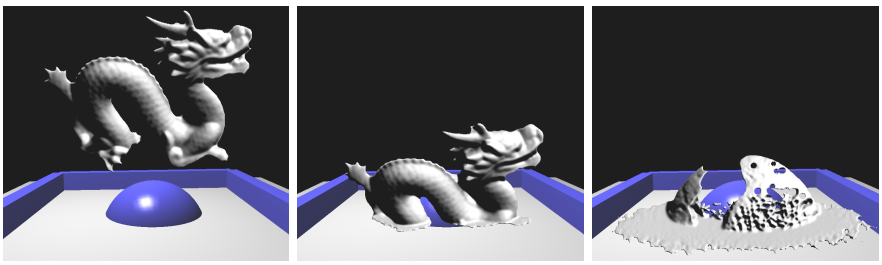


Figure 3. Dragon with lighting only at time steps t_i $(i = 0, 5, 10)$.

2.4. Lighting

All lighting calculations are carried out in screen-space. First, we obtain the light and camera position in screen-space, $\mathbf{l}_p$ and $\mathbf{c}_p$, according to Equation 1. This enables us to calculate the light direction $\mathbf{l}$ and view direction $\mathbf{v}$ for any entry in $\mathbf{Z}$. Then, we evaluate the standard Phong lighting model in screen-space (Figure 3).

Note that, due to smoothing the depth map $\mathbf{Z}$, sharp edges, such as those of liquid contours, are smoothed out and, thus, are shaded incorrectly. We will address this problem in Section 3. Furthermore, we would like to point out that unrealistic lighting effects on smoothed contours can occur, if the liquid lies between camera and light positions. Since we generate the normal map in screen-space, all normals point towards the image plane and, thus, back face normals cannot be represented in a smoothed depth map in general.

3. Instant Liquids – Extensions

In the following, we describe extensions to the basic algorithm to increase visual realism of the rendered liquid. Due to the image-based approach, well-known image filtering operations can be used to increase the quality of the results. For example, we employ additional color maps to map material properties onto the surface as depicted in Figure 6.

Reflection & Refraction Liquids such as water are highly transparent. Therefore, simulating view-dependent optical effects such as reflection and refraction is very important to increase visual realism. One can observe that reflection and refraction effects of real dynamic liquids are so complex that even approximations of these effects are adequate for believable animated sequences. This motivated us to devise an image-based approximation of refraction, since accurate refractions of liquids in complex scenes are difficult to achieve at high frame rates. This allows us to add plausible refraction effects

of liquid surfaces including intersection objects, even close to the viewer. To avoid multi-pass rendering, our technique is restricted to single refraction. This is a valid assumption if liquids are contained in opaque tanks. Thus, light rays refracted once at the liquid surface would not leave the tank (except in very "rough" surface conditions).

Instead of adhering to the laws of physics, reflection and refraction can be approximated by perturbing surface normals [Sousa 05]. We go one step further and determine physical reflection and refraction vectors, which are used to access an environment map. The reflection vector $\mathbf{r}$ is described by the law of reflection

$$\mathbf{r} = \mathbf{v} - 2\,\langle \mathbf{v}, \mathbf{n} \rangle\,\mathbf{n}, \tag{5}$$

and the refraction vector $\mathbf{t}$ is described by the Snell's Law with respect to the refraction ratio of refraction indices $\eta = \frac{n_1}{n_2}$:

$$k = 1 - \eta^2(1 - \langle \mathbf{v}, \mathbf{n} \rangle^2). \tag{6}$$

$$\mathbf{t} = \begin{cases} \eta\mathbf{v} - (\eta\langle \mathbf{v}, \mathbf{n} \rangle + \sqrt{k})\,\mathbf{n} & : \quad k \geq 0 \\ 0 & : \quad k < 0 \end{cases}. \tag{7}$$

Here, $\mathbf{v}$ is the view direction and $\mathbf{n}$ is the surface normal in screen-space. $\langle \mathbf{v}, \mathbf{n} \rangle$ is the scalar product of vectors $\mathbf{v}$ and $\mathbf{n}$. Thus, we can determine the reflection and refraction vectors in screen-space.

We approximate the refraction effect by accessing a refraction map $\mathbf{R}$, which represents the complete actual scene rendered from the camera position – without liquid. This map is accessed by a variation $\mathbf{d} = (x, y)$ of the x and y component of the refraction vector $\mathbf{t}$ in screen-space:

$$\mathbf{d} = c_{\text{int}} \cdot \begin{pmatrix} t_x \\ t_y \end{pmatrix}. \tag{8}$$

Due to the screen-space approach, we can neglect the z-coordinate t_z of $\mathbf{t} = (t_x, t_y, t_z)$. The intensity of refraction effect is controlled by parameter c_{int}.

We employ a standard cube mapping technique to approximate reflections. Liquid surrounding scene is rendered into the cube map from the view of the liquid's center of mass. The cube map generated that way is simply accessed by the reflection vector $\mathbf{r}$. However, this approach cannot reflect objects close to or intersecting the water surface realistically. Additionally, we would like to point out that liquid volumes in a scene comprising a surrounding cube that is at infinite distance can be reflected and refracted realistically using cube mapping.

The composition of reflection and refraction intensities is described by the Fresnel equations and leads, with respect to refraction indices n_1, n_2 and normalized vectors of view direction $\mathbf{v}$, normal $\mathbf{n}$ and refraction vector $\mathbf{t}$, to the reflected and refracted intensities I_{Refr} and I_{Refl} with $I_{Refr} = 1 - I_{Refl}$.

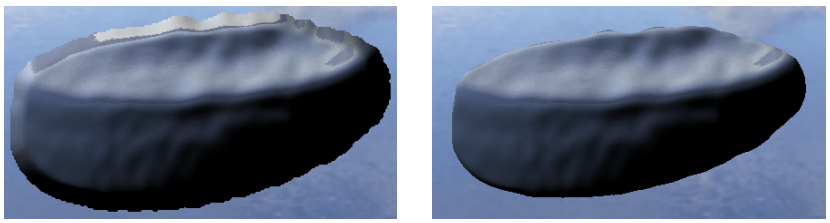


Figure 4. Illustration of silhouette artifacts reduction (Glass). Base (left), using silhouette reduction (right).

Silhouette Artifacts Reduction As described earlier, smoothing the depth map can lead to some unwanted artifacts. The drop of depth map values from maximum depth to liquid depth results in high normal variations, resulting in a silhouette surrounding the liquid. This side effect can be reduced by silhouette detection. Therefore, the smoothed depth map is modified such that every depth value $z_p \neq 1$ is set to 0. Hence, a binary image of the depth map is generated, which is smoothed by the same kernel as described in Section 2.2, resulting in a silhouette map. According to this silhouette map, the liquid is just rendered at fragment positions with values equal to one, otherwise the background is rendered. Thus, the unrealistic visual effects within the silhouette are reduced (Figure 4).

4. Examples

We use several real-time 3D-particle simulations with up to 140,000 particles to analyze the performance of *Instant Liquids*. Take note, that some simulations do not utilize a Navier-Stokes Equation solver. Thus, the simulated behavior does not correspond to real liquids in these scenarios. Since a real-time, physically based simulation of such an amount of particles has not been presented so far, we utilize among others simple integration schemes to analyze our fast rendering method in real-time environments with many particles. In the following we would like to describe these simulations. Quantitative performance results are discussed in Section 5.

Glass, Cube We use a standard interactive SPH simulation with 2,000 simulated particles (Figure 4, cube: See accompanying video.).

Breaking Waves The complex and fascinating physical behavior of breaking waves has attracted the interest of researchers in the field of computer graphics for quite a while. However, physics-based simulations including rendering at interactive rates have not been presented yet. Recently, [Thuerey

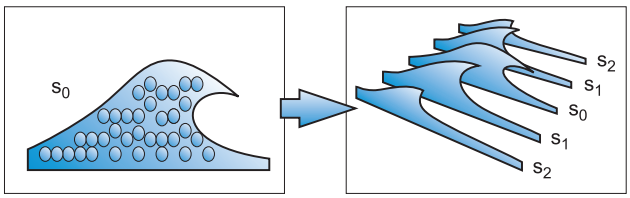


Figure 5. Principle of breaking waves simulation. A 2D SPH simulation is fulfilled (left). The last n timesteps (here: n=2) of simulation data is aligned slice based (right). Thus, the symmetry if a breaking wave can be approximated.

et al. 07] has pushed the limit with an animation-based breaking waves approach coupled to a shallow water simulation. Our simulation method for breaking waves uses a sliced-based approach and in combination with our rendering method, real-time results can be achieved.

A 2D SPH simulation is the basis for our breaking waves simulation (140,000 particles, 200 slices). The idea is, to construct a 3D breaking wave from a 2D simulation. A breaking wave in reality is strongly self-similar. The self-similarity is used, to construct the breaking wave. Therefore, a 2D breaking wave is generated by the use of an external force field within a 2D simulation – the result is shown in Figure 5a. The last n timesteps of all particle positions within the 2D simulation are stored ($p(t_j), j = -n \dots 0$) and are used later to construct the wave.

The front shape of breaking waves is defined by slices s_i along the z-axis (i index of z-values, Figure 5b). The slices are constructed by the use of the last n 2D particle sets $p(t_j)$. The mapping of point sets to slices is described by the time-spacing function $f(i) \in \{-n, \dots, 0\}$: $s_i = p(t_{f(i)})$. Hence, the function $f(i)$ describes the shape of the breaking wave. E.g., $f(i) = 0$ describes a straight font and $f(i) = -i$ describes a spiky front. In our example (Figure 6), $f(i)$ is described by a superposition of wave functions. In principle, any wave shape can be modeled in a similar way. The benefit of this approach lies in the reduced amount of simulation time. Only a fast 2D SPH-simulation

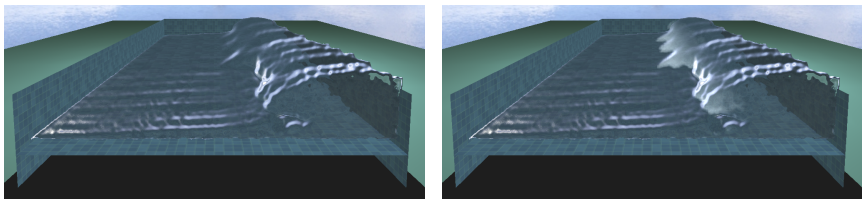
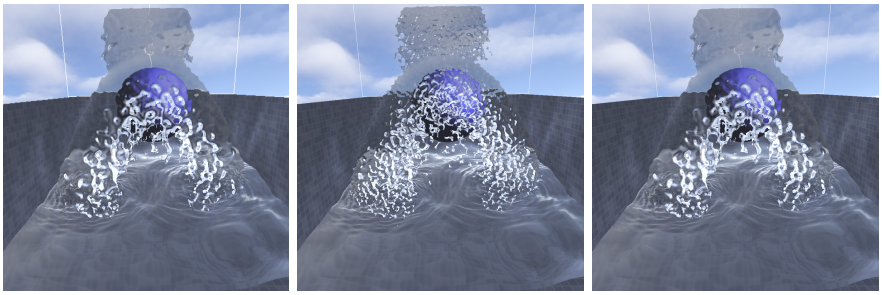


Figure 6. Breaking Waves. Basic approach (left), using an additional velocity map, to represent spray (right). The velocity map is defined by particle velocities (here: along the y-axis) at a given position.



(a) (b) (c)

Figure 7. Waterfall. The same scene is rendered with different screen-space resolutions: 512×512 (a), 1024×1024 (b), Scene: 1024×1024 – Liquid: 512×512 (c).

with few particles has to be fulfilled, whereas the 3D wave is constructed by the use of recent slice data.

To enhance visualization of breaking waves (Figure 6), we use an additional velocity map, to represent spray. Particle velocities are projected into screen-space as described in Section 2.1 for particle positions – resulting in the velocity map. Within the rendering pass, the liquids color is shifted to white according to velocity information. In Figure 6b, the velocity in y-direction is used for shifting.

Waterfall The waterfall is realized by utilizing mass-point systems. The frequent use of mass-point systems for approximating liquid simulations is based on the assumption that the interaction between falling liquid particles, splashing particles or drops is negligible. The only acting force is gravity and no liquid specific forces have to be calculated. Hence, the integration over time is trivial, effective and very fast. Thus, more particles can be used in fountains, splashes or falling drops (compared to full physics-based simulation), allowing a more plausible real-time simulation. The waterfall (60,000-100,000 particles, Figures 1 and 7) is approximated by a randomized particle creation process, where particles are created randomly within a given volume distributed along a main velocity vector.

Dragon The dragon (64,474 particles, Figure 8) is created by positioning the particles according to the given vertexes. The simulation is realized by utilizing a mass-point simulation as well.

Pool To provide enhanced visualization of mass-point simulations, we use the mode-splitting technique [Cords 07], to simulate the liquid surface in the pool examples realistically (Figures 7 and 8). The idea is, to couple a 2D

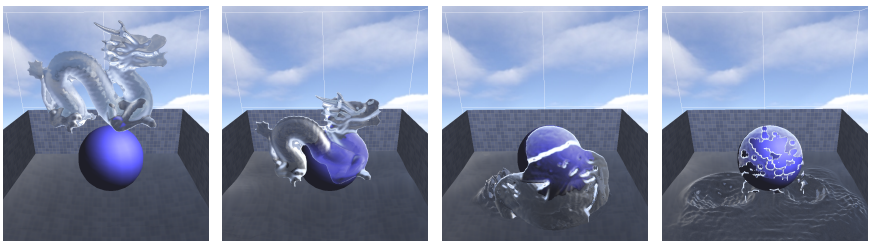


Figure 8. Liquid dragon in pool (consisting of 64,474 particles) is falling at interactive rates (timesteps t_i with $i = 0, 10, 20, 30$).

wave equation solver with a 3D SPH simulation and utilize a height field based rendering approach. Thus, detailed surface waves can be created (e.g., by falling particles).

5. Discussion

The presented experiments were performed on a dual-core 2,6 GHz AMD Athlon 64 CPU, 2GB of RAM and an ATI Radeon x1900 GPU (512MB). Since we use the GPU for rendering and would like to demonstrate our method on a typical personal computer, we have to use a CPU-based implementation for all presented simulations. By using an additional graphics card, GPU-based simulation methods can be used to improve overall system performance.

The performance measures of our implementation, using just one core for the simulated examples, are shown in Table 1. We present the results for screen-space resolutions of 512×512 and 1024×1024. All measured frame rates lie between 30-60 FPS and allow full interactivity (see accompanying video). Note that most of the time is spent for simulation, not for rendering with Instant Liquids. Due to the image-based approach, the complexity and, thus, the performance directly depends on the number of rendered fragments. It may be practical, to create the normal map (representing the liquid) at reduced resolution and upsample it bilinearly. Afterwards, the reflection/refraction step is executed according to the full resolution. Results for this special case are also included in Table 1. Thus, a very good performance/quality tradeoff is achieved (Figure 7) – all other images within this paper were generated using this technique. Due to the use of an image-space approach, the resulting object topology depends on the resolution. For screen-space resolutions of 1024×1024 performance decreases, due to the intense use of the fragment shader.

In addition to the Instant Liquids algorithm we employ the mode-splitting method (see Section 4, Figures 7, 8), resulting in frame rates of approximately

	sim	512			1024			512/1024		
	[ms]	**Z** [ms]	IL [ms]	[fps]	**Z** [ms]	IL [ms]	[fps]	**Z** [ms]	IL [ms]	[fps]
Dragon	15.0	1.9	0.13	58.7	3.3	2.8	47.3	1.9	0.13	58.7
Waterfall	22.1	2.9	0.53	39.1	3.1	2.6	35.9	2.9	0.61	38.7
Waves	9.3	9.0	0.43	53.2	12.0	2.7	41.5	9.0	0.46	53.1
Glass	13.5	3.5	0.12	58.4	5.1	3.1	46.0	3.5	0.26	57.9
Cube	13.3	5.3	0.57	52.9	9.0	3.8	38.7	5.3	0.73	52.5

Table 1. Performance measurements of the *Instant Liquids* method in different scenarios. Shown are the times needed for simulation (sim), scene and point rendering (e.g., creating the depth map **Z**) and the additional *Instant Liquids* method (IL) – at different resolutions.

25 FPS using mainly one single core. Only the height field surface extraction is fulfilled on the second core. *Instant Liquids* run times are similar to the scenes without pool (Table 1).

The complexity of the technique scales linearly. Within the creation of the depth map the complexity depends on the number of particles n: $O(n)$. In the following steps, the complexity depends on the number of processed fragments: $O(W \times H)$. Hence, processing time increases linearly according to the number of particles and according to the screen resolution.

In the remainder of this section we discuss some of the limitations of our method. For static objects (e.g., a stopped animation), blobby artifacts can occur while rotating the object as depicted in Figure 9.

As mentioned before, the smoothing step described in Section 2.2 can lead to visual artifacts at the silhouette. The method introduced in Section 3 effectively reduces these artifacts at the liquid's outer silhouette for convex shapes. In the case of concave shapes or multiple liquids, however, those artifacts can even occur within the liquid's contour. In the case of multiple liquid volumes, these artifacts can be alleviated following a multipass rendering approach: Each liquid volume is rendered separately and, thus, the overlapping contours are not smoothed.

In the case of concave surface shapes, however, it is not as easy to solve this problem without significant performance hits. In our method, smoothing is an essential step for generating the surface. A more sophisticated edge-preserving filter could be used to solve the problem for concave shapes. Standard edge preserving smoothing filters (e.g., median or bilateral filters) cannot handle the problem, since the depth map mainly consists of discrete depth values. More complex edge preserving reconstruction algorithms (e.g., [Gijbels et al. 06]) can handle those situations, but these approaches are too complex to operate in real time. Even with our simple separable binomial fil-

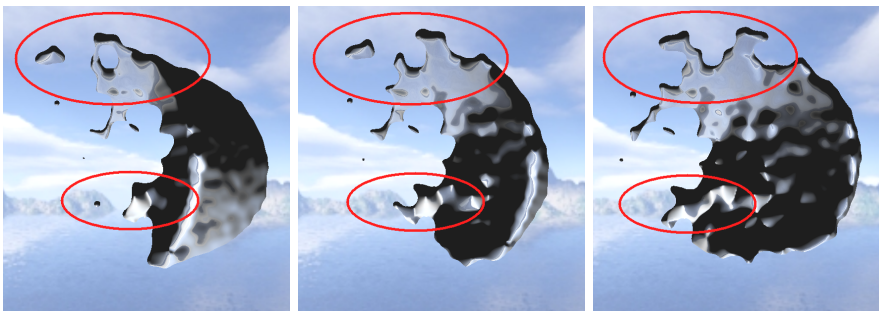


Figure 9. Artifacts appear while rendering static objects. A static liquid object within a turbulent situation is rotated slightly (along y-axis). Thereby, the shape changes slightly and smooth. This example utilizes just 2,000 particles – the side effect is reduced for significantly more particles, since splat radius and smoothing kernel size can be reduced.

ter, smoothing is the performance bottleneck of Instant Liquids. In practice, smoothing artifacts within concave liquid volumes occur mostly in turbulent situations. In fast animations, the resulting visual artifacts can be seen (e.g., Figure 9), but are acceptable for real-time environments where we have to trade off accuracy for high performance and interactivity.

Furthermore, the use of single refraction can lead to implausible results in the case of breaking waves, since situations occur where double refraction would occur in reality. In principle, Instant Liquids could be expanded to rendering of front- and back-face liquids. Therefore, the normal map is generated within a two-pass approach as a front and back view of the liquid. Subsequently, a real-time rendering approach for transparent objects with two interfaces can be adapted (e.g., [Wyman 05]).

Another problem within the field of particle-based surface reconstruction is the visual loss of volume in situations where particle density decreases strongly (e.g., particles separate within simulation). The use of implicit functions (or, in our case, the smoothing step) cannot produce a closed surface for those particles. This undesirable effect could be reduced by increasing the splat radiuses in those situations (e.g., depending on local density).

Compared to the Screen Space Meshes approach [Müller et al. 07], our method can visualize much more particles at same frame rates. Mller et al report frame rates for different scenarios with and without the use of Screen Space Meshes. Among others, they report 65 FPS for the SPH-simulation of 5000 particles. Including the Screen Space Meshes approach the frame rate drops to 55 FPS with 5000 triangles and to 40 FPS with 20,000 triangles. For environments with up to 16,000 particles, they report frame rates less than 23 FPS. The Instant Liquids method renders more than 100,000 particles with comparable or better frame rates (Table 1), running entirely on the GPU

and thus, relieving the CPU whereas the Screen Space Meshes method is performed on the CPU and GPU.

The Instant Liquids method outperforms the particle splatting method as well [Adams et al. 06]. This method uses an additive alpha blending for interpolating normals in screen space. They report frame rates of round about 10 FPS for environments with approximate 100,000 particles and 100 FPS for environments with approximate 10,000 particles. These performances are measured without an underlying simulation – only the rendering performance is given in FPS. Our wave and waterfall examples simulate and render 100,000 or more particles with 38 and 53 FPS. Beside their use of a GeForce 7800 graphics card, the performance gain of Instant Liquids result from the expensive two-pass rendering approach and the computationally expensive blending steps used in [Adams et al. 06]. Additionally, the particle splatting method result in unrealistic silhouettes, since individual particles are still visible. Moreover, our examples seem to be more detailed and our reflection and refraction method seems to be more realistically as well.

Hence, the Instant Liquids method described in this work significantly improve performance of surface extraction of particle clouds in interactive environments. In principle, our technique can be used for screen-space lighting of *polygonal* objects without given normals as well. Therefore, the depth map is generated as the typical depth map of the polygonal object and the smoothing step can be disregarded. The depth map is given to the normal generation step and can be directly rendered afterwards as described in Section 2.4.

References

[Adams et al. 06] Bart Adams, Toon Lenaerts, and Philip Dutre. "Particle Splatting: Interactive Rendering of Particle-Based Simulation Data." *Technical Report CW 453, Katholieke Universiteit Leuven.*

[Cords 07] H. Cords. "Mode-Splitting for Highly Detailed, Interactive Liquid Simulation." *Proceedings of Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (Graphite)*, pp. 265–272.

[Gijbels et al. 06] Irene Gijbels, Alexandre Lambert, and Peihua Qiu. "Edge-Preserving Image Denoising and Estimation of Discontinuous Surfaces." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28:7 (2006), 1075–1087.

[Lorensen and Cline 87] W. E. Lorensen and H. E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm." In *Computer Graphics (Proceedings of SIGGRAPH 87*, 21, 21, pp. 163–169. ACM, 1987.

[Müller et al. 03] M. Müller, D. Charypar, and M. Gross. "Particle-based fluid simulation for interactive applications." In *Proceedings of Symposium on Computer animation*, pp. 154–159. Eurographics Association, 2003.

[Müller et al. 07] M. Müller, S. Schirm, and S. Duthaler. "Screen space meshes." In *Proceedings of Symposium on Computer animation*, pp. 9–15. Eurographics Association, 2007.

[Sousa 05] T. Sousa. "Generic Refraction Simulation." In *GPU Gems 2*, pp. 295–305. Addison-Wesley, 2005.

[Stam 99] J. Stam. "Stable Fluids." In *Proceedings of ACM SIGGRAPH 1999, Computer Graphics Proceedings*, pp. 121–128. ACM Press/ACM SIGGRAPH, 1999.

[Tessendorf 04] J. Tessendorf. "Simulating Ocean Water." In *Course Notes Siggraph: The Elements of Nature: Interactive and Realistic Techniques (Course 31)*, 2004.

[Thuerey et al. 07] N. Thuerey, M. Müller, S. Schirm, and M. Gross. "Real-time Breaking Waves for Shallow Water Simulation." In *Proceedings of Pacific Graphics*, 2007.

[Wyman 05] Chris Wyman. "An approximate image-space approach for interactive refraction." *ACM Transactions on Graphics* 24:3 (2005), 1050–1053.

Web Information:

Accompanying video (DivX):
http://www.informatik.uni-rostock.de/∼hc009/videos/instantliquids.avi

Hilko Cords, Oliver Staadt
Institute of Computer Science
Visual Computing Group
Albert-Einstein-Str. 21
18051 Rostock, Germany
{hilko.cords, oliver.staadt}@uni-rostock.de

Received [DATE]; accepted [DATE].