Particle Splatting: Interactive Rendering of Particle-Based Simulation Data

Bart Adams
Toon Lenaerts
Philip Dutré

Report CW 453, July 2006

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Particle Splatting: Interactive Rendering of Particle-Based Simulation Data

Bart Adams
Toon Lenaerts
Philip Dutré

Report CW 453, July 2006

Department of Computer Science, K.U.Leuven

Abstract

Particle-based simulation methods are gaining popularity for creating animations of physical phenomena such as fluids and melting solids. Extracting and visualizing an explicit surface corresponding to the volume of particles is however a difficult and time-consuming task, especially with increasing particle set sizes. In this paper, we present a novel interactive rendering algorithm for rasterizing a surface which conforms to the particle cloud. We show how projection and blending of overlapping spheres assigned to the particles yields smooth surfaces and discuss how this algorithm can be implemented on the GPU.

The presented rendering algorithm allows rapid high-quality surface visualization for particle-based simulations, without the need to extract an explicit surface representation. Therefore, it has applications for both preview rendering of particle simulations and real-time fluid visualization for interactive applications.

CR Subject Classification : I3.5, I3.7

Particle Splatting: Interactive Rendering of Particle-Based Simulation Data

Bart Adams, Toon Lenaerts, Philip Dutré

July 10, 2006

Abstract

Particle-based simulation methods are gaining popularity for creating animations of physical phenomena such as fluids and melting solids. Extracting and visualizing an explicit surface corresponding to the volume of particles is however a difficult and time-consuming task, especially with increasing particle set sizes. In this paper, we present a novel interactive rendering algorithm for rasterizing a surface which conforms to the particle cloud. We show how projection and blending of overlapping spheres assigned to the particles yields smooth surfaces and discuss how this algorithm can be implemented on the GPU.

The presented rendering algorithm allows rapid high-quality surface visualization for particle-based simulations, without the need to extract an explicit surface representation. Therefore, it has applications for both preview rendering of particle simulations and real-time fluid visualization for interactive applications.

1 Introduction

Particle-based methods are gaining popularity for physics-based animation of natural phenomena such as fluids or melting solids. Particles are used for example in grid-based methods to accurately track the fluid motion (e.g., [4, 9]) or as simulation nodes in meshless methods for solving the governing equations, e.g., using smoothed particle hydrodynamics (SPH) [7, 19].

The motion of a particle system can be visualized by rendering the individual particles as simple point sprites or spheres. This can be performed efficiently using current graphics hardware and is a viable approach for previewing visualizations of particle simulations. However, the resulting image is often not satisfactory due to high frequency artifacts and lack of proper visibility and shading computations.

High quality renderings can be generated by constructing a surface corresponding to the volume of particles. The most popular method for defining such a surface is by iso-surface extraction of an implicit function defined by the particles (e.g., using metaballs [1]). The extraction is most often performed using some variant of the marching cubes algorithm [15] on a background grid.

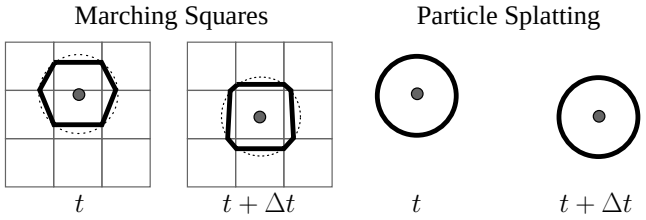

Figure 1: Left: the marching cubes (marching squares in 2D) algorithm can produce significantly different discretizations of a single particle sphere in subsequent time steps. Right: our particle splatting algorithm results in time-coherent renderings of the particle surface.

However, such grid-based methods entail various limitations. First, evaluating the implicit function at the grid points is computationally expensive, especially for large particle set sizes. Second and most importantly, when the particles move through the grid, the triangulation obtained from the marching cubes algorithm changes from one time step to the next (see also figure 1), leading to temporal discretization artifacts (such as popping highlights), which are clearly visible when using a low grid resolution. Hiding these artifacts requires increasing the grid resolution which in turn further prohibits interactive visualizations. Finally, marching cubes methods are often CPU intensive, significantly slowing down the whole physics-based simulation.

1.1 Contribution

In this paper, we will present a novel surface rendering algorithm for particle-based simulations. We overcome aforementioned limitations by using a splatting approach which directly uses the particles and can be entirely implemented on the GPU. From a high level point of view our algorithm proceeds by rasterizing each particle as a sphere and by blending the foremost overlapping spheres in image space to obtain smooth shaded surfaces. No connectivity or other neighborhood information is needed and grid-related discretization artifacts are completely avoided (cf. figure 1). Our rendering method results in interactive and high-quality renderings of large dynamic particle animations.

Note that although from an implementation point of view our algorithm proceeds along similar steps as a surfel splatting algorithm, the problem statement is quite different. We are given with an input point set of unoriented particles which are used as nodes for the physical simulation. They can be considered a discrete sampling of the fluid's *volume*. In contrast, when visualizing point set surfaces using surfel splatting, the input is a set of oriented surface points (i.e., surface normals are provided) which can be considered a discrete sampling of the object's *surface*.

1.2 Related Work

Because we will treat the simulation framework in this paper as a black box, the discussion of related work is limited to methods for surface extraction and rendering algorithms for particle-based simulations. For a more general overview on simulation algorithms for computer graphics, we refer the reader to [21].

A very popular particle-based method for fluid simulation is smoothed particle hydrodynamics (SPH) (see [17, 16, 14] for good overviews). Desbrun and Cani introduced SPH to the computer graphics community in a series of papers on the animation of soft highly deformable substances [6, 7, 8]. Based on [24], they propose to define the surface as the iso-level set of an implicit function defined from the mass density [7]. A similar surface definition for particle simulations is used in [19], [22] and [5]. They define the surface as the iso-level of a color field function [18] and use the marching cubes algorithm [15] to extract an explicit triangle mesh which samples the level set. An alternative surface tracking algorithm for particles is proposed in [8]. They store and evolve the implicit function on a background grid and introduce surface tension to hide the particle granularity.

A common limitation of the discussed Eulerian surface extraction algorithms is that (temporal) discretization artifacts may appear, due to the use of a background grid (cf. figure 1). To overcome this limitation, Müller et al. [19] propose a Lagrangian surface extraction method where they classify particles near the border of the fluid domain as *surface particles*. By converting the volumetric particles to surfels, a standard surface splatting algorithm [26] can be used. Although they avoid the discussed grid-related problems, their method still has to evaluate the (gradient of the) field function at the particle positions. Moreover, it fails in the case of a single isolated particle. Another Lagrangian surfel-based fluid tracking method is proposed in [12]. They maintain a point-sampled surface wrapped around and tracked along with the simulation particles. However, this method is very CPU intensive and too time consuming for interactive visualizations, due to the constant re-sampling and smoothing of the point set surface.

Our rendering algorithm is inspired by the work of [4], [2] and [11]. Carlson et al. [4] propose to advect and splat particles in a 3D grid. After low-pass filtering, they can construct a high-quality polygonal surface corresponding to the fluid flow. Since our system is targeted at interactive visualizations, we avoid the expensive 3D splatting and polygonization and propose to splat particles to image space using a technique very similar to surface splatting for surfel models [26, 2]. We show how our particle splatting algorithm corresponds to rasterization of a Lagrangian particle level set, introduced by Hieber and Koumoutsakos [11].

1.3 Overview

We start by discussing the surface definition based on a field function defined by the particles (section 2) and motivate how this can be formulated as a par-

ticle splatting algorithm. We discuss the rendering algorithm (section 3) based on sphere rasterization (section 3.1) and smooth blending of particle attributes (section 3.2). The accumulated surface normals and colors are written to a separate buffer which will be used in a final pass for per-pixel shading (section 3.3). We discuss implementation details in section 4 and results in section 5. We conclude this paper in section 6 with a discussion and an outlook on future work.

2 Motivation

The surface of the particle volume is traditionally computed as the iso-level of a field function defined by the particles. By assigning compactly supported weight functions W to the particles, such a field function at position $\mathbf{x}$ can be easily obtained as the sum of the contributions of each of the particles p_i:

$$\phi(\mathbf{x}) = \sum_{p_i} V_i W(\mathbf{x} - \mathbf{x}_i, h_i), \tag{1}$$

with $\mathbf{x}_i$ the center of the particle p_i and V_i and h_i its volume and support radius respectively. The surface can then be defined as $\phi^{-1}(I)$ with I an appropriate iso-value, i.e., it is defined as the location of points where the field function ϕ evaluates to a certain value I. This definition is analogous to the traditional metaballs [1] and is often denoted as a color field [19, 20].

The weight functions used in equation 1 can be used as well for interpolating particle properties such as for example a color value $\mathbf{c}_i$ used for shading:

$$\mathbf{c}(\mathbf{x}) = \frac{\sum_{p_i} \mathbf{c}_i V_i W(\mathbf{x} - \mathbf{x}_i, h_i)}{\sum_{p_i} V_i W(\mathbf{x} - \mathbf{x}_i, h_i)}. \tag{2}$$

Note that one could also use the more accurate opacity-weighted color interpolation scheme as proposed in [25]. However, as will be shown in section 3, this would require the particles to be sorted along the viewing direction, increasing the computational complexity.

The (unnormalized) surface normal $\mathbf{n}(\mathbf{x})$ at a point $\mathbf{x}$ (on the iso-level) can be obtained as the gradient of the level set function:

$$\mathbf{n}(\mathbf{x}) = \nabla\phi(\mathbf{x}) = \sum_{p_i} V_i \nabla W(\mathbf{x} - \mathbf{x}_i, h_i). \tag{3}$$

If radially symmetric weight functions are used, this can be written as:

$$\mathbf{n}(\mathbf{x}) = \nabla\phi(\mathbf{x}) = \sum_{p_i} V_i \frac{\mathrm{d}W(r, h_i)}{\mathrm{d}r} \frac{\mathbf{x} - \mathbf{x}_i}{\|\mathbf{x} - \mathbf{x}_i\|}, \tag{4}$$

with $r = \|\mathbf{x} - \mathbf{x}_i\|$. Thus: the surface normal at a point $\mathbf{x}$ is computed as the weighted sum of the surface normals (evaluated at $\mathbf{x}$) of the spheres with center $\mathbf{x}_i$ and radius $r = \|\mathbf{x} - \mathbf{x}_i\|$.

We will follow this observation and compute weighted average sphere normals for smooth shading of the fluid surface. We will do this by rasterizing the zero level of ϕ, i.e., the surface is defined as $\phi^{-1}(0)$. Note, however, that at the zero level set, there is only one particle contributing to ϕ and $\nabla\phi$. As a result, using colors and surface normals as in equations 2 and 4, results in non-smooth and discontinuous shading. Instead of computing surface properties exactly on the zero level set, we will follow a splatting approach where sphere colors and normals within an ϵ-distance to the zero level set are blended to obtain smooth surfaces. This can be implemented in a similar fashion as traditional surfel splatting algorithms [26, 23, 2] as will be shown in the next section. However, special attention is required to the construction of appropriate weight functions to ensure smooth continuous renderings.

Our method has the advantage that no neighborhood information between particles is required and that there is no root finding, i.e., we avoid explicit (and expensive) evaluation of the field function. Each particle is treated separately and splatted individually to accumulate the final shading parameters. As opposed to traditional particle-based fluid visualization algorithms (e.g., [19, 20, 5]) based on the marching cubes algorithm [15], our method does not suffer from temporal discretization artifacts (cf. figure 1).

3 Particle Splatting Algorithm

Our algorithm proceeds as follows (see also figure 2):

1. The zero level $\phi^{-1}(0)$ is rendered in the depth buffer by rasterizing the spheres $(\mathbf{x}_i, h_i)$ (see section 3.1). The resulting depth value is shifted over a distance ϵ to the back. Similar to [2], we refer to this step as the *visibility splatting* step.

2. Next, the spheres $(\mathbf{x}_i, h_i)$ are rendered again, using additive blending to accumulate the normals $\mathbf{n}_i$ and colors $\mathbf{c}_i$ of the fragments corresponding to the particles which pass the depth test (i.e., the fragments which are within a distance ϵ to the zero level set). During blending, we use appropriate weight functions which ensure a continuous blending of sphere colors and normals (see section 3.2). Hence, in this *attribute blending* step, we approximate the computation of a smooth surface color and normal as defined in equations 2 and 4 .

3. Finally, in the *shading step*, we normalize the accumulated particle properties and perform per-pixel shading to compute the actual color of the image pixel.

Note that, although we render a volume of particles, this rendering algorithm will rasterize a surface corresponding to these particles.

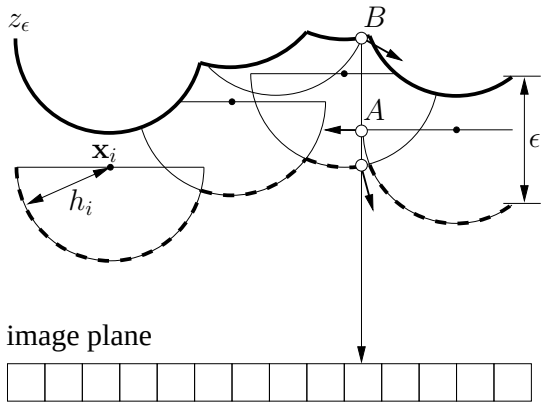

Figure 2: Illustration of the particle splatting algorithm. In a first pass, the particles are rendered as spheres to determine the depth of the foremost particles (dashed line). This depth is shifted backwards over a distance ϵ (bold line). In a second step, we blend normals and colors of the spheres which pass the depth test using additive blending. Appropriate weight functions ensure continuous blending (fragment A and B are assigned a zero weight). The blended attributes are used in a final pass for per-pixel shading.

3.1 Sphere Rasterization

Both in the visibility pass and the attribute blending pass, each particle p_i is rendered as a sphere. Given the particle's position $\mathbf{x}_i$ and radius h_i, we approximate the size h_i' of the projected sphere by perspectively scaling h_i [3]. Each particle p_i is then rendered as a single square of size $2h_i' \times 2h_i'$ in image space.

For each of the fragments covered by this square, we decide whether it lies inside or outside the projection of the sphere. Instead of using local ray casting (as in [2]) to obtain perspectively correct sphere projections, we simply discard fragments with projected position $\mathbf{x}'$ outside a circle with center $\mathbf{x}_i'$ and radius h_i', i.e., when $\|\mathbf{x}' - \mathbf{x}_i'\| > h_i'$. Note that, although computing perspective correct sphere projections can be implemented fairly easy, it requires considerably more computations. As we experienced no significant quality degradation, we propose to use the discussed approximation.

Given the fragment position $\mathbf{x}'$ in the projected square and its world space position $\mathbf{x}$ (on the square), we can compute the depth value $z_i(\mathbf{x}')$ for this fragment simply as:

$$z_i(\mathbf{x}') = z_i - \sqrt{(h_i)^2 - \|\mathbf{x} - \mathbf{x}_i\|^2}, \tag{5}$$

with z_i the depth value corresponding to the sphere center $\mathbf{x}_i$. We define $z_\epsilon(\mathbf{x}') = \min_{p_i}(z_i(\mathbf{x}')) + \epsilon$ as the pixel's ϵ-shifted minimal depth value which is written to the z-buffer as the result of the first rendering pass (see also figure 2).

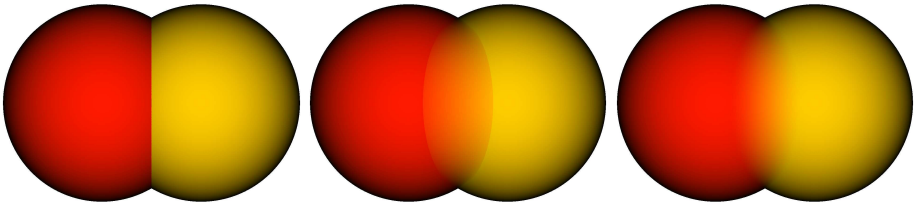

Figure 3: Two spheres of equal radius h_i are rendered a distance h_i apart. On the left, the two spheres are rendered without blending, causing a discontinuity in normals and colors. In the middle image, the spheres are blended using only W_i^1. As can be seen at the transition from one sphere to the next, discontinuities are still visible. In the right image, both W_i^1 and W_i^2 are used, resulting in a smooth blending. We used $\epsilon = h_i/4$ for the middle and right image.

Similar calculations yield the normal $\mathbf{n}_i(\mathbf{x}')$ corresponding to fragment $\mathbf{x}'$ for particle p_i. We will use these sphere normals in the second rendering pass to compute smooth surface normals.

3.2 Sphere Attribute Blending

In the second pass of our algorithm, the frontmost sphere attributes are accumulated using additive blending. To obtain a smooth shaded surface without discontinuities, we use a multiplication of two weighting functions. The first function defines a linear fall-off around the projected particle's position $\mathbf{x}_i'$ in the plane parallel to the view plane:

$$W_i^1(\mathbf{x}') = \begin{cases} 1 - \frac{\|\mathbf{x}' - \mathbf{x}_i'\|}{h_i'} & \text{if } \|\mathbf{x}' - \mathbf{x}_i'\| \leq h_i', \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

This weight ensures that fragments corresponding to points on the silhouette of a sphere (e.g., fragment A on figure 2) have zero weight, thereby enabling smooth blending of overlapping spheres. Note that we get the computation of W_i^1 for free, as we also use it for testing whether a fragment lies inside or outside the projected sphere. The effect of using W_i^1 is illustrated for two spheres in the middle of figure 3.

Although W_i^1 allows smooth blending of overlapping sphere properties, discontinuities in the blended shading attributes can still arise. As depicted in figure 2, some spheres are only partially visible, i.e., parts of are in front of the z-buffer, others are behind. Blending discontinuities arise at fragments that correspond to points on a sphere intersecting the z-buffer depth (e.g., fragment B on figure 2), as the weight W_i^1 is typically non-zero there. This can be clearly seen on the middle image of figure 3. To avoid these discontinuities, we introduce a second weighting function W_i^2 which takes into account the depth

information:

$$W_i^2(\mathbf{x}') = \frac{z_\epsilon(\mathbf{x}') - z_i(\mathbf{x}')}{\epsilon}. \tag{7}$$

Hence, W_i^2 evaluates to zero at fragments which coincide with the ϵ-shifted depth buffer.

The product of the two weighting functions results in a C^0 continuous function (except at depth discontinuities) which evaluates to zero both at the silhouette of a sphere as well as at the sphere positions which coincide with the depth buffer. As a consequence, weighting the sphere colors and normals with $W_i^1 W_i^2$ results in a continuous blending of overlapping particles (see also figure 3, right).

Note that other weight functions can be used for W_i^1 and W_i^2 as long as their product satisfies aforementioned properties. As defined in equation 7, W_i^2 mostly evaluates to 1 and is only introduced to ensure zero contribution at spheres crossing the depth image, justifying the choice of such a simple blending function.

To summarize, in the second pass of the algorithm, we compute for each fragment the accumulated color, normal and weight:

$$\mathbf{c}(\mathbf{x}') = \sum_{p_i} W_i^1(\mathbf{x}') W_i^2(\mathbf{x}') \mathbf{c}_i, \tag{8}$$

$$\mathbf{n}(\mathbf{x}') = \sum_{p_i} W_i^1(\mathbf{x}') W_i^2(\mathbf{x}') \mathbf{n}_i(\mathbf{x}'), \tag{9}$$

$$W(\mathbf{x}') = \sum_{p_i} W_i^1(\mathbf{x}') W_i^2(\mathbf{x}'). \tag{10}$$

3.3 Per-pixel Deferred Shading

In the final pass of the algorithm, we compute per-pixel shading using the accumulated surface properties. The normalized surface normal is given by $\mathbf{n}(\mathbf{x}')/\|\mathbf{n}(\mathbf{x}')\|$ and the normalized color attributes by $\mathbf{c}(\mathbf{x}')/W(\mathbf{x}')$. The world coordinates for each fragment can easily be computed from the depth value $z_\epsilon(\mathbf{x}')$, obtained from the first pass, by inverse projection.

As will be shown in section 5, we can incorporate various custom shading effects in the deferred shading pass, similar to [2].

4 Implementation Details

We implemented the proposed particle splatting algorithm using OpengGL and Cg. Sphere rasterization is performed as discussed in section 3.1. We send each particle as a single vertex to the pipeline and render it as a `GL_POINT`. A vertex shader is responsible for computing the projected sphere size. In the first pass, a simple fragment program is used to fill the ϵ-shifted depth buffer (we use $\epsilon = 1.5h_i$, with h_i the support radius of particle p_i). In the attribute blending pass, we proceed along similar steps and render the particles a second time. A more advanced fragment program is used to compute the weights,

Animation	Visibility	Blending	Shading
Melting (110k)	35%	60%	5%
Beethoven (18k)	25%	45%	30%
Bowl 1 (59k)	29%	48%	23%
Bowl 2 (63k)	29%	52%	19%

Table 1: Average number of particles and percentage of time spent on each of the three rendering passes, averaged over the whole animation, while moving the camera around the scene.

colors and normals as discussed in section 3.2. Fragments which are invisible are discarded in the beginning of the fragment program using the depth buffer from the first rendering pass. This way, we can cull most of the spheres early and spend most of the work on the contributing particles. We use multiple render targets and additive blending to accumulate the weighted colors and normals. In the final shading pass, we render a window-sized rectangle and normalize the accumulated surface normals and color attributes in a fragment program which are then used for per-pixel shading.

5 Results

The resulting images and accompanying movies are rendered on a 2.2GHz desktop PC with a GeForce 7800 graphics board at a resolution of 640×480 pixels. Relative timings for each of the three passes of the rendering algorithm are given in table 1.

Figure 4 shows four frames from a pre-computed melting simulation of the Stanford bunny [4]. Approximately 110k particles were used to track the motion of the melting bunny. For this animation we used shadow mapping and a custom jelly shader, which calculates Phong shading together with refraction and reflection. The animation is rendered at an average rate of 11 FPS (or 1.2M particles per second). This timing includes all shading computations.

Figure 5 shows different frames from an SPH-based viscous fluid animation, rendered with our algorithm. The average number and maximal number of particles are 18k and 20k respectively. The average rendering time, including the rendering of Beethoven's bust and shadow computation, is 47 FPS (or 850k particles per second).

Finally, figure 6 shows renderings for two other SPH-based viscous fluid simulations. Two streams of oil-like fluids are mixed in a bowl. Due to the blending of particle colors in the attribute blending pass, we are able to visualize the mixing of fluid colors in the simulations. The average and maximal number of particles are approximately 60k and 120k respectively. The renderings of these animations run at an average of 20 FPS (or 1.2M particles per second).

Compared to other interactive particle visualization techniques, we believe that we achieve significantly higher frame rates. Müller et al. [19] report frame rates of 5 FPS for a simulation with 2.2k particles and the marching cubes algo-

rithm for surface extraction. Using the surfel rendering algorithm, they obtain frame rates of 20 FPS. Note that these timings include the time spent on the SPH simulation (they do not provide separate timings for surface construction and rendering). Keiser et al. [12] report surface reconstruction timings of 1.3 seconds for a fluid simulation with 3k particles (the time spent on rendering is not included here). Our system achieves rendering frame rates of over 100 FPS for dynamic simulations of approximately 10k particles and around 10 FPS for simulations of approximately 100k particles.

6 Discussion & Future Work

The main strength of our algorithm lies in the fact that it works directly with the particle positions and radii. There is no explicit surface extraction step involved and no spatial data structures and neighborhood queries are required to evaluate the field function. Therefore, it is extremely well-suited for dynamic particle sets.

Our visualization method is *Lagrangian* in nature, i.e., in essence, the visualized surface detail is advected with the particles and evaluated at image resolution. This way, we avoid discretization artifacts which typically occur in *Eulerian* grid-based methods. As such, we feel that our particle splatting algorithm is in the same spirit and has similar advantages as meshless particle-based simulation algorithms, which are also Lagrangian formulations.

Although our particle splatting algorithm allows rapid and high-quality visualization of surfaces corresponding to particle-based simulations, there are some limitations and possible directions for future work.

Firstly, as discussed in section 2, the rasterized surface corresponds to the zero level set of the field function defined by the particles. As a consequence, the visualized surface might appear blobby. This is especially the case when single disconnected particles are visualized (see for example figure 5). To obtain surfaces which more tightly fit the particle set, it is preferable to use other level sets $\phi(I)^{-1}$ with $I > 0$. Visualizing the corresponding surface would require ray tracing with root finding (see [10] for a thorough discussion). However, as only a few particles contribute at each point on the level set, this root finding is a very localized operation. We are currently exploring ways to efficiently implement this on graphics hardware using a similar particle splatting approach.

Another limitation of our rendering algorithm is that it does not support backface culling. At the time of rendering, there is no surface information available and culling can only be performed in the second pass for particles which are occluded by the ϵ-shifted depth buffer. Culling these particles results in approximately 15% speed increase for the Beethoven and bowl simulations and even 25% for the melting bunny scene. However, efficient data structures might help culling more invisible spheres.

Better rendering quality could be obtained by using potential additional information provided by the simulation algorithm. For example when using surface tension forces in an SPH simulation [19], the gradient of the level set

function (cf. equation 3) is computed and known for the particles. Passing this information on to the rendering algorithm could allow using other shapes such as ellipsoids to rasterize the individual particles. This could reduce the blobbyness of the resulting surfaces. However, the method proposed in this paper is general enough to also be used for other particle-based simulations that do not provide this information (e.g., [4]).

Finally, we are planning to incorporate screen-space anti-aliasing filters, similar to the EWA approximation of [2] to handle extreme minifications. Also, it would be interesting to use our particle splatting algorithm together with a GPU-based particle simulation algorithm such as the one proposed in [13].

7 Conclusion

Our particle splatting algorithm provides a new way to interactively visualize a smoothly shaded surface corresponding to a volume of particles. In contrast to other techniques, the algorithm does not require any neighborhood information, nor does it suffer from time-dependent discretization artifacts as the surface detail is simply advected with the particles and evaluated at image resolution. The algorithm furthermore scales linearly in the number of particles and can be implemented entirely on the GPU, freeing CPU power which can be used by the simulation algorithm.

References

[1] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.

[2] Mario Botsch, Alexander Hornung, Matthias Zwicker, and Leif Kobbelt. High-quality surface splatting on today's GPUs. In *Proceedings of Symposium on Point-Based Graphics 2005*, pages 17–24, 2005.

[3] Mario Botsch, Michael Spernat, and Leif Kobbelt. Phong splatting. In *Proceedings of Symposium on Point-Based Graphics 2004*, pages 25–32, 2004.

[4] Mark Carlson, Peter Mucha, R. Brooks Van Horn III, and Greg Turk. Melting and flowing. In *Proceedings of the 2002 ACM SIGGRAPH Symposium on Computer Animation*, pages 167–174, July 2002.

[5] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based viscoelastic fluid simulation. In *Symposium on Computer Animation 2005*, pages 219–228, july 2005.

[6] Mathieu Desbrun and Marie-Paule Cani. Animating soft substances with implicit surfaces. In *Computer Graphics Proceedings*, pages 287–290. ACM SIGGRAPH, 1995.

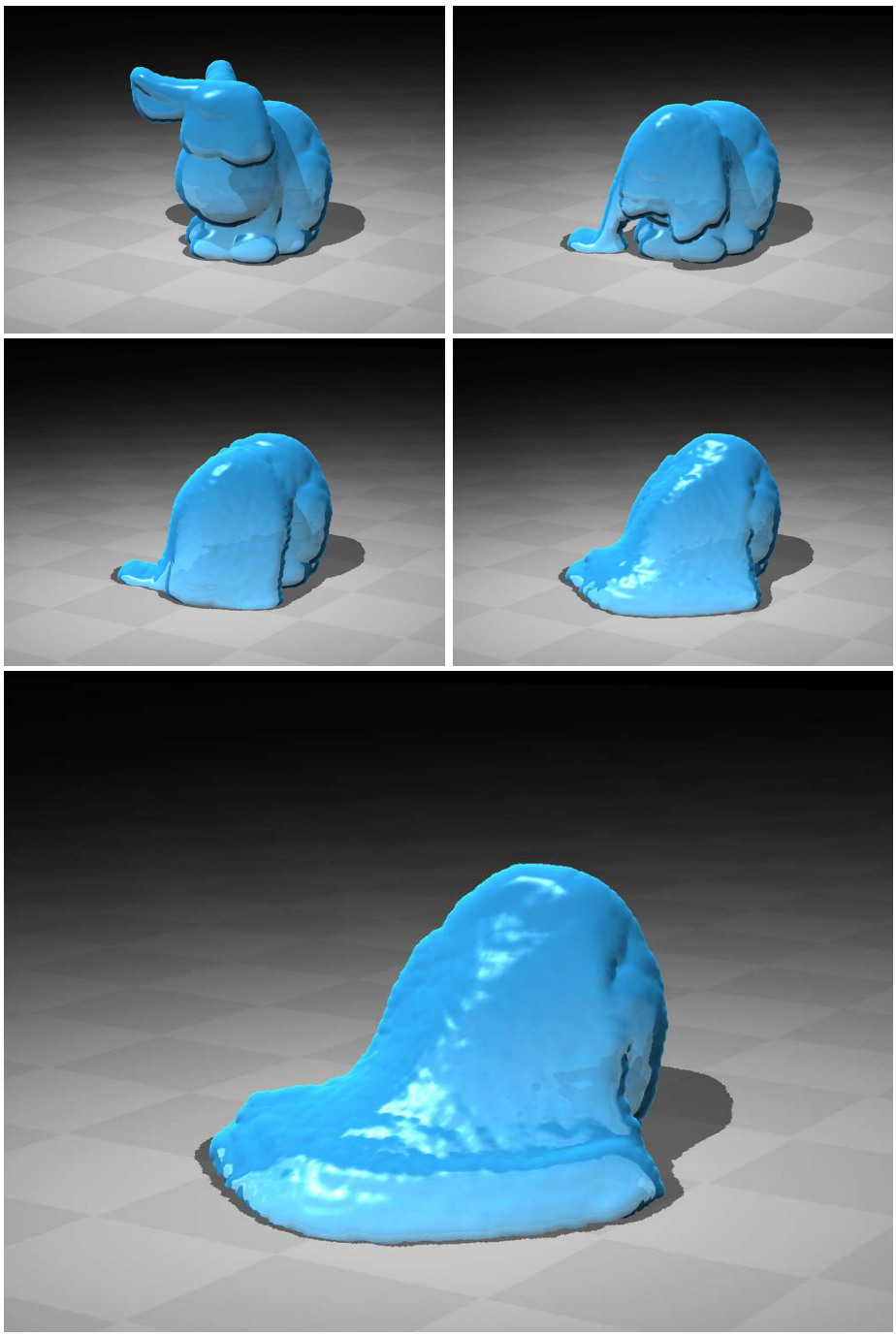

Figure 4: The animation of the melting Stanford bunny [4] is visualized with our particle splatting algorithm and shaded with a custom jelly shader.

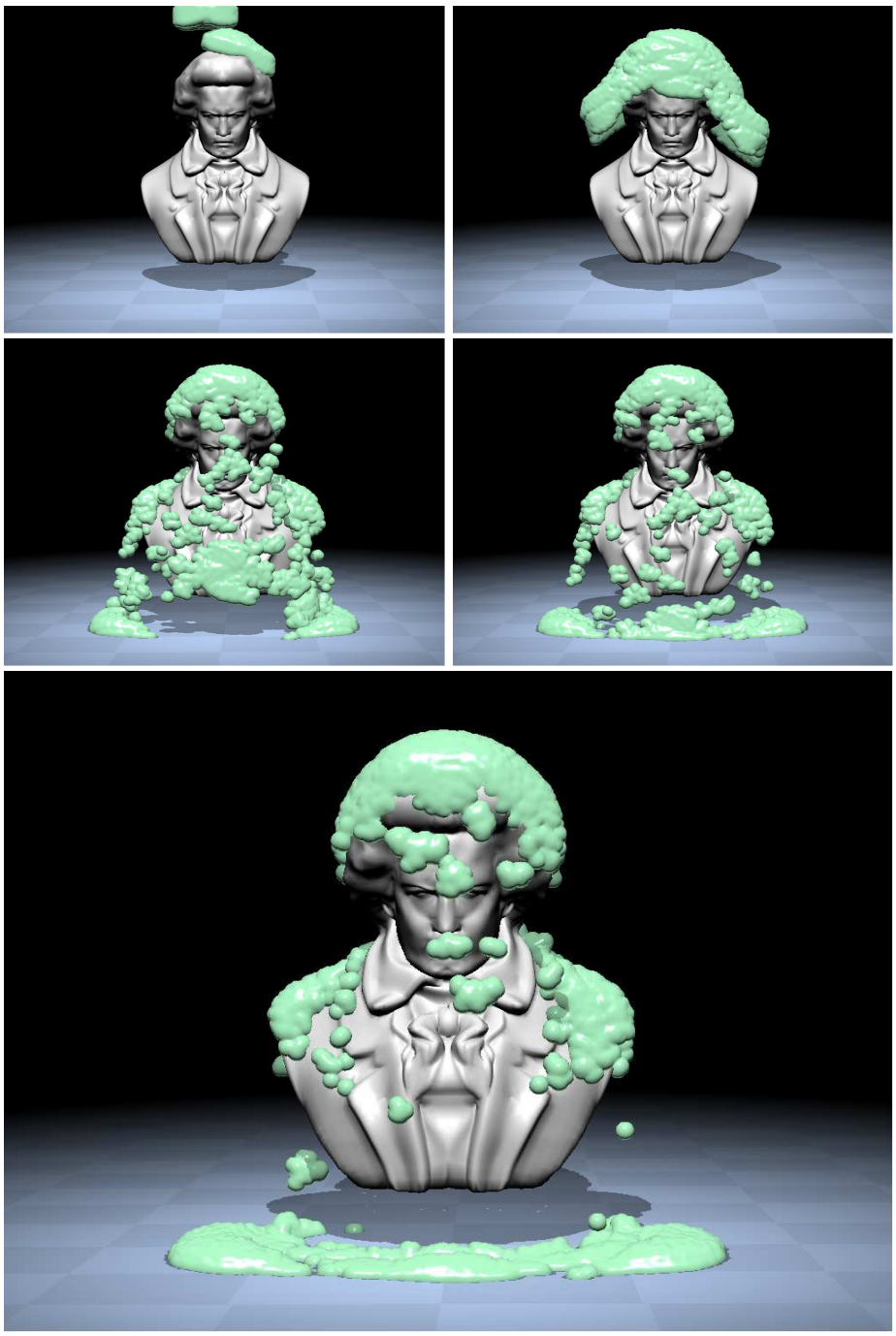

Figure 5: Dropping viscous goop on Beethoven's bust. This animation is obtained from a viscous fluid simulation using smoothed particle hydrodynamics.

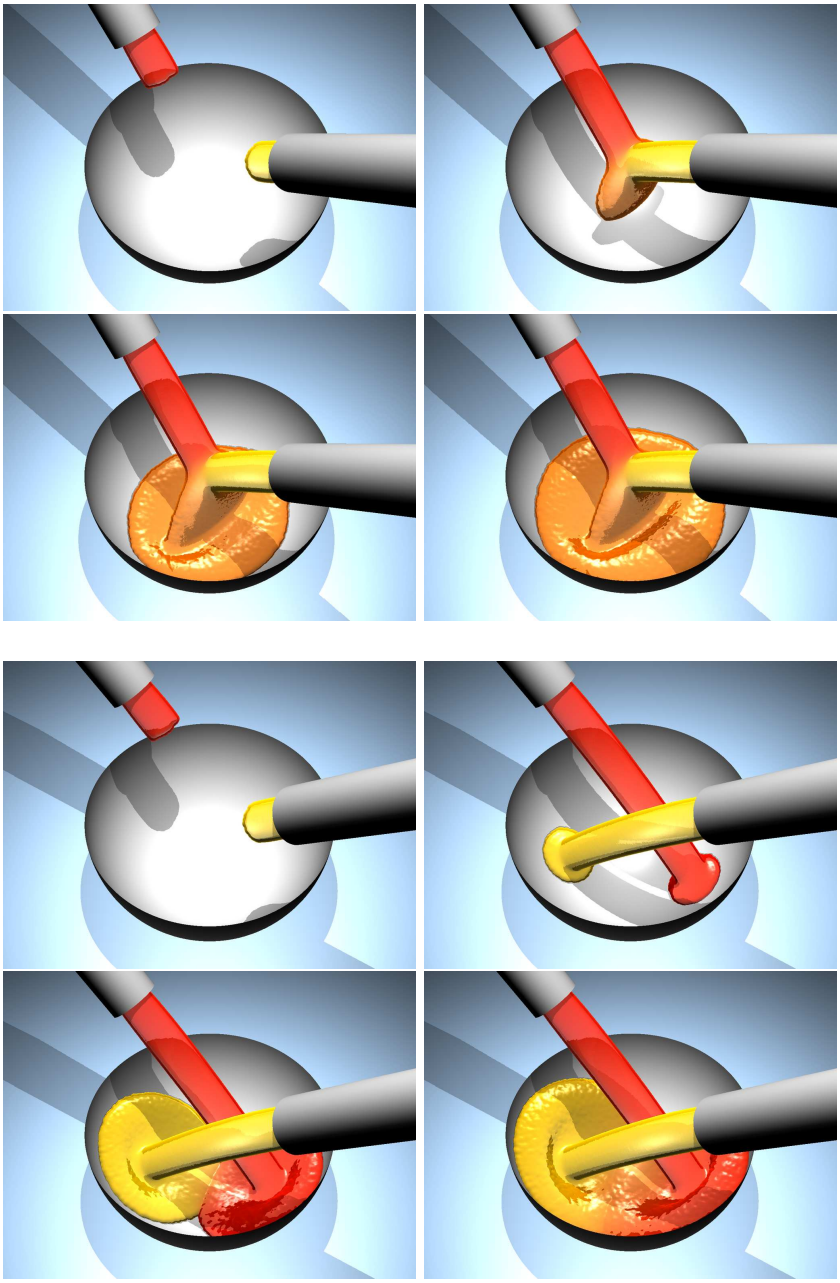

Figure 6: Mixing two oil-like fluids in a bowl. The simulations are performed using smoothed particle hydrodynamics. Due to the color blending in the second rendering pass, we are able to visualize the mixing fluid colors.

[7] Mathieu Desbrun and Marie-Paule Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. In *6th Eurographics Workshop on Computer Animation and Simulation '96*, pages 61–76, 1996.

[8] Mathieu Desbrun and Marie-Paule Cani-Gascuel. Active implicit surface for animation. In *Proceedings of Graphics Interface*, pages 143–150, 1998.

[9] Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys.*, 183(1):83–116, 2002.

[10] John C. Hart. Ray tracing implicit surfaces. In *SIGGRAPH 93 Modeling, Visualizing, and Animating Implicit Surfaces course notes*, pages 13–1 to 13–15. 1993.

[11] Simone E. Hieber and Petros Koumoutsakos. A Lagrangian particle level set method. *Journal of Computational Physics*, 210(1):342–367, nov 2005.

[12] Richard Keiser, Bart Adams, Dominique Gasser, Paolo Bazzi, Philip Dutré, and Markus Gross. A unified Lagrangian approach to solid-fluid animation. In *Proceedings of the Eurographics Symposium on Point-Based Graphics*, 2005.

[13] Andreas Kolb and Nicolas Cuntz. Dynamic particle coupling for GPU-based fluid simulation. In *Proceedings of the 18th Symposium on Simulation Techniques*, pages 722–727, 2005.

[14] G. R. Liu and M. B. Liu. *Smoothed particle hydrodynamics, a meshfree particle method*. World Scientific Publishing, 2003.

[15] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, New York, NY, USA, 1987. ACM Press.

[16] J. J. Monaghan. Smoothed particle hydrodynamics. *Rep. Prog. Phys.*, 68:1703–1758, 2005.

[17] J.J. Monaghan. Smoothed particle hydrodynamics. *Annu. Rev. Astron. and Astrophysics*, 30:543, 1992.

[18] J.P. Morris. Simulating surface tension with smoothed particle hydrodynamics. *International Journal for Numerical Methods in Fluids*, 33(3):333–353, 2000.

[19] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 154–159, 2003.

[20] Matthias Müller, Barbara Solenthaler, Richard, and Markus Gross. Particle-based fluid-fluid interaction. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 237–244, New York, NY, USA, 2005. ACM Press.

[21] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. In *Eurographics 2005 State of the Art Report*, 2005.

[22] Simon Premoze, Tolga Tasdizen, James Bigler, Aaron E. Lefohn, and Ross T. Whitaker. Particle-based simulation of fluids. *Computer Graphics Forum*, 22(3):401–410, 2003.

[23] L. Ren, H. Pfister, and M. Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum*, 21(3):461–470, 2002.

[24] D. Tonnesen. Modeling liquids and solids using thermal particles. In *Proceedings of Graphics Interface*, pages 255–262, 1991.

[25] Craig M. Wittenbrink, Thomas Malzbender, and Michael E. Goss. Opacity-weighted color interpolation, for volume sampling. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 135–142, New York, NY, USA, 1998. ACM Press.

[26] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *Proceedings of ACM SIGGRAPH 2001*, pages 371–378, 2001.