Real-Time Screen-Space Liquid Rendering with Two-Sided Refractions

Takuya Imai **Yoshihiro Kanamori** **Yukio Fukui** **Jun Mitani**

University of Tsukuba

imai-t@npal.cs.tsukuba.ac.jp, {kanamori, fukui, mitani}@cs.tsukuba.ac.jp

Abstract
In interactive applications such as games, particle-based liquids are often used for simulation. Such liquids are rendered with only a single refraction at the front-facing surface, which damages photorealism because it cannot express the thickness of liquid. In this paper, we propose an approach to render particle-based liquid with twice refraction at front- and back-facing surfaces in real time. We use two existing approaches, i.e., one is to generate the liquid surface from particles in real time, and the other is to render a polygon mesh with handling two-sided refraction in real time. We further accelerate the intersection calculation of the back-facing surface and viewing rays by the secant method.

1 Introduction

In interactive applications such as games, liquid simulation is performed by discretizing liquid as a finite set of particles and by calculating the motion of particles. Results of rendering such particle-based liquids in real time are, however, not sufficiently photorealistic because light reflection and refraction are approximated and calculated only once when each ray enters the liquid, which damages photorealism because it cannot express the thickness of liquid.

In this paper, we propose an approach to render particle-based liquid with two-sided refractions, considering twice refractions at front- and back-facing surfaces in real time (see Figure 1). For twice refractions, there exist real-time rendering methods for polygon meshes [1, 2], which require front- and back-facing depth and normal maps. We thus obtain depth and normal maps of particle-based liquid by extending a screen-space method [3] and then calculate twice refractions. To reduce rendering passes required for front- and back-facing depth maps, we obtain them simultaneously using *Dual Depth Peeling* [4]. We further accelerate the intersection calculation of the back-facing surface and viewing rays by the secant method instead of binary search used in [2]. We demonstrate the effectiveness of our method through comparisons with two results where only a single refraction is considered or intersections are calculated using binary search.

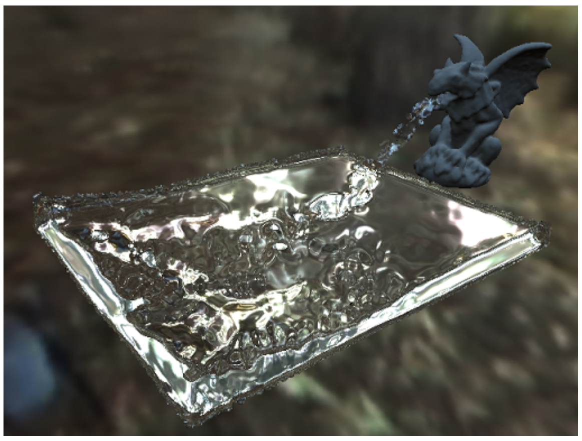


Figure 1: Our result of liquid rendered with twice refractions. 8,000 liquid particles are rendered at 85.9 fps at resolution of 640×480.

2 Related Work

Particle-based liquid is often rendered as metaballs. Kanamori et al. [5] extracted metaball surfaces using *Bézier*

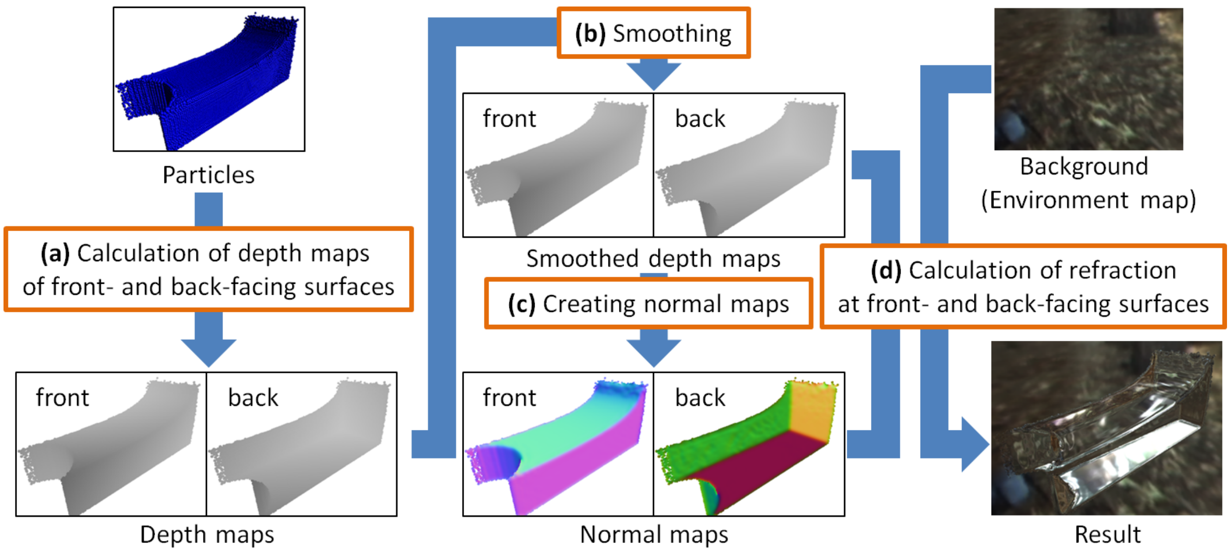

Particles
(a) Calculation of depth maps of front- and back-facing surfaces
front
back
Depth maps
(b) Smoothing
front
back
Smoothed depth maps
(c) Creating normal maps
front
back
Normal maps
Background (Environment map)
(d) Calculation of refraction at front- and back-facing surfaces
Result

Figure 2: Overview of our method. (a) First, depth maps of front- and back-facing surfaces are created by rendering particles as spheres. (b) Second, depth maps for both front- and back-facing surfaces are smoothed simultaneously by applying the bilateral filter iteratively. (c) Third, normal maps are created from the smoothed depth maps. (d) Finally, twice refractions at front- and back-facing surfaces are calculated to obtain the final result.

Clipping and *Depth Peeling*, which handled only a single refraction. Gourmel et al. [6] accelerated ray tracing of metaballs using *Bounding Volume Hierarchy* (BVH). Yu and Turk [7] introduced anisotropic kernels to reduce the bumpiness of metaball surfaces. Although these approaches can generate more photorealistic results, they are too expensive to render liquid in real time.

Müller et al. [8] extracted liquid surfaces as screen-space polygon meshes from particles in real time. Cords and Staadt [3] generated a surface by smoothing a depth map obtained after rendering particles as spheres. van der Laan el al. [9] also smoothed depth maps using *Mean Curvature Flow*, which minimizes mean curvature of liquid surface. They further calculated the pseudo thickness to imitate the thickness of liquid. These approaches target only a single refraction at front-facing surfaces without handling twice refractions. In this paper we present an approach handling twice refractions at front- and back-facing surfaces.

Wyman [1] showed that just twice refractions at front- and back-facing surfaces make results sufficiently photorealistic for polygon meshes. However, Wyman's approach requires precalculation of the distance from each vertex to an exiting intersection calculated along the normal direction, which is not applicable to deformable objects. Wyman's approach is further extended by calculating the ray-intersection at back-facing surfaces using binary search for handling deformable objects [2] and by calculating total internal refraction [10]. These approaches target polygon meshes. If we render particle-based liquids using these approaches, we require depth maps of liquid surfaces. In this paper, we achieve rendering of particle-based liquid with twice refractions in real time using a screen-space method of generating liquid surfaces.

3 Our Method

3.1 Outline

The input data are the position and the radius of each particle. We extend the method by Cords and Staadt [3] to obtain liquid surfaces. We then extend the method by Oliveira and Brauwers [2] to calculate twice refractions with obtained depth maps of front- and back-facing surfaces of liquid. Our method approximately determines pixel colors by linearly blending the colors of light reflected at the front-facing surface and light refracted at the back-facing surface. Figure 2 shows the overview of our method.

We improve the rendering performance as follows. Creating depth maps of front- and back-facing surfaces re-

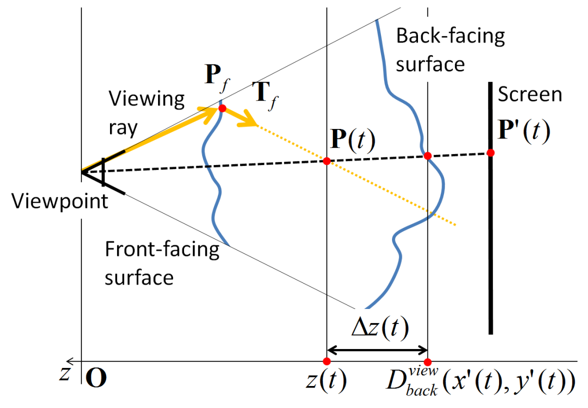


Figure 3: Illustration of the secant method.

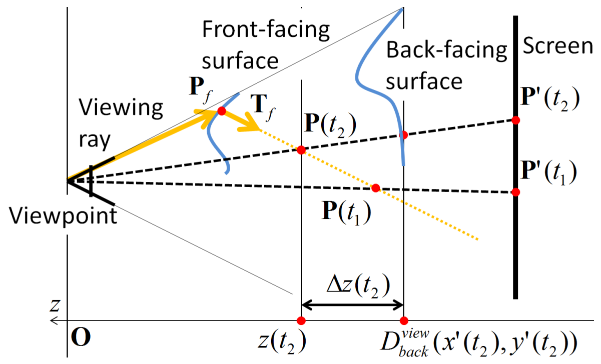


Figure 4: Solution in case that an invalid depth value is referenced. At $t = t_1$, the secant method cannot be applied because $D^{view}_{back}(x'(t_1), y'(t_1))$ is invalid. We halve t iteratively until a valid depth value is obtained.

quires rendering a large number of spheres twice, which is computationally expensive. To reduce the rendering passes, we obtain the two depth maps simultaneously using *Dual Depth Peeling* [4]. For smoothing the depth maps, instead of the binomial filter used in [3], we apply the bilateral filter to preserve depth edges. We use a filter kernel of small radius and apply the filter iteratively to reduce computational cost. Both depth maps of front- and back-facing surfaces are smoothed simultaneously to reduce rendering passes.

In [2], intersections of refracted rays and the back-facing surface are calculated using binary search, which requires additional calculation of minimum and maximum values for the back-facing depth map to limit the search range. Instead we use the secant method to reduce the number of iterations of ray intersection tests and to omit min/max calculation of the depth map.

3.2 Surface Generation

We obtain liquid surfaces by rendering particles as spheres and by smoothing the resultant depth maps, similarly to Cords and Staadt [3]. We use an optimized method for rendering spheres, proposed by Kanamori et al. [5]. For smoothing the depth maps, we apply the bilateral filter iteratively, instead of the binomial filter used in [3]. Let D be an input depth map before smoothing. The depth value at pixel (i, j) in the filtered depth map D' is then calculated as follows:

$$D'(i,j) = \frac{\sum_{n=-w}^{w} \sum_{m=-w}^{w} D(i+m, j+n) W_1(m,n) W_2(m,n,i,j)}{\sum_{n=-w}^{w} \sum_{m=-w}^{w} W_1(m,n) W_2(m,n,i,j)}, \tag{1}$$

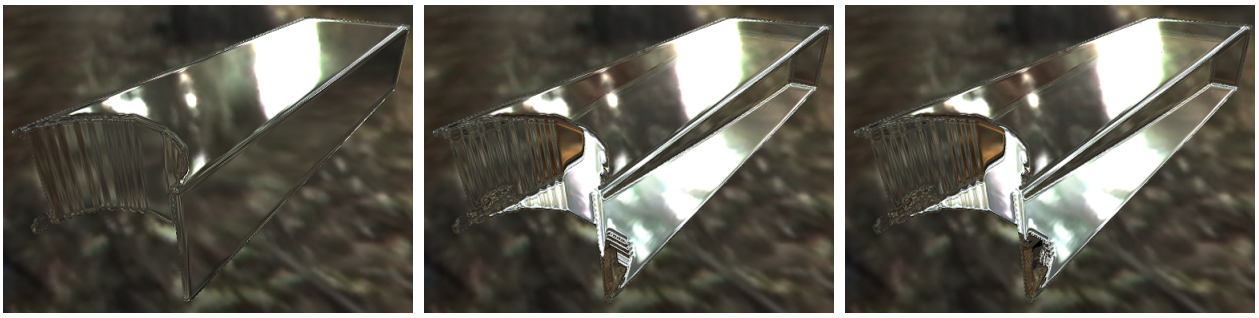
$$W_1(m,n) = \exp\left(-\frac{m^2+n^2}{2\sigma_1^2}\right), \tag{2}$$

$$W_2(m,n,i,j) = \exp\left(-\frac{(D(i,j) - D(i+m, j+n))^2}{2\sigma_2^2}\right), \tag{3}$$

where w is the kernel radius, and σ_1 and σ_2 are the constants of the bilateral filter. In our results, we set $\sigma_1 = \frac{w}{2\sqrt{2}}$ and $\sigma_2 = 2\sqrt{2}r$, where r is the particle radius. We empirically determined $w = 4$ and the number of iterations for filtering as 20. Finally, normal maps are calculated from the smoothed depth maps, as described in [9].

3.3 Intersection Calculation

For calculating the refraction at the back-facing surface, we need to calculate the intersection of viewing rays and back-facing surface. For this, Oliveira and Brauwers applied the binary search, which requires two initial points to restrict the search range; one is nearer and another is farther than the back-facing surface. Their method thus additionally calculates the minimum and maximum depth values of the back-facing depth map to limit the search range. Instead we use the secant method



(a) Result with single refraction only (b) Result using binary search (c) Our result

Figure 5: Comparison of different rendering methods.

so that we can calculate intersections with fewer iterations without additional calculation of min/max depth values. The secant method also requires two points for initialization but, unlike binary search, they do not have to be points with min/max depth values.

We calculate intersections at the back-facing surface as follows. Let $\mathbf{P}_f$ and $\mathbf{T}_f$ be the intersection point and the unit directional vector of a viewing ray refracted at the front-facing surface, respectively. Point $\mathbf{P}(t)$ along the refracted ray is then expressed with a ray parameter $t(\geq 0)$ as follows:

$$\mathbf{P}(t) = \mathbf{P}_f + t\,\mathbf{T}_f. \tag{4}$$

Let $\mathbf{P}'(t) = (x'(t), y'(t), z'(t))$ be the point obtained by projecting $\mathbf{P}(t) = (x(t), y(t), z(t))$ onto the screen. Also let D^{view}_{back} be a depth map that records depth values in the viewing coordinates for the back-facing surface. $D^{view}_{back}(x'(t), y'(t))$ represents the depth value at $(x'(t), y'(t))$ in the screen coordinates. The difference between the z component of the point $\mathbf{P}(t)$ and $D^{view}_{back}(x'(t), y'(t))$ is then calculated as follows (see Figure 3):

$$\Delta z(t) = z(t) - D^{view}_{back}(x'(t), y'(t)). \tag{5}$$

We define a linear function $f(t)$ that passes through $(t_s, \Delta z(t_s))$ and $(t_e, \Delta z(t_e))$, where $\Delta z(t_s)$ and $\Delta z(t_e)$ represent the z-component differences for two points $\mathbf{P}(t_s)$ and $\mathbf{P}(t_e)$ $(t_s < t_e)$ along the refracted ray,

$$f(t) = \frac{\Delta z(t_e) - \Delta z(t_s)}{t_e - t_s}(t - t_s) + \Delta z(t_s). \tag{6}$$

Ray parameter t^* that yields $f(t^*) = 0$ is calculated as

$$t^* = t_s - \frac{t_e - t_s}{\Delta z(t_e) - \Delta z(t_s)}\Delta z(t_s). \tag{7}$$

At $t = t^*$, $\Delta z(t^*)$ represents the difference between $\mathbf{P}(t^*)$ and the back-facing surface. If $|\Delta z(t^*)|$ is below a certain threshold ε, $\mathbf{P}(t^*)$ is considered as the intersection of the refracted ray and the back-facing surface. Otherwise, either $\mathbf{P}(t_s)$ or $\mathbf{P}(t_e)$ is replaced with $\mathbf{P}(t^*)$ to satisfy the condition $t_s < t_e$, and then the procedure is repeated until convergence. At the begining, we initialize $t_s = 0$ and t_e as

$$t_e = (Z_{far} - P_f.z)/T_f.z, \tag{8}$$

where Z_{far} is z coordinate of the far plane of the viewing frustum, $P_f.z$ and $T_f.z$ are z components of $\mathbf{P}_f$ and $\mathbf{T}_f$.

In case that no particles are rendered at pixel $(x'(t), y'(t))$, the depth value referenced at that pixel is invalid and thus cannot be used as the initial value for the secant method. In this case we halve ray parameter t iteratively until a valid depth value is obtained at $(x'(t), y'(t))$ (see Figure 4).

3.4 Calculation of the Output Color

The output color $\mathbf{C}$ is calculated as follows:

$$\mathbf{C} = F(\mathbf{v}, \mathbf{n}_f)\mathbf{C}_f + (1 - F(\mathbf{v}, \mathbf{n}_f))\mathbf{C}_b. \tag{9}$$

where $\mathbf{v}$ is the unit directional vector of a viewing ray, $\mathbf{n}_f$ is the normal vector at the front-facing surface, $\mathbf{C}_f$ is the color fetched from the environment map along the directional vector reflected at the front-facing surface, $\mathbf{C}_b$ is the color fetched from the environment map along the directional vector refracted at the back-facing surface, and F is the Fresnel coefficient for which we employ Shlick's approximation [11].

4 Results

We implemented our method using C++ with OpenGL, GLUT, GLUI and GLEW. We wrote the shader codes

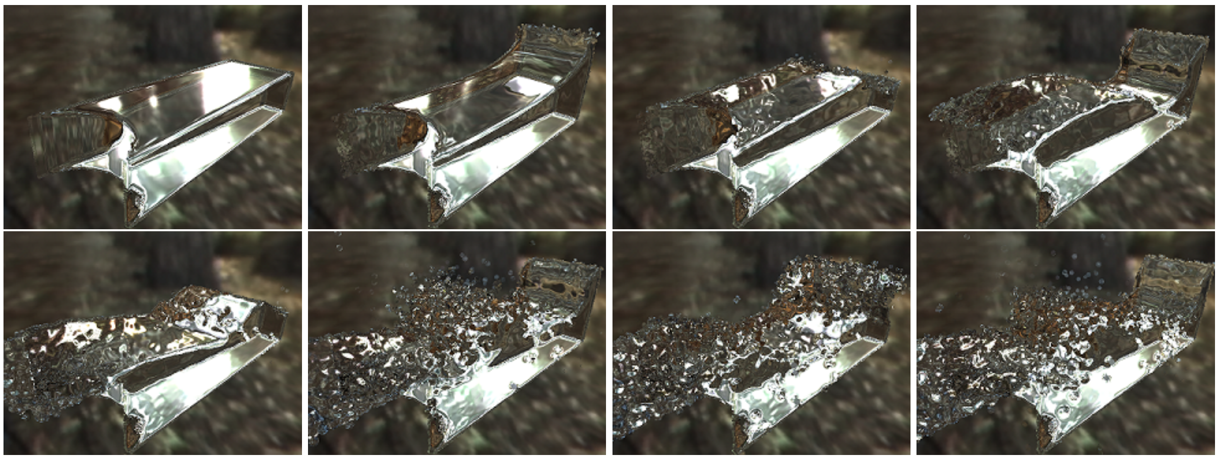


Figure 6: The results of an animating liquid. The liquid consists of about one hundred thousand particles, and are rendered at 79.1 fps at resolution of 640×480. Each image is obtained per 30 frames.

using GLSL. The experiments were conducted on a PC equipped with an Intel Core i7-4770 3.40GHz CPU, 8GB RAM, and an NVIDIA GeForce GTX TITAN GPU. The following results were rendered at resolutions of 640×480 and 1024×768 using pre-simulated liquid consisting of about one hundred thousand particles. The reported computational times do not include the time for simulation.

Figure 5 shows a comparison of the results using our method, binary search used for ray-surface intersection in [2], and with single refraction only. Also, Tables 1, 2 and 3 respectively represent the computational times required for rendering using our method, binary search, and with single refraction only. In each table, "*Depth map*" represents the construction of depth maps by rendering spheres, "*Smoothing*" the iterative smoothing by the bilateral filter, "*Normal map*" the construction of normal maps, "*Min/max comp.*" the calculation of min/max values of the back-facing depth map, and "*Refraction*" the calculation of refraction as well as the final colors. Tables 1 and 2 show that our method is faster than using binary search, while the resultant images are almost identical, as shown in Figures 5(b) and 5(c). On the other hand, Tables 1 and 3 show that we can render images about 1.5 times faster when we only consider single refraction. However, as shown in Figures 5(a) and 5(c), our result conveys the thickness of the liquid more faithfully.

Table 1: Computational times (msec) per frame using our method.

	640×480	1024×768
Total	12.64	30.78
Depth map	2.13	5.14
Smoothing	10.24	25.02
Normal map	0.10	0.25
Refraction	0.17	0.37

Table 2: Computational times (msec) per frame using binary search [2].

	640×480	1024×768
Total	12.72	30.88
Depth map	2.13	5.14
Smoothing	10.24	25.02
Normal map	0.10	0.25
Min/max comp.	0.12	0.20
Refraction	0.13	0.27

Table 3: Computational times (msec) per frame with single refraction only.

	640 × 480	1024 × 768
Total	8.78	19.83
Depth map	0.71	1.91
Smoothing	8.00	17.77
Normal map	0.03	0.06
Refraction	0.04	0.09

5 Conclusions and Future Work

In this paper, we have proposed a method for rendering particle-based liquids with handling twice refractions in real time. We extend and utilize the previous methods for screen-space extraction of liquid surfaces as well as calculation of twice refractions for polygon meshes. We enhance the rendering performance using *Dual Depth Peeling* [4] to reduce rendering passes, and using the secant method to reduce the cost for calculating intersections of refracted rays and back-facing surfaces.

Our method assumes that the space between front- and back-facing surfaces is fully filled with liquid, and thus yields unnatural refraction in case that splashes are flying apart in front of the liquid body, as shown in Figure 6. A solution for this problem is, for example, to calculate more accurate refraction with three or more depth maps by applying *Dual Depth Peeling* iteratively. We would like to develop an efficient way for rendering liquid with more complicated refractions in real time.

References

[1] Chris Wyman. An approximate image-space approach for interactive refraction. *ACM Trans. Graph.*, 24(3):1050–1053, 2005.

[2] Manuel M. Oliveira and Maicon Brauwers. Real-time refraction through deformable objects. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, pages 89–96, 2007.

[3] Hilko Cords and Oliver Staadt. Instant liquids. In *Poster proceedings of ACM Siggraph/Eurographics symposium on computer animation*, 2008.

[4] Louis Bavoil and Kevin Myers. Order independent transparency with dual depth peeling. Technical report, NVIDIA, 2008.

[5] Yoshihiro Kanamori, Zoltan Szego, and Tomoyuki Nishita. GPU-based fast ray casting for a large number of metaballs. *Computer Graphics Forum*, 27(2):351–360, 2008.

[6] Olivier Gourmel, Anthony Pajot, Mathias Paulin, Loïc Barthe, and Pierre Poulin. Fitted BVH for fast raytracing of metaballs. *Computer Graphics Forum*, 29(2):281–288, 2010.

[7] Jihun Yu and Greg Turk. Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Trans. Graph.*, 32(1):5:1–5:12, 2013.

[8] Matthias Müller, Simon Schirm, and Stephan Duthaler. Screen space meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '07, pages 9–15, 2007.

[9] Wladimir J. van der Laan, Simon Green, and Miguel Sainz. Screen space fluid rendering with curvature flow. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, pages 91–98, 2009.

[10] Scott T Davis and Chris Wyman. Interactive refractions with total internal reflection. In *Proceedings of Graphics Interface 2007*, pages 185–190, 2007.

[11] Christophe Schlick. An inexpensive BRDF model for physically-based rendering. *Computer Graphics Forum*, 13(3):233–246, 1994.