A Level-set Method for Skinning Animated Particle Data

Haimasree Bhattacharya, Yue Gao, and Adam W. Bargteil

Abstract—In this paper, we present a straightforward, easy to implement method for particle skinning—generating surfaces from animated particle data. We cast the problem in terms of constrained optimization and solve the optimization using a level-set approach. The optimization seeks to minimize the thin-plate energy of the surface, while staying between surfaces defined by the union of spheres centered at the particles. Our approach skins each frame independently while preserving the temporal coherence of the underlying particle animation. Thus, it is well-suited for environments where particle skinning is treated as a post-process, with each frame generated in parallel. Moreover, our approach is integrated with the OpenVDB library and the underlying partial differential equation is amenable to implicit time integration. We demonstrate our method on data generated by a variety of fluid simulation techniques and simple particle systems.

Index Terms—Particle skinning, level-set methods, particle systems, fluid simulation, surface smoothing, constrained smoothing.

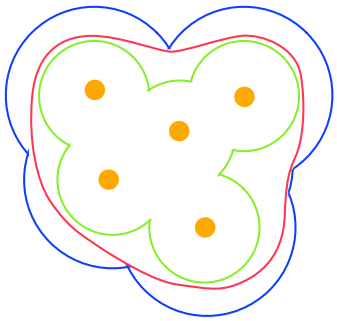

Fig. 1. Our approach seeks a smooth surface (red curve) that is constrained to lie between two surfaces, S_{min} (green curve) and S_{max} (blue curve), defined by the union of spheres centered at the particles (orange points).

1 Introduction

Particles are a ubiquitous primitive in computer animation. From simple particle systems to high-resolution smoothed particle hydrodynamics simulations, particles have been used to animate a vast range of phenomena and a huge number of special effects. While early particle systems rendered the particles directly into a framebuffer [31], [36], more recently it has become common to use the particles to define a volume, which is bounded by a surface. Consequently, a vast number of techniques and industry tools have been developed to generate surfaces defined by particle animations, an operation we refer to as *particle skinning*.

Inspired by the work of Williams [44], we cast particle skinning as a constrained optimization problem: intuitively, we seek the "smoothest" surface that approximates the geometry implied by the particles. We formalize the intuitive notion of "smoothness" by minimizing the thin-plate energy of the surface. To ensure that the surface captures the geometry implied by the particle set we constrain the surface to lie between two surfaces S_{min} and S_{max}, defined as the constructive solid geometry (CSG) union of spheres of radius r_{min} and r_{max}, respectively, centered at the particles (see Figure 1).

Our key technical contribution is that, unlike Williams [44], we solve this optimization problem using a level-set approach. We store signed-distance functions that represent the constraint surfaces and the surface being smoothed. Enforcing the constraints is as simple as taking a min and max for each grid-point after each smoothing iteration. Smoothing is accomplished by solving a level-set equation based on the biharmonic operator, ∇^4. Our level-set approach allows us to use the same spatial discretization for every frame, which is key to achieving temporal coherence even though we process every frame independently.

We achieve efficient memory utilization and multi-threaded computation by integrating our approach with OpenVDB [25]—"an open source C++ library comprising a novel hierarchical data structure and a suite of tools for the efficient storage and manipulation of sparse volumetric data discretized on three-dimensional grids." [28] We also describe an implicit integration scheme for solving the level-set partial differential equation. These improvements result in lower memory usage and up to an order of magnitude improvement in speed over the initial explicit, uniform-grid implementation of our method.

Additionally, our approach is flexible and supports variably sized and anisotropic particles, with anisotropy defined either by particle neighborhoods [47] or velocities. Most importantly, our approach produces smooth, temporally coherent surfaces while processing every frame independently. We demonstrate our method on data generated by a variety of fluid simulation techniques and simple particle systems (see Figure 14) [1].

H. Bhattacharya and A.W. Bargteil are with the University of Utah. Y. Gao is with TangoME, Inc.

1. A preliminary version of this article appeared in the proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation [5]. This extended version includes integration with the OpenVDB library (see Section 3.4), implicit time integration (see Section 3.5), and obstacle handling (see Section 3.6).

2 Related Work

The pioneering work of Blinn [6] introduced *blobbies*, also known as metaballs, as a primitive for representing shapes. This approach defines a scalar field that is the sum of three-dimensional Gaussian kernels centered at a set of points. Surfaces are then taken to be a particular iso-contour of this scalar field. Blobbies tend to smooth regions of the surface between several particles, making them preferable to a union of spheres in many contexts. As the name implies, this approach tends to produce "blobby," or lumpy, surfaces. In practice, smoothing techniques are often applied to these surfaces in an attempt to remove the "blobbiness." One of the earliest attempts to skin animated particle systems [10] rasterized blobbies onto a regular grid and applied surface tension forces to smooth the surface.

Recent interest in the graphics community in particle-based fluid simulations has sparked a renewed interest in creating surfaces from particle sets. Müller and colleagues [22] improved upon metaballs by dividing a particle's contribution to the scalar field by the SPH estimate of the density. Zhu and Bridson [50] describe a quite different approach that defines a distance to the surface at every point in space. This distance is computed based on scattered data interpolation of particle radii and a weighted average of the distance to neighboring particles. While this approach produces very good results early in the animation, without a way to update particle radii, the results deteriorate over time. Shen and Shah [34] use a similar approach and note temporal discontinuities, which they address by blending adjacent frames. Adams and colleagues [1] improved upon the results of Zhu and Bridson [50] by recomputing the particle-to-surface distances every timestep. Unfortunately, the changing distances make surface generation a sequential process, ill-suited to treatment as a parallel post-process. Moreover, both of these approaches sometimes produce spurious surfaces in concavities. Solenthaler and colleagues [38] address this problem by preventing the generation of surface in regions where the average neighbor position changes quickly. Museth and colleagues [26] describe a very high-resolution particle surfacing pipeline that incorporates a variety of post-processing techniques including temporal and spatial anti-aliasing. More recently, Museth [24] presented a particle skinning approach that applies of a variety of simple and fast three-dimensional operators, such as dilation and erosion filters, in the OpenVDB framework.

Perhaps, the first to cast the problem of generating surfaces from particle data as a constrained optimization problem was Williams [44]. We embrace this formulation, but diverge in how the optimization is performed. Whereas they tessellate an initial surface using a variation of marching cubes [21] and perform mesh-based smoothing to optimize the surface, we perform the optimization using a level-set approach. Mesh-based smoothing methods are highly sensitive to mesh topology and Williams' *marching tiles* mesh extraction method cannot guarantee that meshes adjacent in time have the same topology. Consequently, some temporal incoherence is expected. By adopting a level-set framework, we can ensure that the level-set mesh is identical for every frame allowing us to preserve the temporal coherence in the particle data. We note that Williams [44] suggested a level-set approach as future work and that Sin and colleagues [37] briefly mention using such a variation (actually, an early version of our code). This paper details the approach and provides a variety of examples and comparisons with alternative methods.

More recently, Yu and Turk [47] demonstrated very impressive results for the particle skinning problem. Their approach used a single pass of Laplacian smoothing of particle positions, followed by defining a metaballs-like surface with anisotropic smoothing kernels. The anisotropic kernels conform far better to the surface defined by the particles than isotropic kernels. Theirs is the first approach we know of that achieves appealing, smooth surfaces for each frame independently without introducing any temporal artifacts. This paper only addresses the surface smoothing aspect of the problem. Instead of performing a single pass of Laplacian smoothing we minimize the thin-plate energy using a level-set method. This approach results in smoother surfaces than those of Yu and Turk [47] (see Figure 12). Moreover, our approach can also make use of anisotropic kernels (see Figure 11).

Our work is also related to the rich body of research on surface smoothing, or fairing. Graphics researchers have largely focused on smoothing surfaces with explicit, mesh-based representations (see e.g., [8], [11], [18], [20], [32], [40]–[43]). However, to avoid flickering artifacts from changing mesh structure, we prefer a level-set approach. Level-set methods are well known in computer graphics and vision and have been used for a wide variety of problems from virtual sculpting [2] to surface reconstruction [48] to liquid surface tracking [15]. A principal use of level-set methods is for surface smoothing. Early work examined motion by mean curvature, which is closely related to Laplacian smoothing. Chopp and Sethian [9] (see also [39]) examined motion by intrinsic Laplacian of curvature and describe the numerical difficulties they encounter. As such motion minimizes the bending energy independent of the quality of the parameterization, we initially took this approach. However, we found that the linear biharmonic operator gave as good results and was numerically much more stable. Premoze and colleagues [30] also seek a smooth surface that approximates particle data and take a level-set approach. However, their approach initializes the optimization for every frame with the surface from the previous frame. Surface generation then becomes a sequential process, ill-suited to parallel processing. For a treatment of level-set methods in general see the texts by Sethian [33] and Osher and Fedkiw [29].

Generating surfaces from animated particles can be cast as a surface tracking problem and in this way is related to the vast literature on surface tracking for the simulation of liquids (e.g. [4], [14], [17], [23], [45], [46]). Most closely related to our approach are the particle level-set methods [14], [17]. Like our approach these methods combine tracked particles with level-set methods. However, these approaches allow bidirectional feedback between the particle and level-set representations and are sequential in nature. In contrast, in our approach the particles initialize the level set and provide constraints, but are static during the smoothing iterations.

3 METHOD

Our method comprises three steps. First, a volumetric grid is initialized with signed-distance fields representing S_{min}, S_{max}, and S_0, the initial guess of the smooth surface. Second, a constrained optimization smoothes the surface. Third, an explicit representation of the surface is extracted and rendered.

For clarity of exposition, we first describe the method in the simple context of a regular grid and explicit time integration before describing integration with OpenVDB and implicit time integration.

3.1 Level-set Initialization

We begin by initializing three scalar fields: ϕ_{min}, ϕ_{max}, and ϕ_0. ϕ_{min} and ϕ_{max} represent the two union-of-spheres constraint surfaces, while ϕ_0 is the starting point for the optimization (see Section 3.2). The initialization of ϕ_{min} and ϕ_{max} is straightforward and involves a distance calculation to the nearest particle and subtraction of r_{min} and r_{max}, respectively. More formally,

$$\begin{aligned}\phi_{min}(i,j,k) &= \min_p \|\mathbf{T}(\mathbf{x}(i,j,k)-\mathbf{x}(p))\| - r_{min}\\ \phi_{max}(i,j,k) &= \min_p \|\mathbf{T}(\mathbf{x}(i,j,k)-\mathbf{x}(p))\| - r_{max},\end{aligned} \quad (1)$$

where (i,j,k) refers to a grid point, and $\mathbf{x}(i,j,k)$ and $\mathbf{x}(p)$ are the world-space positions of the grid point and the particle, p. $\mathbf{T}$ is an optional transformation to allow for anisotropy. We accomplish this initialization by performing a generalized rasterization of the particles onto the level-set gridpoints. More specifically, we initialize the level-set grid values to a large positive background value. Then we iterate over the particles and, for each grid point within some radius, set its value to the minimum of its current value and the value determined by the current particle.

Choosing the initial surface too near either constraint surface can increase the number of smoothing iterations required. While many values work well, we generally initialize ϕ_0 as,

$$\phi_0(i,j,k) = 0.5\,(\phi_{min}(i,j,k)+\phi_{max}(i,j,k)). \quad (2)$$

After initialization, a fast sweeping method [49] is run to ensure that ϕ_{min}, ϕ_{max}, and ϕ_0 are signed-distance functions. In all our examples, we apply a small number of iterations (15) of Laplacian smoothing ($\phi_t = \nabla^2\phi\|\nabla\phi\|$) to ϕ_0. Like Williams [44], we have found that this preprocessing reduces the number of constrained optimization iterations required.

As is typical in level-set methods, our approach requires that we maintain ϕ only in a narrow band around the surface. In our implementation, ϕ is updated at all points within three grid cells of the zero level-set. We store all values on six regular grids (ϕ, ϕ_{min}, ϕ_{max}, and three grids for intermediate computations) of doubles, leading to a memory cost of 48 bytes per grid-point. Thus, grids on the order of 200^3 typically fit in less than 0.5 GB, while 400^3 requires more than 3 GB. However, we note that the streaming nature of the smoothing computations leads to near-optimal cache performance.

All our level-set grids can be thought of as being subgrids of an infinite background grid with grid-spacing h and containing the origin $(0,0,0)$. The fact that every frame uses the same background grid is the key to maintaining the temporal coherence of the particle data.

3.2 Constrained Optimization

We seek to minimize the thin-plate energy,

$$\begin{aligned}&E_{thinplate}(\phi) =\\ &\quad\frac{1}{2}\int_\Omega \left(\phi_{xx}^2+\phi_{yy}^2+\phi_{zz}^2+\,2\phi_{xy}^2+2\phi_{yz}^2+2\phi_{zx}^2\right)dxdydz.\end{aligned} \quad (3)$$

The thin-plate energy is a linearization of the bending energy. However, the thin-plate energy is not an intrinsic property of the surface and penalizes the parameterization of the surface as well as the shape of the surface. In practice, the muddying of the parameterization and the shape is not a serious concern as maintaining $\|\nabla\phi\| \approx 1$ is common practice in level-set methods and, if ϕ is nearly a signed-distance field, $E_{thinplate}$, like the various intrinsic bending energies, is at a minimum for a sphere (see Figure 10). Moreover, the thin-plate energy leads to a much simpler and numerically stable optimization problem than intrinsic energy measures; see Chopp and Osher [9] for a discussion of the difficulties encountered when solving for motion by the intrinsic Laplacian of curvature. The variational derivative of $E_{thinplate}$ is the biharmonic (or bi-Laplacian) operator leading to the level-set equation,

$$\begin{aligned}&\phi_t = -\nabla^4\phi\|\nabla\phi\| =\\ &\quad-\left(\,\phi_{xxxx}+\phi_{yyyy}+\phi_{zzzz}+2\phi_{xxyy}+2\phi_{yyzz}+2\phi_{zzxx}\right)\|\nabla\phi\|,\end{aligned} \quad (4)$$

which we integrate through fictitious time by solving

$$\phi^{t+\Delta t} = \phi^t - \Delta t\nabla^4\phi\|\nabla\phi\|. \quad (5)$$

The various high-order derivatives of ϕ in Equation (4) can be straightforwardly discretized using two rounds of second-order centered finite differences. We also use a second-order centered difference for the $\|\nabla\phi\|$ term. Constraints are enforced after every smoothing iteration by taking

$$\phi(i,j,k) = \min(\phi_{min}(i,j,k), \max(\phi_{max}(i,j,k), \phi(i,j,k))). \quad (6)$$

Periodically, a fast sweeping method ensures that ϕ is approximately a signed-distance function, preventing $\|\nabla\phi\|$ from becoming near zero everywhere.

To avoid applying different degrees of smoothing to different frames, which may lead to temporal incoherence, we do not iterate until convergence. Instead, each frame is integrated for the same total fictitious time by applying a fixed number of smoothing passes. This approach does imply that the number of smoothing passes is determined by the frame that requires the most, but this restriction has not been problematic in practice. In fact, as described in Section 3.7 we set the default number of smoothing passes sufficiently large to handle all of our examples.

Finally, we note that we do not have a proof that this simple optimization algorithm necessarily converges to the minimum thin-plate energy. However, this algorithm has always achieved good results in practice.

3.3 Surface Extraction

The final step in our approach is to extract the surface—the zero-set of ϕ_{final}. Our implementation applies marching tetrahedra [7] using trilinear interpolation on the level-set grid. Note that the extracted surface is only used for rendering and, while it does introduce typical marching cubes artifacts, does not introduce any temporal incoherence.

3.4 OpenVDB Integration

One limitation of the approach described thus far is the use of regular grids when only a narrow-band is really necessary. This limitation is especially problematic when the axis-aligned bounding box used for the grid does not fit well with the simulation domain. To address this issue, we have integrated our technique into the OpenVDB framework [25]. As an additional bonus, OpenVDB provides us with multi-threading for free.

OpenVDB is an open-source C++ library designed for storing and manipulating volumetric data. The library uses a three-level B-tree to provide fast random access while also supporting sequential access operators similar to the DT-Grid [27]. Data is stored in *buffers* and a given B-tree has a single *read*-buffer, but may have an arbitrary number of *auxiliary* buffers. Random access, which is required to lookup neighboring values when using finite differencing stencils, is only supported for the read buffer. However, auxiliary buffers may be quickly *swapped* with the read buffer by changing pointers. Individual grid cells may be labeled *on* and *off*. Cells labeled off can be quickly skipped when performing narrow band operations.

We use a total of five auxiliary buffers in our implementation. Four are used to store the constraint surfaces ϕ_{min} and ϕ_{max}, the Laplacian $\nabla^2\phi$, and the magnitude of the gradient $\|\nabla\phi\|$. A fifth buffer is used to store the result of intermediate computations (updated ϕ values). During the initial Laplacian smoothing we only use three of these buffers ($\nabla^2\phi$ and $\|\nabla\phi\|$ are not required). Fast sweeping also uses the fifth auxiliary buffer for updated ϕ values. Pseudo-code is given in Algorithm 1.

Initialization: As with the full regular grid, we must initialize fields for ϕ_{min}, ϕ_{max}, and ϕ_0, which is accomplished using OpenVDB's particle rasterization routines. The constraint surfaces are placed in auxiliary buffers and ϕ_0 is placed in the read buffer. With a regular grid we only need to specify the bounding box of the grid. In contrast, with OpenVDB, we must additionally specify the *width* of the tree at initialization. Here, "width" is not the branching factor of the tree, but rather, how wide a band we wish to have around the zero level-set—OpenVDB will not allocate grid-points outside this width. Furthermore, we can not set this width to be the same as the traditional narrow band parameter, but rather, we must choose the maximum distance from the initial surface that we would ever require a level-set value. We wish to maintain the level set within a narrow-band of three cells and the biharmonic operator requires values two cells away. Moreover the surface may move to either constraint during optimization, thus we may require values as far from the initial surface as

$$5 + \left\lceil \max\left(\frac{r_{max} - r_{init}}{h}, \frac{r_{init} - r_{min}}{h} \right) \right\rceil . \tag{7}$$

As before we perform fast sweeping to ensure that ϕ_{min}, ϕ_{max}, and ϕ_0 are distance fields.

Algorithm 1 OpenVDB Implementation

```
Initialize OpenVDB data structures
Rasterize particles onto the grid
Compute φ_min and φ_max and store them in buffers
Perform fast sweeping on φ_0, φ_min and φ_max
while (i < l_max_iter) do
    Activate cells in narrow band of width 4h
    Constrained Laplacian Update
    Swap(φ^{t+Δt}, read)
end while
while (i < b_max_iter) do
    Activate cells in narrow band of width 4h
    Compute Laplacian
    Swap(Laplacian, read)
    Activate cells in narrow band of width 3h
    Constrained Biharmonic Update
    Swap(φ^{t+Δt}, read)
    if ((i % 50) == 0) then
        Perform fast sweeping
    end if
end while
Extract zero isosurface of φ using marching cubes
```

Optimization: Each iteration of biharmonic smoothing involves six steps. First, we activate cells in the narrow band of width $4h$ by setting them *on*. Second, we compute the Laplacian of ϕ and store it in an auxiliary buffer. Third, we *swap* this auxiliary buffer with the read buffer. Fourth, we activate cells in a narrow band of width $3h$. Fifth, we compute the biharmonic operator and apply the update (including applying constraints), storing the result, $\phi^{t+\Delta t}$, in an auxiliary buffer. Finally, we swap this auxiliary buffer with the read buffer. The magnitude of the gradient is computed at the same time as the Laplacian—when ϕ is in the read buffer. For Laplacian smoothing, we skip one of the swaps and need not store the Laplacian or magnitude of the gradient. Note that, as before, the narrow band where we are actually performing computation may shift during optimization.

Surface Extraction: Conveniently, OpenVDB includes an implementation of dual contouring [19] that we use to extract the zero iso-surface of ϕ_{final}.

Redistancing: As before, we perform redistancing at initialization and periodically during smoothing. We do so using a fast sweeping method [49]. Values are initialized and *fixed* at vertices next to the interface—these values are turned *off* during fast sweeping iterations, all others are turned *on*. Then several passes are made sequentially over the grid and level-set values at vertices that are turned *on* are updated using neighbors that are closer to the interface. After each pass, the auxiliary buffer containing updated level-set values is swapped with the read buffer.

There are two differences between this algorithm and the one originally proposed by Zhao [49]. First, instead of sweeping over the grid in different orderings, we traverse the grid in parallel in the order determined by OpenVDB's sequential access routines. Second, to avoid race conditions, we use Jacobi rather than Guass-Seidel iterations. These modifications likely result in slower convergence, but allow for parallel computation and leverage OpenVDB's fast sequential access routines. Note that, it is not critical that we maintain a highly accurate signed-distance function. In practice, we have found that occasionally running eight Jacobi iterations of this approximate algorithm sufficiently maintains the signed distance property in the narrow band.

3.5 Time Integration

We have also explored two improved time integration strategies: adaptive timesteps and implicit integration.

Adaptive Timestepping: The Courant-Freidrichs-Lewy (CFL) condition states that information should travel no more than one grid-cell in a single timestep. Thus we can compute the largest allowable timestep, Δt_{max}, as

$$\Delta t_{max} = \frac{h}{v_{max}}, \tag{8}$$

where h is the grid spacing and v_{max} is the maximum speed of the level-set function (i.e. $\max(\nabla^4\phi \| \nabla\phi\|)$). While we found this approach to be stable, varying the timestep unfortunately introduces flickering in the resulting animations. An example using adaptive timestepping is included in the accompanying video. We note that, although the adaptive timestepping is not effective for generating animations, the approach could be used to warn a user if too large a fixed timestep was being used.

Implicit Integration: An alternative approach to avoiding timestep restrictions is implicit time integration [3], [11]. To do so we must solve

$$\phi^{t+\Delta t} = \phi^t - \Delta t \nabla^4 \phi^{t+\Delta t} \|\nabla\phi^{t+\Delta t}\|. \tag{9}$$

The non-linear $\|\nabla\phi^{t+\Delta t}\|$ term is problematic, but since we maintain a signed distance function through periodic redistancing we assume that $\|\nabla\phi^{t+\Delta t}\| = 1$ and drop it from the system. Combining terms we arrive at the linear system

$$(\mathbf{I} + \Delta t \nabla^4)\phi^{t+\Delta t} = \phi^t. \tag{10}$$

Because ∇^4 is a positive semi-definite operator, the system in Equation (10) is positive definite and can be solved using conjugate gradients [35]. However, there are two wrinkles to solving the constrained optimization problem. First, we must take some care in handling boundary conditions. Second, we must incorporate the inequality constraints in Equation (6). Please see the appendix for a simple didactic example.

Our solution strategy is inspired by the *active set* methods frequently used to solve linear complementarity problems (see, e.g. the book by Dostl [12]). Essentially, we repeatedly attempt to solve the linear system in Equation (10), but whenever a constraint is violated, we abort the solve and modify the system and boundary conditions. Specifically, we designate each cell as *free* or *constrained*. We initially designate all cells in a band from $3h - 5h$ as constrained to their current ϕ values; these cells form our initial boundary conditions. We initialize the right hand side by applying our operator to these constrained cells. We then attempt to solve our linear system using conjugate gradients. If during the solve a constraint is violated in the solution vector $\mathbf{x}$, we move the violating cell to the constrained set, constrain its value, recompute the right hand side, and restart conjugate gradients. While ϕ is redistanced and the constrained set is re-initialized every timestep, we note that we do not allow cells to go from the constrained set to the free set during a single timestep. This restriction simplifies the implementation and should speed convergence, but also intuitively means that once the surface hits one of the constraints it "sticks" to the constraint for the remainder of the timestep. While there may be circumstances under which cells should move from the constrained set back to the free set, we believe such situations are rare. At any rate, we did not observe any artifacts from this restriction, perhaps because of the re-initialization that occurs for each timestep.

More specifically, as before we maintain level-set values in a band of $3h$ around the zero level-set and treat the values in the range of $3h$-$5h$ as boundary conditions. Thus, we initially mark all cells in a narrow-band of $3h$ as *free* cells and cells in the range of $3h$-$5h$ are marked *constrained*. We initialize our solution vector to ϕ^t and compute the initial residual as

$$\mathbf{r} = \phi^t - (\mathbf{I} + \Delta t \nabla^4)\phi^t. \tag{11}$$

The residual and other conjugate gradients variables are only valid at the free grid cells. Consequently, conjugate gradient operations (e.g. dot products) are computed using only values at these cells by activating them and turning all other cells *off*. As mentioned above, after each conjugate gradient iteration, we enforce the constraints. If any constraints are violated, the violating cells are marked constrained and conjugate gradients is restarted with the current solution and the new right hand side. The residual is re-computed by applying our operator to the current solution in free cells and the constrained values/boundary conditions at constrained grid cells. We still break our time integration into several (e.g. 5) timesteps to allow for periodic redistancing. The method is summarized in Algorithm 2.

The OpenVDB library does not include an implementation of conjugate gradients, but the algorithm is straightforward to implement in the OpenVDB framework. The method requires the storage of five vectors: the (current) solution ($\mathbf{x}$), the residual ($\mathbf{r}$), the (conjugate) search direction ($\mathbf{d}$), a flag determining whether a cell is free or constrained ($\mathbf{c}$), and a temporary vector that stores the linear operator applied to the search direction ($\mathbf{q} = \mathbf{Ad}$). We store each of these in auxiliary buffers. In addition to applying the linear operator in Equation (10) the method requires several basic linear algebra operations: dot products, vector scaling, and vector addition. The dot products use `PARALLEL_REDUCE` calls while the other operations allow `PARALLEL_FOR` calls. These linear algebra operations account for roughly 20% of total runtime (see "CG Overhead" in Tables 1 and 3 and Figure 19).

A couple of minor notes are in order. First, care must be taken when activating cells. We must always apply our

Algorithm 2 Implicit Time Integration

1: **for** $(i = 0)$ **to** $(i =$ ntimesteps$)$ **do**
2: Copy ϕ to auxiliary buffer in band of $5h$
3: Initialize $\mathbf{x} = \phi$ in band of $5h$
4: **while** (constraints violated) **do**
5: Initialize $\mathbf{r} = \phi - \mathbf{Ax}$ in band of $3h$, 0 elsewhere
6: Initialize $\mathbf{d} = \mathbf{r}$ in band of $3h$, 0 elsewhere
7: $\delta_{new} = r^T r$
8: **if** $(i == 0)$ **then**
9: $tol = \varepsilon^2 \delta_{new}$
10: **end if**
11: **while** (true) **do**
12: Swap($\mathbf{d}$, read)
13: Apply operator ($\mathbf{q} = \mathbf{Ad}$)
14: Activate free cells
15: $\alpha = \delta_{new} / (\mathbf{d}^T \mathbf{q})$
16: $\mathbf{x} = \mathbf{x} + \alpha \mathbf{d}$
17: $\mathbf{r} = \mathbf{r} - \alpha \mathbf{q}$
18: Apply constraints, mark constrained cells
19: **if** (constraints violated) **then**
20: break;
21: **end if**
22: $\delta_{old} = \delta_{new}$
23: $\delta_{new} = \mathbf{r}^T \mathbf{r}$
24: $\beta = \delta_{new} / \delta_{old}$
25: $\mathbf{d} = \mathbf{r} + \beta \mathbf{d}$
26: **if** $(\delta_{new} < \varepsilon^2 \delta_0) \,||\, (\delta_{new} < tol)$ **then**
27: break;
28: **end if**
29: **end while**
30: **if** $(\delta_{new} < \varepsilon^2 \delta_0) \,||\, (\delta_{new} < tol)$ **then**
31: break;
32: **end if**
33: **end while**
34: read = $\mathbf{x}$ in band of $3h$
35: **if** $((i \,\%$ redistanceFrequency$) == 0)$ **then**
36: Perform fast sweeping
37: **end if**
38: **end for**

operator to values in a band of width $5h$, but it is important that all the conjugate gradient calculations apply only to the free cells. For example, the residual should be computed only at free cells and should be treated as zero at constrained cells. Second, during application of the operator, we must first swap the read buffer (containing $\mathbf{d}$) with the auxiliary buffer allocated for $\mathbf{d}$, before swapping the read buffer with the newly computed Laplacian (between lines 12 and 13 in Algorithm 1). Finally, all of the linear algebra operations use sequential access and, therefore, do not require any swapping with the read buffer. That is, the auxiliary buffers for $\mathbf{x}$, $\mathbf{r}$, etc. are accessed directly. The read buffer is only required for applying the finite difference stencils in the linear operator.

3.6 Obstacle handling

Nearly all particle animations involve interactions with obstacles. While a simulation may ensure that particles do not penetrate obstacles, the skinned animation may not preserve this property. Additionally, the skinning process may introduce small *voids* between the the surface and obstacles (see Figure 2). Fortunately, level-set methods are excellent for performing Boolean operations and addressing these artifacts is straightforward. For any obstacles, we can build a level set, ϕ_o. Then, to address intersections, we introduce an additional constraint,

$$\phi(i,j,k) = \max(\phi(i,j,k), -\phi_o(i,j,k)). \tag{12}$$

Dealing with voids is slightly trickier. In this case, we must choose a threshold for the size of a void we wish to remove. Choosing too large a threshold will result in temporal incoherence when the threshold is suddenly satisfied. Too small a threshold will leave small voids in place. At any rate, if a point is outside both the surface and the obstacle, but the sum of the level-set values is small, we wish to replace ϕ with $-\phi_o$. Specifically,

if $\phi(i,j,k) > 0$ **and** $\phi_o(i,j,k) > 0$ **and**
$\phi(i,j,k) + \phi_o(i,j,k) < \varepsilon$ **then**
$\phi(i,j,k) = -\phi_o(i,j,k)$
end if

These constraints are handled in the same manner as the union-of-spheres constraints.

3.7 Parameters

At first it may seem that there are many parameters to our approach: r_{min}, r_{max}, the number of smoothing passes, frequency of fast sweeping, timestep, and grid spacing. While hand-tuning these parameters can produce slightly improved results and reduce running times, good default values can be obtained by requiring the user to specify only the grid-spacing, h, which is related to the inter-particle spacing and the desired level of detail in the resulting surfaces. The grid spacing can be quickly found by viewing ϕ_0 and ensuring that there is the desired amount of overlap between particles. Given h, we set

$$r_{min} = \sqrt{3}h. \tag{13}$$

This choice guarantees that every particle rasterizes to eight grid cells and ensures that the particle is well represented. The ratio between r_{min} and r_{max} provides a tradeoff between surface smoothness and faithfulness to the underlying particles

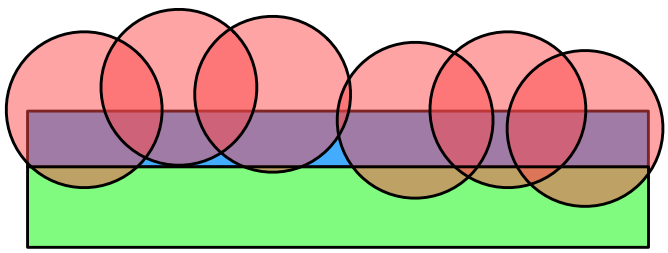

Fig. 2. Depending on the details of the underlying simulation's handling of particle-obstacle interactions, there may be voids (blue) and/or intersections (orange/brown) between our surface and the obstacle geometry.

(see Figure 3). Larger ratios generate smoother surfaces, while smaller ratios more faithfully represent the particles. We have found that a ratio of 2 works well for many examples.

For the explicit time integration scheme, we default to 500 passes of biharmonic smoothing with fast sweeping performed every 50 passes. The timestep defaults to

$$\Delta t = 0.01h^4, \tag{14}$$

though this relation is not perfect and we anticipate that some examples will require a smaller timestep. Implicit integration admits a larger timestep; we use

$$\Delta t = 20h^4 \tag{15}$$

in our examples. As mentioned above we still take several timesteps, 5 in our examples, to allow for redistancing. Note that this results in longer integration times, and smoother surfaces, than the default settings for explicit integration. We found that even larger timesteps remained stable and did not lead to artifacts. But these longer timesteps did require more conjugate gradient iterations and hence longer runtimes without producing significantly better results. We found that this timestep achieved a good balance, but we did not perform exhaustive experiments. We further note that there is a relationship between integration time and the ratio, r_{max}/r_{min}. In short, longer integration times require smaller ratios to remain faithful and avoid temporal artifacts, while shorter integration times require larger ratios to achieve smoothness. The implicit integration scheme uses three additional parameters: relative and absolute thresholds for convergence of conjugate gradients and a cap on the number of conjugate gradient iterations. We use default values of 1e-4, 1e-6, and 1500 respectively. All the examples in Figure 14 are generated using these parameters.

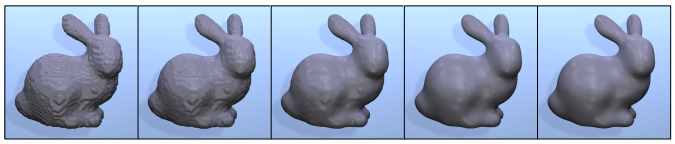

Fig. 3. The effect of changing the ratio, r_{max}/r_{min}, from left to right, 1, 1.25, 1.5, 2, and 4.

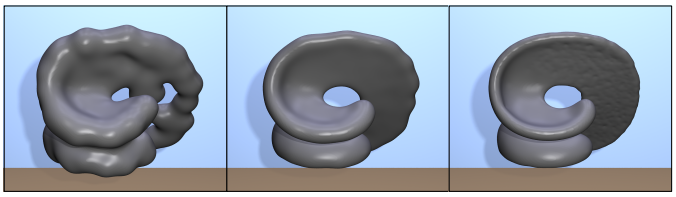

Fig. 4. The "Enright" test [13]. The leftmost image is a very low resolution example (500 particles); the middle image has an intermediate resolution (5000 particles); and the rightmost example has higher resolution (50,000 particles). With very coarse particles sets, the underlying discretization becomes apparent, but as the videos demonstrate, our technique maintains temporal coherence. The underlying analytic velocity field results in a non-uniform particle sampling that can be observed in the rightmost image. Of course, relaxing the faithfulness criterion will result in a smoother surface. To avoid marching cubes artifacts the low and intermediate resolution examples used the same grid spacing (but different r_{min} and r_{max}) as the higher resolution example.

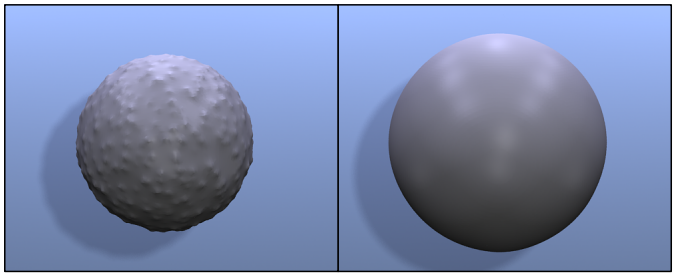

Fig. 5. Laplacian smoothing (left) shrinks the sphere until it starts interacting with the constraints, while our biharmonic smoothing (right) converges to a sphere.

4 Results and Discussion

We have four implementations of our particle skinning method: BasicExplicit, which uses regular grids and explicit integration; BasicImplicit, which uses regular grids and implicit integration; OpenVDBExplicit, which uses explicit integration within the OpenVDB framework; and OpenVDBImplicit, which employs implicit integration within the OpenVDB framework. We have tested all four implementations with data from a variety of particle-based animation systems and several analytical tests. All the images demonstrating our method in this paper were generated using OpenVDBImplicit unless otherwise noted. In the accompanying video, we provide animations of these, and additional, results as well as comparisons between the different implementations. These examples demonstrate that our method generates temporally coherent, smooth surfaces while preserving much of the richness of the underlying particle motion. In this section, we first discuss the versatility of our method across a range of examples before shifting to performance analysis.

4.1 Examples

Figure 4 shows the result of the "Enright" test [13]. In this example, we have tested our method with different resolutions to show that even for very coarse particle sets and grids, temporal coherence is maintained. The supplementary material also shows a Zalesak notched sphere example, which also does not flicker. We note that in these tests the particles are passively advected through the analytic flow field and, consequently, we would not expect to see the same sort of distortions commonly found in other surface tracking methods.

In Figure 5, we show the advantage of biharmonic smoothing over the simpler Laplacian smoothing that is commonly employed. At first Laplacian smoothing quickly smoothes the surface, but additional iterations shrink the surface such that it starts interacting with the constraints. Volume correction techniques [11] could be employed to address the shrinkage, but these techniques lead to other artifacts (e.g. volume moving from one region of the fluid to another). Alternatively, a limited

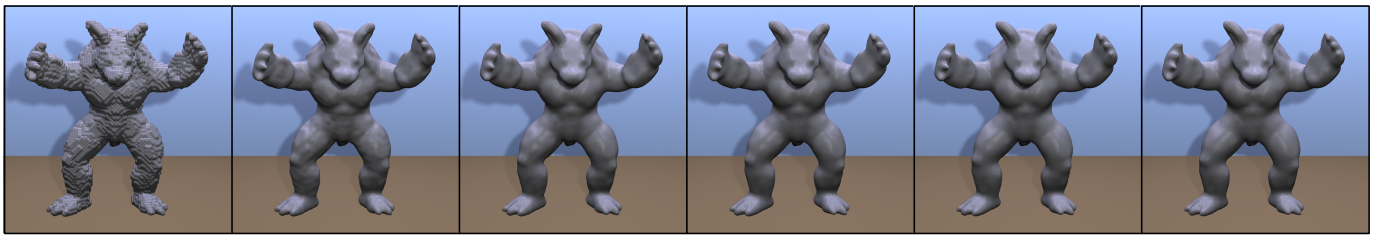

Fig. 6. The surface after (from left to right) 0, 2000, 4000, 6000, 8000 and 10000 timesteps using OpenVDBExplicit.

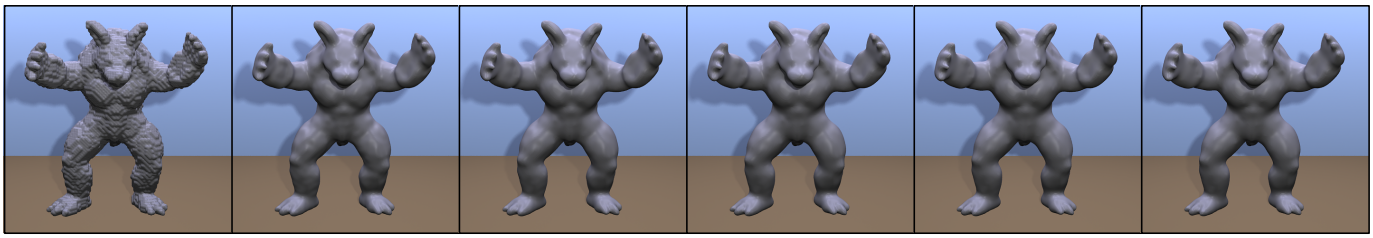

Fig. 7. The surface after (from left to right) 0, 1, 2, 3, 4 and 5 timesteps using OpenVDBImplicit.

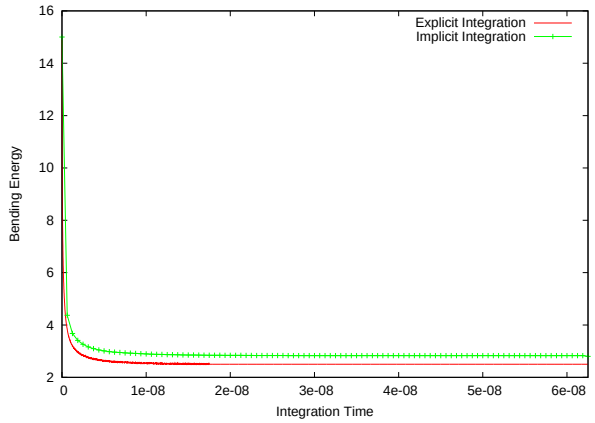

Fig. 8. Bending energy as a function of integration time for both OpenVDBExplicit (see Figure 6) and OpenVDBImplicit (see Figure 7). This graph indicates that the thin-plate energy is a good proxy for bending energy and that the two approaches have very similar behavior.

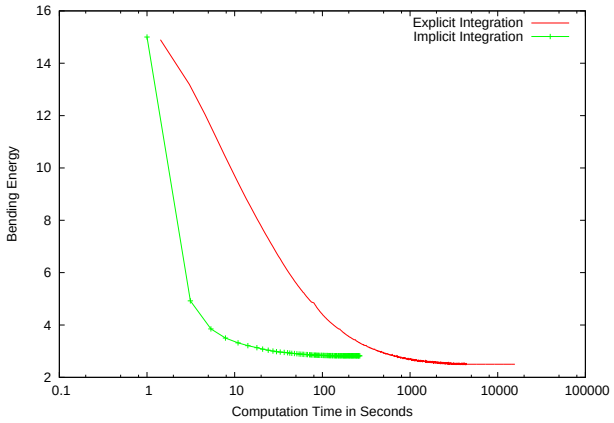

Fig. 9. Bending energy as a function of computation time (in seconds) for both OpenVDBExplicit (see Figure 6) OpenVDBImplicit (see Figure 7). Implicit integration performs substantially better.

number of smoothing passes could be applied, however it is difficult to know, a priori, how many passes to apply. Biharmonic smoothing, on the other hand, converges to a sphere without shrinking the surface and applying additional smoothing passes produces no ill effects. Figures 6 and 7 show the results after various numbers of explicit and implicit timesteps, respectively, and Figures 8 and 9 show the bending energy (measured as the integral of mean curvature over the output polygonal surface) as a function of timestep and computation time. In particular, Figure 9 demonstrates the computational superiority of our implicit integration scheme.

We also demonstrate one of the primary limitations of our approach in Figure 10. For the example on the left, we created a scalar field that is a signed distance function for $x > 0$, but is a function that rapidly increases away

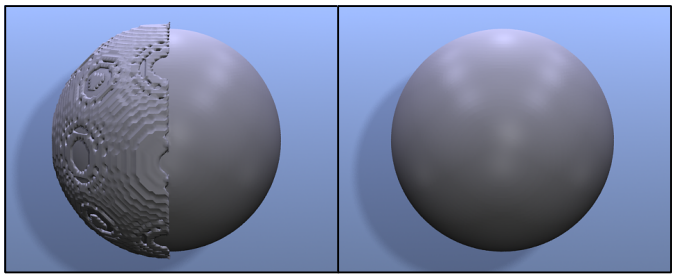

Fig. 10. When ϕ is not a signed-distance function (left), smoothing fails. With a signed distance function (right), biharmonic smoothing converges nicely.

from the surface for $x < 0$. While smoothing by the intrinsic Laplacian of curvature [9] could theoretically handle this case because such motion is independent of the parameterization,

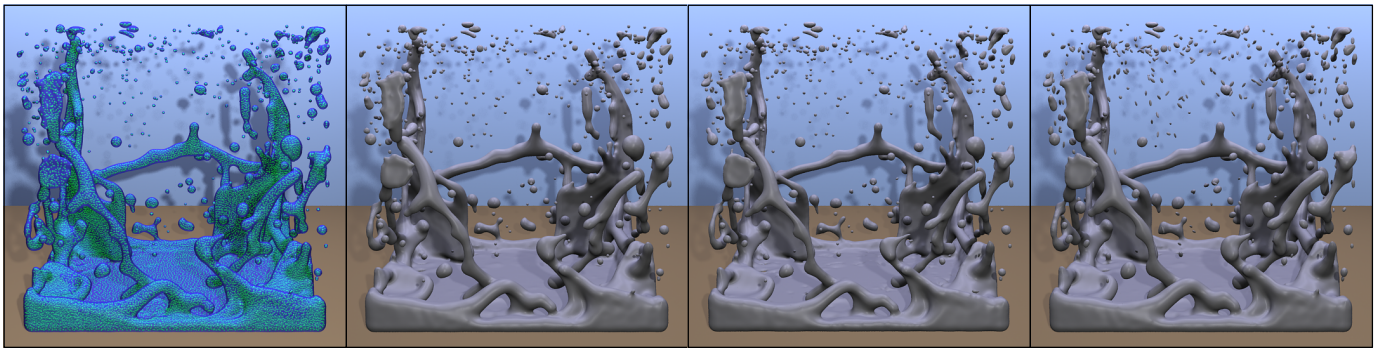

Fig. 11. From left: a rendering of the particles inside the generated surface, isotropic constraints, neighbor-based anisotropy and velocity-based anisotropy. All surfaces in this figure were generated using the BasicExplicit implementation.

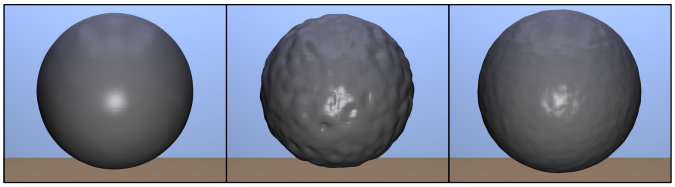

Fig. 12. A comparison with the method of Yu and Turk [47]. A sphere is sampled (uniformly at random) with particles and given as input. Our approach (left) nearly recreates the sphere with 50,000 particles. The method of Yu and Turk [47], produces a fairly lumpy surface at 50,000 particles (center), but does much better with 1,000,000 particles (right), though some lumpiness is still present. These results imply that our approach may be especially well-suited for low particle counts. The leftmost surface was generated using the BasicExplicit implementation.

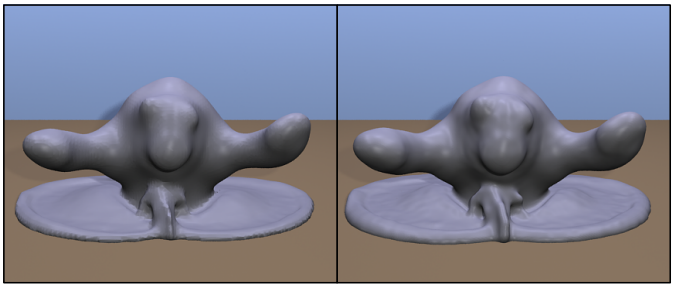

Fig. 13. A comparison of Adams et al [1] approach (left) with ours (right) on an example with variably sized particles.

the bad parameterization interferes with our linear operator and causes the smoothing to fail. Fortunately, this limitation can be addressed by occasionally applying a fast sweeping method to re-establish the signed-distance property. Another limitation, not so easily addressed, is that our method has difficulty generating very thin surfaces, even at high grid-resolutions. While for a mesh-based approach, the topology of the surface is fixed before smoothing begins, in a level-set approach the topology is free to change throughout the smoothing process. Thus, with our approach if r_{min} is set small enough that spheres centered at the particles do not overlap, the surface can break apart. With a mesh-based approach if pieces of surface are initially connected, they may get arbitrarily thin without breaking apart.

We also provide comparisons with previous approaches in the accompanying video. In particular, we compare with the method of Adams and colleagues [1] in the falling armadillo example; and with the mesh-based method of Williams [44] in the example with the smiley face board. In both these cases it is clear that our approach maintains better temporal coherence. In Figure 12, we show a comparison with the method of Yu and Turk [47]. Our surface is quite smooth at modest particle counts, while theirs retains some lumpy features even with many more particles. An additional comparison with their double dam break is included in the supplementary material. Note that the double dam break uses the BasicExplicit implementation.

In Figure 13, we demonstrate our method's ability to handle variably sized particles. In this case r_{min} and r_{max} vary per particle and are a function of both the particle "radius" given by the simulator and the level-set grid-spacing. In Figure 11, we demonstrate our method's ability to handle anisotropic particles. The anisotropy can be determined either using a particle's local neighborhood as in Yu and Turk [47] or using velocity stretching where $\mathbf{T}$ in Equation (1) is given by

$$\mathbf{T} = \begin{pmatrix} \mathbf{v} & \mathbf{t}_0 & \mathbf{t}_1 \end{pmatrix} \begin{pmatrix} \frac{1}{(1+s)^2} & 0 & 0 \\ 0 & 1+s & 0 \\ 0 & 0 & 1+s \end{pmatrix} \begin{pmatrix} \mathbf{v} \\ \mathbf{t}_0 \\ \mathbf{t}_1 \end{pmatrix}, \tag{16}$$

where $\mathbf{v}$ is the normalized particle velocity, $\mathbf{t}_0$ and $\mathbf{t}_1$ are orthogonal to $\mathbf{v}$ and to each other, and s is the particle's (scaled) speed. This transformation has the effect of reducing distances in the velocity direction (and increasing distances in directions tangent to the velocity), thereby stretching particles in the direction of movement. These two approaches to anisotropy achieve different artistic effects and we do not advocate one over the other. Note that the warping induced by $\mathbf{T}$ in Equation (1) occurs before redistancing ensures that the various scalar fields are signed distance functions. In this way, our contribution is complementary to the anisotropic kernels advocated by Yu and Turk [47].

To demonstrate our obstacle handling, we sample the surface of a tetrahedron with particles and apply our approach (see Figure 15). The surface mesh of the tetrahedron is converted to a level set, ϕ_o, and treated as an obstacle as we apply intersection and void removal. Interestingly, intersection

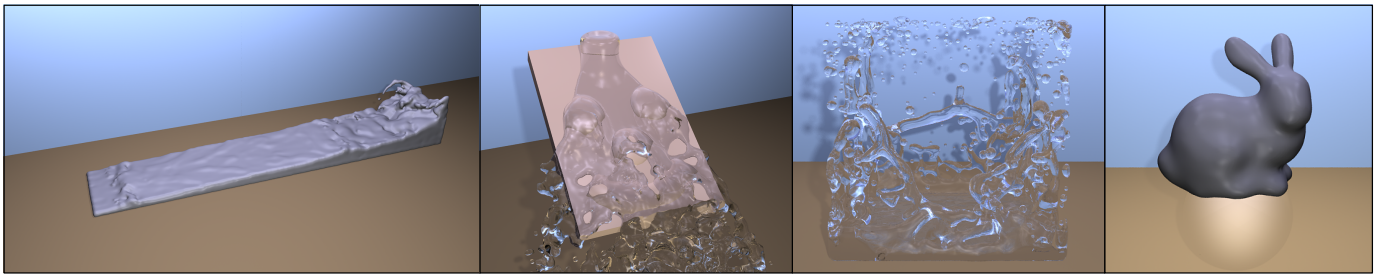

Fig. 14. We have applied our particle skinning technique to data from a variety of particle simulation systems. From left to right: A fluid-implicit particle (FLIP) simulation [50] (217K particles, grid resolution $= 693 \times 93 \times 130$), a simulation using the method of Sin et al. [37] (38K particles, grid resolution $= 219 \times 212 \times 305$), a smoothed particle hydrodynamics simulation [1] (52K particles, grid resolution $= 237 \times 235 \times 220$), and an elasto-plastic smoothed particle hydrodynamics simulation [16] (40K particles, grid resolution $= 174 \times 110 \times 215$).

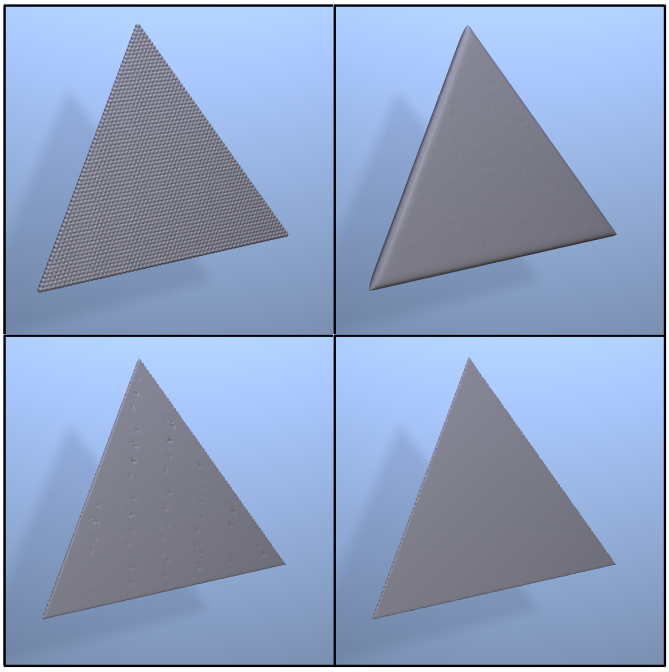

Fig. 15. This figure demonstrates our obstacle handling. Top-left: the surface of a tetrahedron is sampled with particles. Top-right: constrained biharmonic smoothing, no obstacle handling. Bottom-left: constrained biharmonic smoothing with intersection removal. Bottom-right: constrained biharmonic smoothing with intersection and void removal. All surfaces in this figure were generated using the BasicExplicit implementation.

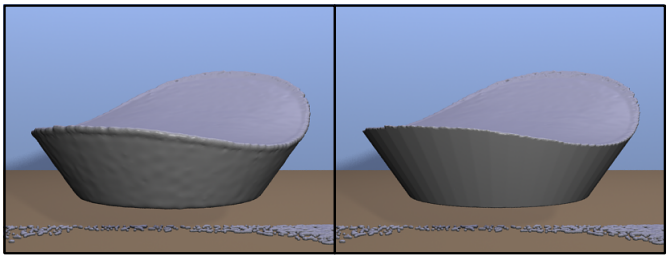

Fig. 16. The result of applying our obstacle handling to simulation data. Left: no obstacle handling. Right: intersection and void removal. All surfaces in this figure were generated using the BasicExplicit implementation.

Example	Left	Center-Left	Center-Right	Right
BasicExplicit				
Total Time	2339.1	2319.6	2504.7	946.2
Rasterization	11.5	1.5	3.7	0.3
Fastsweeping	18.4	24.6	35.8	10.2
Apply Operator	1486.1	1430.2	1491.5	542.9
Compute $\|\nabla\phi\|$	468.7	466.2	497.4	203.6
Memory in GB	0.79	0.53	0.7	0.32
OpenVDBImplicit				
Total Time	79.5	74.9	72.8	14.5
Rasterization	11.9	0.7	2.7	0.8
Fastsweeping	3.4	1.5	1.9	1.1
Apply Operator	30.9	37.6	33.1	6.1
CG overhead	17.2	18.9	18.8	2.9
Activate	13.7	13.4	13.2	3
Apply Constraints	2.1	2.3	2.3	0.5
Memory in GB	0.82	0.26	0.46	0.18
CG iterations	969	1940	1301	758
CPU Usage	1113	1158	1124	1110

TABLE 1
Performance statistics for the examples in Figure 14 with BasicExplicit and OpenVDBImplicit implementations. Timings are given in seconds. *CG Overhead* refers to the various linear algebra operations involved in conjugate gradients and *Activate* refers to activating/deactivating cells.

removal alone introduces artifacts that are more objectionable than performing no obstacle handling. However, with both intersection and void removal the surface is nice and sharp. In Figure 16 we show the result of applying our obstacle handling to simulation data.

4.2 Performance Analysis

Now, we shift our attention to performance analysis of our particle skinning method, using the animation examples in Figure 14 and the simple static examples in Figure 17. All statistics were recorded on a Ubuntu 14.04 linux server with dual Intel Xeon X5650 6-core processors (12 cores total) running at 2.8 GHz and 48 GB of memory. All reported running times except those in Table 2 ignore file I/O.

In Table 1 we compare timing results for the BasicExplict and OpenVDBImplicit implementations for the examples in Figure 14. All statistics are averages over the frames in the

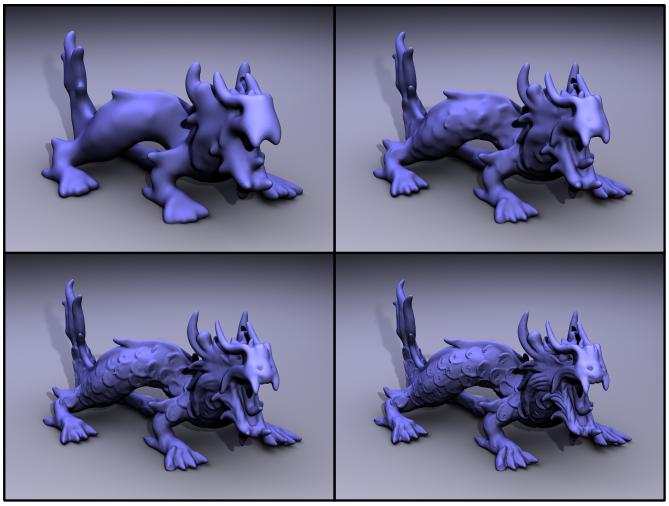

Fig. 17. Our algorithm is applied to synthetic particle sets that approximate a dragon model. Particles were sampled from regular grids and rejected if they fell outside the model. The grid resolutions varied in a geometric progression from 0.8 (upper-left) to 0.1 (lower-right). See Table 3 for performance statistics.

accompanying animations. These examples were generated using 15 Laplacian smoothing passes followed by 10000 explicit timesteps or 5 implicit timesteps. Letting n be the number of timesteps, the size of the timesteps, Δt, was chosen so that the total integration time, $T = n\Delta t$, was the same regardless of the integration scheme. Implicit integration and parallelization with OpenVDB clearly pay off, resulting in more than an order of magnitude better performance. Also note that with 12 cores available the CPU usage is between 1110% and 1158%, which indicates good resource usage. Except for one example, which is well approximated by an axis-aligned bounding box, we also see significant improvements in memory usage with the OpenVDB implementation.

We also examined how well our algorithm and its various steps parallelize and scale. To do so, we created synthetic particle sets at various resolutions that represent the dragon model in Figure 17. The particle sets where created by sampling from a regular grid and including samples that fall inside the model while rejecting exterior samples. We then applied our algorithm while varying the number of threads available to OpenVDB. Figure 18 shows the total time required to generate the surface in Figure 17 (upper-right) as the number of threads increases. The baseline (in green) corresponds to OpenVDB's automatic thread management. Figure 19 gives a breakdown of the various steps of our algorithm. Clearly we extract the most parallelism from the application of our operator, but even here the returns quickly diminish.

We also compare to more naïve parallelism, where a number of separate jobs are run at once. To do so, we ran a "large workload" consisting of 120 total jobs and vary the number of jobs running simultaneously and the number of threads allocated to each job such that 12 threads are always running. The results are shown in Table 2. Somewhat surprisingly, running 12 jobs at once with 1 thread each achieved the fastest result. While we did observe a slight slowdown when running

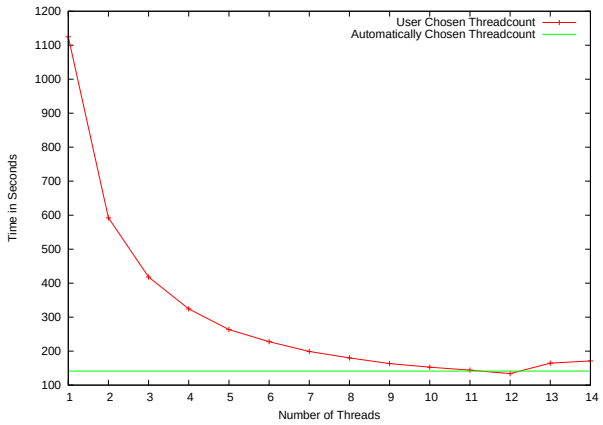

Fig. 18. Total time to generate the surface in Figure 17 (upper-right) while varying the number threads.

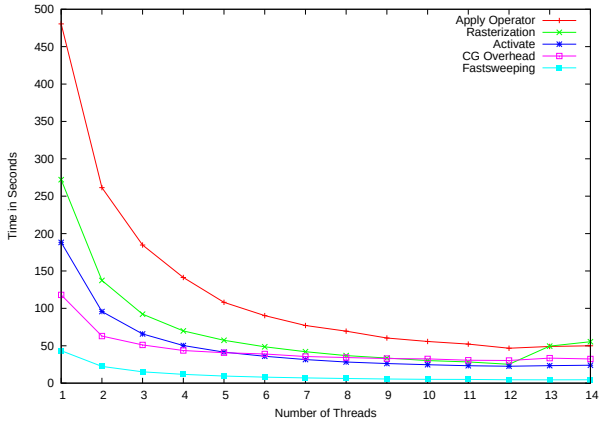

Fig. 19. Breakdown of the various components of our algorithm for Figure 18.

# processes	# threads per process	Total Time
12	1	03:32
6	2	03:36
4	3	03:44
3	4	03:51
2	6	04:09
1	12	04:43

TABLE 2
Time (in hh:mm) to compute the surface in Figure 17 (upper-right) a total of 12 times while varying the number of simultaneous processes and number of threads per process.

12 single-threaded jobs compared to one single-threaded job, it seems there was only minor contention for the cache. There is a notable jump between 2 and 1 simultaneous jobs, probably due to the higher cost of communication between processors than between cores on the same processor. Of course, very large jobs, such as the surface in Figure 17 (lower-right), are constrained by the total memory available—at high grid resolutions it is not possible to run a large number of jobs.

To understand how our algorithm scales, we present performance statistics for the four examples in Figure 17 in Table 3. Our approach scales well. For each 2^3 increase in grid reso-

Grid Spacing (h)	0.8	0.4	0.2	0.1
Total Time	25.5	141.2	971.2	6450.2
Rasterization	3.7	25.3	196.1	1551.9
Fastsweeping	1.8	4.6	22.9	147.9
Apply Operator	9.1	46.3	309.7	1641.9
CG overhead	5.5	36.3	256.1	1838.7
Activate	5.24	24.5	160.1	1133.1
Apply Constraints	0.9	4.1	22.2	115.1
Number of particles	603 K	4.82 M	38.6 M	309 M
Memory in GB	0.35	1.1	4.4	28
CG iterations	938	1181	1965	2144
CPU usage	1089	1125	1138	1135

TABLE 3
Performance statistics for the examples in Figure 17 with varying resolution. Timings are given in seconds. We use the same naming conventions as in Table 1.

lution, the running time increases by a factor of between 5.5 and 7. The largest increase in running time occurs between columns 2 and 3, which also corresponds to a dramatic jump in the number of conjugate gradient iterations—likely indicating more interaction with the constraint surfaces. Note that this sublinear scaling is expected because some operations, like rasterization, vary with volume (i.e. n^3); others, like the level-set operations, varying more more closely with the surface area (i.e. n^2). A graph of the "speedup" due to parallelization, measured as the ratio of sequential execution to parallel execution, is shown in Figure 20. The speedup remains relatively constant across grid-resolutions, with slightly higher speedups achieved at higher resolutions.

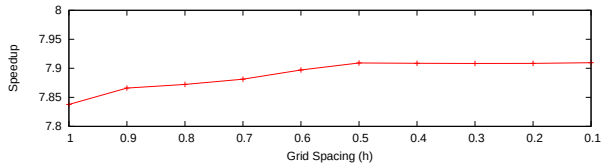

Fig. 20. Speedup, computed as ratio of sequential runtime to parallel runtime, as grid spacing decreases. The speedup remains relatively constant across grid-resolutions, with slightly higher speedups achieved at higher resolutions.

We have presented a method for skinning particle animations that processes each frame independently *and* maintains the temporal coherence of the particle data. Our method is fast and easy to implement, integrates easily with OpenVDB, admits implicit integration, can handle boundaries and anisotropic or variable-sized particles, and generates appealing, smooth surfaces. We fully expect our method to find its way into many production toolboxes where it will complement the vast array of particle skinning tools currently available.

Acknowledgments

We wish to thank Bart Adams for making his SPH source code available, Funshing Sin and Dan Gerszewski for sharing their particle data, and Jihun Yu for answering our questions and sharing his data and source code. We would also like to thank the Peter-Pike Sloan, Ladislav Kavan, and the anonymous reviewers for their helpful feedback. This work was funded in part by NSF awards CNS-0855167, IIS-1045032, IIS-1249756, IIS-1314896, and gifts from Disney Interactive Research and Adobe Systems.

References

[1] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. Adaptively sampled particle fluids. *ACM Trans. Graph.*, 26(3):48, 2007.
[2] J. Andreas Bærentzen and Niels Jørgen Christensen. Interactive modelling of shapes using the level-set method. *International Journal of Shape Modelling*, 8(2):79–97, 2002.
[3] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 43–54, 1998.
[4] Adam W. Bargteil, Tolga G. Goktekin, James F. O'Brien, and John A. Strain. A semi-lagrangian contouring method for fluid simulation. *ACM Trans. Graph.*, 25(1), 2006.
[5] Haimasree Bhattacharya, Yue Gao, and Adam W. Bargteil. A level-set method for skinning animated particle data. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Aug 2011.
[6] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, 1982.
[7] Jules Bloomenthal. An implicit surface polygonizer. In *Graphics Gems IV*, pages 324–349. Academic Press Professional, Inc., 1994.
[8] Alexander I. Bobenko and Peter Schröder. Discrete willmore flow. In *Symposium on Geometry Processing*, pages 101–110, 2005.
[9] David L. Chopp and J. A. Sethian. Motion by intrinsic laplacian of curvature. *Interfaces and Free Boundaries*, 1:1–18, 1999.
[10] Mathieu Desbrun and Marie-Paule Gascuel. Active implicit surface for animation. In *Graphics Interface*, pages 143–150, 1998.
[11] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *The Proceedings of ACM SIGGRAPH*, pages 317–324, 1999.
[12] Zdeněk Dostál. *Optimal Quadratic Programming Algorithms: With Applications to Variational Inequalities*. Springer Publishing Company, Incorporated, 1st edition, 2009.
[13] 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.
[14] Douglas P. Enright, Stephen R. Marschner, and Ronald P. Fedkiw. Animation and rendering of complex water surfaces. *ACM Trans. Graph.*, 21(3):736–744, 2002.
[15] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *the Proceedings of ACM SIGGRAPH 2001*, pages 23–30, 2001.
[16] Dan Gerszewski, Haimasree Bhattacharya, and Adam W. Bargteil. A point-based method for animating elastoplastic solids. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Aug 2009.
[17] Simone E. Hieber and Petros Koumoutsakos. A lagrangian particle level set method. *J. Comput. Phys.*, 210(1):342–367, 2005.
[18] Klaus Hildebrandt and Konrad Polthier. Constraint-based fairing of surface meshes. In *The Proceedings of the Eurographics symposium on Geometry processing*, pages 203–212, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.
[19] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. *ACM Trans. Graph.*, 21(3):339–346, July 2002.
[20] Leif Kobbelt. Discrete fairing. In *The Proceedings of the IMA Conference on the Mathematics of Surfaces*, pages 101–131, 1996.
[21] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *The Proceedings of ACM SIGGRAPH*, pages 163–169, 1987.
[22] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *The Proceedings of the Symposium on Computer Animation*, pages 154–159, 2003.
[23] Matthias Müller. Fast and robust tracking of fluid surfaces. In *The Proceedings of the Symposium on Computer Animation*, pages 237–245, New York, NY, USA, 2009. ACM.
[24] Ken Museth. A flexible image processing approach to the surfacing of particle-based fluid animation. In *Mathematical Progress in Expressive Image Synthesis*, page 4. Springer, 2013.
[25] Ken Museth. Vdb: High-resolution sparse volumes with dynamic topology. *ACM Trans. Graph.*, 32(3):27:1–27:22, July 2013.

[26] Ken Museth, Michael Clive, and Nafees Bin Zafar. Blobtacular: surfacing particle system in "pirates of the caribbean 3". In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 20, New York, NY, USA, 2007. ACM.

[27] Michael B. Nielsen and Ken Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *J. Sci. Comput.*, 26(3):261–299, 2006.

[28] OpenVDB. www.openvdb.org. Accessed: 2014-06-23.

[29] Stanley Osher and Ronald Fedkiw. *The Level Set Method and Dynamic Implicit Surfaces*. Springer-Verlag, New York, 2003.

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

[31] W. T. Reeves. Particle systems—a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, 2(2):91–108, 1983.

[32] Robert Schneider, Leif Kobbelt, and Hans-Peter Seidel. Improved bi-laplacian mesh fairing. In *Mathematical Methods for Curves and Surfaces*, pages 445–454. Vanderbilt University, 2001.

[33] James Albert Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge Monograph on Applied and Computational Mathematics. Cambridge University Press, Cambridge, U.K., 2^{nd} edition, 1999.

[34] Chen Shen and Apurva Shah. Extracting and parametrizing temporally coherent surfaces from particles. In *The Proccedings of ACM SIGGRAPH (sketches)*, page 66, 2007.

[35] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994.

[36] Karl Sims. Particle animation and rendering using data parallel computation. In *The Proceedings of ACM SIGGRAPH*, pages 405–413, New York, NY, USA, 1990. ACM.

[37] Funshing Sin, Adam W. Bargteil, and Jessica K. Hodgins. A point-based method for animating incompressible flow. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Aug 2009.

[38] Barbara Solenthaler, Jürg Schläfli, and Renato Pajarola. A unified particle model for fluid-solid interactions. *Journal of Visualization and Computer Animation*, 18(1):69–82, 2007.

[39] Tolga Tasdizen, Ross Whitaker, Paul Burchard, and Stanley Osher. Geometric surface processing via normal maps. *ACM Trans. Graph.*, 22(4):1012–1033, 2003.

[40] Gabriel Taubin. A signal processing approach to fair surface design. In *The Proceedings of ACM SIGGRAPH*, pages 351–358, New York, NY, USA, 1995. ACM.

[41] Max Wardetzky, Miklós Bergou, David Harmon, Denis Zorin, and Eitan Grinspun. Discrete quadratic curvature energies. *Comput. Aided Geom. Des.*, 24(8-9):499–518, 2007.

[42] Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. Discrete laplace operators: no free lunch. In *The Proceedings of Eurographics symposium on Geometry processing*, pages 33–37, 2007.

[43] William Welch and Andrew Witkin. Free-form shape design using triangulated surfaces. In *The Proceedings of ACM SIGGRAPH*, pages 247–256, New York, NY, USA, 1994. ACM.

[44] Brent Williams. Fluid surface reconstruction from particles. Master's thesis, University of British Columbia, 2008.

[45] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Deforming meshes that split and merge. *ACM Trans. Graph.*, 28(3):1–10, 2009.

[46] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Physics-inspired topology changes for thin fluid features. *ACM Trans. Graph.*, 29(4):1–8, 2010.

[47] Jihun Yu and Greg Turk. Reconstructing surfaces of particle-based fluids using anisotropic kernels. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 217–225, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[48] H. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *IEEE Workshop on Variational and Level Set Methods*, pages 194–202, 2001.

[49] Hongkai Zhao. A fast sweeping method for eikonal equations. *Mathematics of Computation*, 74:603–627, 2004.

[50] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM Trans. Graph.*, 24(3):965–972, 2005.

APPENDIX

In this appendix we examine a simple didactic constrained optimization problem with boundary conditions. Consider the system

$$\begin{pmatrix} -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ x \\ y \\ 7 \end{pmatrix} = \mathbf{0}, \tag{17}$$

where x and y are unknowns and 1 and 7 are boundary conditions. Clearly the solution is $x = 3$ and $y = 5$. The equivalent square system is

$$\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 7 \end{pmatrix} \tag{18}$$

Thus the right-hand side can be computed by applying the non-square operator to the boundary conditions and multiplying by -1.

Now, let us assume that y is constrained to be less than or equal to 4. In this case, when y exceeds 4 during the solve we must apply this constraint. Doing so results in the smaller system

$$(2)(x) = (5) \tag{19}$$

as the constrained value at y is now treated in the same way as the boundary conditions.

Haimasree Bhattacharya is a sixth year PhD student in computer science at the University of Utah. Her primary research interests lie in the area of physics-based animation. She spent summer 2010 as a research intern at PDI/DreamWorks and summer 2011 as a research intetn at Weta Digital.

Yue Gao received the PhD degree from the Department of Computer Science and Technology of Tsinghua University in 2011. His research interests are in physical simulation and rendering. He was a visiting student at University of Utah in 2009.

Adam W. Bargteil received the PhD degree in computer science from the University of California, Berkeley and spent two years as a postdoctoral fellow in the Robotics Institute at Carnegie Mellon University. He is an assistant professor at the University of Utah. His primary research interests lie in the area of physics-based animation. He is currently an associate editor of the ACM Transactions on Graphics. He was program cochair of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation in 2011 and has served on several program committees, including ACM SIGGRAPH, ACM SIGGRAPH Asia, ACM/EG SCA, and Pacific Graphics. From 2005 to 2007, he was a consultant at PDI/DreamWorks, developing fluid simulation tools that were used in “Shrek the Third” and “Bee Movie.”