Deformation Embedding for Point-Based Elastoplastic Simulation

Ben Jones, Stephen Ward, Ashok Jallepalli, Joseph Perenia, and Adam W. Bargteil
University of Utah

We present a straightforward, easy-to-implement, point-based approach for animating elastoplastic materials. The core idea of our approach is the introduction of *embedded space*—the least-squares best fit of the material's rest state into three dimensions. Nearest neighbor queries in the embedded space efficiently update particle neighborhoods to account for plastic flow. These queries are simpler and more efficient than remeshing strategies employed in mesh-based finite element methods. We also introduce a new estimate for the volume of a particle, allowing particle masses to vary spatially and temporally with fixed density. Our approach can handle simultaneous extreme elastic and plastic deformations. We demonstrate our approach on a variety of examples that exhibit a wide range of material behaviors.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Animation*

Additional Key Words and Phrases: viscoelastic materials, point-based animation, natural phenomena, physics-based animation.

1. INTRODUCTION

Computer animation of elastoplastic materials, such as modeling clay, chewing gum, and bread dough has long been a goal of computer graphics, probably because these materials demonstrate such intriguing behavior. As a field we have made progress toward this goal and elastoplastic materials have recently been showcased in special effects, for example the honey in *Bee Movie* [Ruilova 2007] and the food in *Ratatouille* [Goktekin et al. 2007]. However, significant limitations in current techniques remain: some handle only limited plastic deformation, some handle only limited elastic deformation, and others require complex remeshing methodologies.

We present a straightforward, easy-to-implement, point-based approach for animating elastoplastic materials. Our approach can handle extreme elastic and plastic deformations. Because the approach is point-based, there is no need for complex remeshing—the corresponding operation is a simple neighborhood query.

A material that has undergone plastic deformations does not, in general, have a zero-stress state that can be realized in three-dimensional space. Put another way, the *rest space* is not embeddable in three-dimensional space. This fact poses a challenge because elastic forces depend on a mapping from rest space to deformed or *world space*. Some authors have addressed this challenge by keeping plastic offsets from an initial rest state, but this places limitations on the amount of plastic deformation that is possible. Others have abandoned the rest state and keep elastic offsets, resulting in limited elastic deformations and/or artificial plasticity. Borrowing an idea from Wicke and colleagues [2010], we take a compromise approach and store an *embedded space*—the weighted least-squares best embedding of rest space into three dimensions, while preserving the rest space exactly per particle. Nearest neighbor queries can be performed efficiently in embedded space, allowing us to update particle neighborhoods, while deformation can be computed accurately from the rest space. As Figure 5 demonstrates, without updating particle neighborhoods, simulations become unstable under large plastic flows. By maintaining both spaces, our method can handle large plastic and elastic deformation simultaneously.

While our primary contribution is the development of our embedding into three-dimensions, we also suggest an approach for estimating volume that allows for non-uniform samplings. We demonstrate our approach on a variety of examples that display a large range of material behaviors, including simultaneous elastic and plastic deformations. Our method is the first point-based approach capable of simulating these examples, and is much simpler to implement than competing finite element approaches.

2. RELATED WORK

The pioneering work of Terzopoulos and colleagues [1988; 1989] first addressed elastoplastic deformations almost 25 years ago. Perhaps foreshadowing their use in *Ratatouille*, elastoplastic materials played a key role in the short film *Cooking with Kurt* [Fleischer et al. 1987]. Elastoplastic materials have also generated interest in the recent renaissance in physics-based animation. O'Brien and colleagues [2002] modeled ductile fracture by keeping track of plastic offsets that could be subtracted from the total deformation computed using a finite element method, essentially adding limited plasticity to a simulator designed for elastic solids. Irving and colleagues [2004] introduced a multiplicative model of plasticity that is more appropriate for finite plastic deformations. Still, the amount of plastic deformation had to be limited to avoid inverting singular matrices. Bargteil and colleagues [2007] allowed for large plastic flow by abandoning the initial rest configuration in favor of storing a rest state for every element. When rest states became ill-conditioned, the entire object was remeshed (in world space) and simulation variables interpolated to the new mesh. A similar approach was employed by Wojtan and Turk [2008] who incorporated a high-resolution surface mesh. Unfortunately, this wholesale remeshing leads to artificial diffusion due to resampling.

Wicke and colleagues [2010] addressed these problems by introducing the notion of a *material space*—the equilibrium pose of the object in the absence of external forces. Note that this pose has minimum, but non-zero, elastic energy. By performing local remeshing in the material space, artificial diffusion is limited to small errors exactly where plastic flow is occurring. Their approach allows for arbitrary elastic deformations and robustly handles extreme plastic deformations. They achieve very impressive results, but quality tetrahedral mesh generation is still an open problem and local remeshing is even more difficult. Even interfacing with such libraries is necessarily more complex than using point-based structures such as KD-trees. More recently, Narain and colleagues [2013] developed a similar method for thin shells, while Clausen and colleagues [2013] extended the method into a unified framework for animating solids and incompressible liquids.

Our own approach adapts the key idea of Wicke and colleagues [2010], but we apply it in a point-based context where the equivalent operation to remeshing is a simple neighborhood query in the embedded space. Additionally, instead of performing a nonlinear optimization to find the equilibrium pose, we cast the embedding problem as a least-squares optimization, which results in a single linear solve. We use the term *embedded space* instead of *material space* to highlight this difference. Our embedding formulation is quite similar to the "global stitching" approach that Tsiknis and Bridson [2006] developed for cloth. In that context they timestep individual cloth elements independently, then perform a least squares solve to find node positions that match triangle edge vectors as well as possible.

An alternative approach to modeling elastoplasticity has focused on adding elastic behavior to fluid simulations [Goktekin et al. 2004; Losasso et al. 2006]. As one would expect, this approach easily handles arbitrary plastic flow, but demonstrates only limited elastic effects. However, this approach is able to animate a variety of common materials, such as honey [Ruilova 2007].

Researchers have also explored point-based animation of elastoplastic materials. In this case, moving least squares methods are used to estimate the deformation gradient and compute elastic forces. To handle limited plasticity, Müller and colleagues [2004] introduced plastic offsets similar to those employed by O'Brien and colleagues [2002]. For larger plastic deformations, they switched to a model closer to that of Goktekin and colleagues [2004] that essentially stores elastic offsets that can be used to compute elastic forces. In their framework, materials can switch from one model to the other. Keiser and colleagues [2005] introduced a unified framework by combining forces from smoothed particle hydrodynamics (SPH) with elastic forces. Plasticity was modeled with additive plastic offsets, resulting in limitations on the amount of plastic flow. Solenthaler and colleagues [2007] also adopted additive plastic offsets. Gerszewski and colleagues [2009] developed a technique that tracked the deformation gradient through time, allowing the use of multiplicative plasticity and abandoning the storage of a rest state. This resulted in a unified framework for elastic solids and fluids, but, because they lack a rest state, they were only able to handle limited elastic deformations. They also demonstrate the limitations of both additive strain models and naíve plastic offsets.

The material point method (MPM) [Sulsky et al. 1994] is another point-based approach to simulating elastic and elastoplastic materials. It extends marker-and-cell [Harlow and Welch 1965], particle-in-cell [Harlow 1963] and fluid-implicit-particle [Brackbill and Ruppel 1986] methods to "history dependent materials." Instead of direct communication between particles, they communicate through a background grid; making the method extremely efficient and popular for solving large problems on large computing clusters. Unfortunately, like other methods without a rest state MPM suffers from drift, which is exacerbated by a Taylor expansion approximation of the deformation gradient, limiting the ability to simulate large elastic deformations over long timeframes. MPM was recently used in graphics to animate snow by Stomakhin and colleagues [2013].

Clavet and colleagues [2005] modeled elastoplastic materials with a mass-spring system in which springs were dynamically inserted and removed. Their springs explicitly model elastic and viscous forces and include a model of plastic flow. By explicitly storing an evolving rest state that accounts for plastic deformation we are, for the first time, able to accommodate arbitrary elastic deformations and large plastic flow in a unified point-based setting. More recently, Müller and Chentanez [2011] extended point-based elastoplastic simulation to handle degenerate neighborhoods with *oriented particles*; this might be a compelling idea for future work.

It is worth noting that most previous point-based approaches for animating large plastic flows incoporated SPH-like *pressure forces*. Such forces provide additional stability [Gerszewski et al. 2009] by favoring uniform sampling of simulated materials. However, they also alter the behavior of the underlying material model, preventing, for example, the simulation of hyperelastic materials. Moreover, such forces do not allow for adaptive sampling as in Figure 8 without employing complex adaptive simulation frameworks [Adams et al. 2007]. By eschewing these non-physical pressure forces and relying exclusively on elastic forces and a volume preserving plasticity model, our framework allows for the simulation of arbitrary material models. While a subtle point, this represents a significant scientific advance and our examples demonstrate far larger plastic deformations than previously possible with point-based methods for purely elastoplastic materials.

3. METHODS

The core of our approach is the maintenance of three domains: world space, rest space, and embedded space. *World space* refers to the current, deformed object configuration. *Rest space* is local to a particle and refers to the particle's preferred relative positions of its neighbors, stored as 3D vectors per neighbor. In general, the rest space cannot be represented as a set of 3D points, as per-neighbor vectors cannot be reconciled exactly between all particles. We introduce the *embedded space*, which is a global, least-squares best fit of the rest space into three dimensions. It can be thought of as an approximate reference configuration for the material. We store the rest space per each particle as a 3D vector for each neighbor. The rest space and world space are used to compute the deformation gradient, from which we can compute elastic forces and plastic deformations. The embedded space allows efficient updates to neighborhoods in the presence of plastic deformation. Our method is summarized in Algorithm 1.

Algorithm 1 Deformation Embedding for Elastoplasticity

```
initialize kernel support radii and particle neighborhoods
while Animating do
    approximate deformation gradient
    factor deformation into elastic and plastic components
    compute elastic forces
    integrate elastic, body, and damping forces
    compute global embedding
    resample particles
    update neighborhoods
end while
```

Notation. We use $\mathbf{x}_{ij}$, $\mathbf{e}_{ij}$, $\mathbf{u}_{ij}$ to denote to *neighbor vectors*—vectors between particles i and j—in world, embedded, and rest space, respectively. Capital bold face letters refer to matrices. The index j refers to the neighbors of particle i, which may change during the course of the simulation.

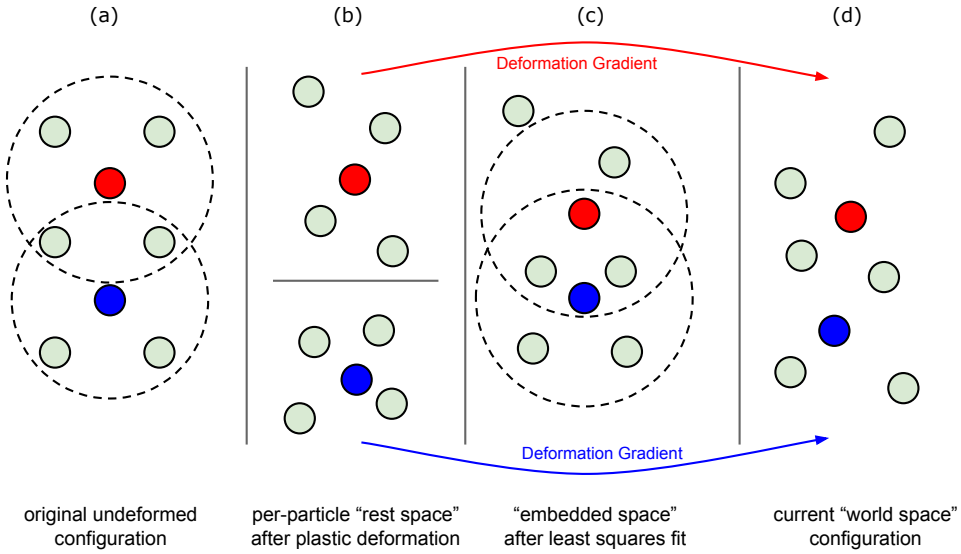

Fig. 1: *From the initial configuration (a), each particle's neighborhood undergoes plastic deformation, resulting in a new* rest space *configuration (b). However, the red and blue particles disagree about where their shared neighbors should be. Performing a least squares global fit, we obtain the* embedded space *configuration (c), which is used to update particle neighborhoods. Each particle's deformation gradient maps from its own rest space to the current* world space *configuration (d).*

3.1 Approximating Deformation

Following the work of Müller and colleagues [2004], we use moving least squares (MLS) to approximate the deformation gradient, $\mathbf{F}$. For each particle i, we seek the matrix $\mathbf{F}_i$ that minimizes

$$\sum_j ||w_{ij}(\mathbf{F}_i\mathbf{u}_{ij} - \mathbf{x}_{ij})||^2 \tag{1}$$

where the sum is taken over the neighbors, j, and w_{ij} is a weighting kernel evaluated in rest space. We compute the minimizer by solving the following linear system,

$$\mathbf{F}_i(\sum_j w_{ij}\mathbf{u}_{ij}\mathbf{u}_{ij}^T) = \sum_j w_{ij}\mathbf{x}_{ij}\mathbf{u}_{ij}^T. \tag{2}$$

We refer to the quantity $\sum_j w_{ij}\mathbf{u}_{ij}\mathbf{u}_{ij}^T$ as the particle i's basis matrix, $\mathbf{A}_i$, which we invert to solve the system.

Particle Volume. It is common in point-based animation to assume that particles have a fixed, often uniform, mass or volume. This assumption often leads to density fluctuations that can cause spurious oscillations. In the context of plastic deformation, it is desirable to allow the sampling to vary over time and thus we require a way of estimating the volume occupied by a particle. Noting that $\mathbf{A}$ serves the same purpose as the basis matrix in tetrahedral finite element methods, that the volume of a tetrahedron is a multiple of the determinant of the basis matrix, and that the units of $\det(\mathbf{A})$ are m^6, we approximate the volume of a particle as,

$$V_i = \sqrt{\frac{\det(\mathbf{A}_i)}{(\sum_j w_{ij})^3}}. \tag{3}$$

The denominator normalizes with respect to the weights used to compute $\mathbf{A}$, which are cubed by the determinant operation. We assign a constant density to the material and allow each particle's mass to change over time. The total mass of the material fluctuates slightly due to this approximation (see Section 4), but we did not observe any resulting artifacts in our examples.

3.2 Elastoplastic Model

We use a multiplicative plasticity model, where the total deformation gradient is the product of elastic and plastic deformations, i.e. $\mathbf{F} = \mathbf{F}_e\mathbf{F}_p$.

Following Irving and colleagues [2004], we diagonalize $\mathbf{F}_e = \mathbf{U}\hat{\mathbf{F}}_e\mathbf{V}^T$ and compute the diagonalized first Piola-Kirchhoff stress:

$$\hat{\mathbf{P}} = \lambda \mathrm{tr}(\hat{\mathbf{F}}_e - \mathbf{I})\mathbf{I} + 2\mu(\hat{\mathbf{F}}_e - \mathbf{I}). \tag{4}$$

We then rotate the diagonalized stress $\mathbf{P} = \mathbf{U}\hat{\mathbf{P}}\mathbf{V}^T$ and compute symmetrized forces

$$\begin{aligned} \mathbf{f}_i &= V_i\mathbf{P}_i\mathbf{A}_i^{-1}w_{ij}\mathbf{u}_{ij} \\ \mathbf{f}_j &= -\mathbf{f}_i \end{aligned} \tag{5}$$

which are linear in position.

Using the plasticity model developed by Bargteil and colleagues [2007], we factor our deformation gradient, $\mathbf{F}$, into elastic and plastic components $\mathbf{F}_e$ and $\mathbf{F}_p$, respectively. This model preserves volume in rest space, and accounts for a range of material properties including yield point, creep, and work hardening/softening. Specifically, we compute the singular value decomposition of $\mathbf{F} = \mathbf{U}\hat{\mathbf{F}}\mathbf{V}^T$ and factor the diagonal matrix $\hat{\mathbf{F}}$ into

$\hat{\mathbf{F}}_e\hat{\mathbf{F}}_p$. To compute $\hat{\mathbf{F}}_p$, we first compute

$$\tilde{\mathbf{F}} = \frac{\hat{\mathbf{F}}}{\det(\hat{\mathbf{F}}^{1/3})}. \quad (6)$$

This computation is essentially normalizing $\tilde{\mathbf{F}}$ to have determinant 1, ensuring volume preservation. Since $\hat{\mathbf{F}}$ is diagonal, we can perform the cube root operation on each entry independently. We then compute $\hat{\mathbf{F}}_p = \tilde{\mathbf{F}}^{\gamma}$, where

$$\gamma(\mathbf{P}, P_Y, \nu, \alpha, K) = \frac{\nu\left(||\mathbf{P}|| - \max\left(P_Y + K\alpha, 0\right)\right)}{||\mathbf{P}||} \quad (7)$$

which is clamped between 0 and 1. We use the total, unfactored stress, $\mathbf{P}$, to compute γ. P_Y is the yield stress (at lower stresses there is no plastic flow); ν is the flow rate, which controls how quickly flow will occur; α is the total accumulated plastic stress as computed by the simulation; and K controls the rate of work hardening or softening. P_Y, ν, and K are user-defined, while α is a simulation variable integrated through time. The $K\alpha$ term can be though of as adjusting the yield point. To prevent rotation of the plastic deformation, we rotate the diagonalized deformation into $\mathbf{F}_e = \mathbf{U}\hat{\mathbf{F}}_e\mathbf{V}^T$ and $\mathbf{F}_p = \mathbf{V}\hat{\mathbf{F}}_p\mathbf{V}^T$. We use $\hat{\mathbf{F}}_e$ to compute elastic stress in Equation (4). For more details, refer to Bargteil and colleagues [2007].

For damping forces, we use the SPH viscosity formulation of Müller and colleagues [2003]. These forces act to minimize velocity differences between neighboring particles, and are computed by

$$\mathbf{f}_{ij} = \eta\frac{m_j}{\rho_j}\left(\mathbf{v}_j - \mathbf{v}_i\right)\nabla^2 W(\mathbf{u}_{ij}, h). \quad (8)$$

Note that this simple model damps all relative motion and is non-zero for rigid rotation. Alternatively one could base damping forces on the time derivative of stress.

To update positions and velocities, we use either an explicit leapfrog scheme, or the following Newmark integration scheme adapted from Bridson and colleagues [2003], which integrates viscous forces implicitly and elastic forces explicitly:

- $v^{n+1/2} = v^n + \frac{\Delta t}{2}a_{viscous}(t^n, x^n, v^{n+1/2})$
- $x^{n+1} = x^n + \Delta t v^{n+1/2}$
- $v^{n+1} = v^{n+1/2} + \frac{\Delta t}{2}a_{elastic,body}(t^{n+1}, x^{n+1}, v^{n+1/2})$

Each timestep of the Newmark scheme is more expensive than the explicit scheme, but for simulations where high viscosity is desired, the implicit viscous solve permits larger timesteps, resulting in better performance overall. We experimented with linearly implicit Euler integration, but found that rotational motion was severely damped (see accompanying video) and did not pursue this further.

3.3 Embedding Deformation

As the material undergoes plastic deformation a particle's neighbors may move far away in rest space and no longer provide useful information. To update these neighborhoods when such changes occur, we maintain a globally optimal embedded space, encoded as a three-dimensional position, $\mathbf{e}_i$, for each particle. To update these positions, we first compute temporary rest space vectors that incorporate the plastic deformation that occurred over the previous timestep, $\tilde{\mathbf{u}}_{ij} = \mathbf{F}_p\mathbf{u}_{ij}$.

The optimal least-squares embedding of the particles into three-dimensional space will minimize the discrepancy of neighbor vectors between the embedded and rest spaces. We formulate this optimization over the embedded positions, $\mathbf{e}_i$, as a weighted linear least-squares problem

$$\underset{\mathbf{e}_i}{\operatorname{argmin}} \sum_{i,j} ||w_{ij}(\tilde{\mathbf{u}}_{ij} - \mathbf{e}_{ij})||^2, \quad (9)$$

which requires solving three decoupled, over-constrained linear systems: one for each dimension, x, y, and z. In matrix form, each system can be expressed as

$$\mathbf{Ce} = \mathbf{u}. \quad (10)$$

$\mathbf{C}$ is similar to the constraint matrices that appear when using Lagrange multipliers or projection methods for constrained dynamics. In our case, each row, r, of $\mathbf{C}$ encodes the (weighted) constraint that if particle j is a neighbor of particle i then $\mathbf{e}_{ij} = \tilde{\mathbf{u}}_{ij}$. More specifically,

$$\begin{aligned}\mathbf{C}_{ri} &= -w_{ij}\\ \mathbf{C}_{rj} &= w_{ij}.\end{aligned}$$

The vector $\mathbf{e}$ contains the embedded space positions for all particles and $\mathbf{u}_{ij} = w_{ij}\tilde{\mathbf{u}}_{ij}$. Note that while $\mathbf{e}_{ij} = -\mathbf{e}_{ji}$, the same is not true for $\tilde{\mathbf{u}}_{\mathbf{ij}}$ and $\tilde{\mathbf{u}}_{\mathbf{ji}}$ because rest space vectors, $\mathbf{u}_{ij}$ are transformed by $\mathbf{F}_{\mathbf{p_i}}$ each timestep and diverge over time.

Because each particle has roughly 32 neighbors (constraints), $\mathbf{C}$ is roughly $32n \times n$. We solve this non-square, highly over-constrained system by applying conjugate gradients to the normal equations,

$$\mathbf{C}^T\mathbf{Ce} = \mathbf{C}^T\mathbf{u}. \quad (11)$$

Denoting the set of neighbors of i as $n(i)$, the entries of $\mathbf{C}^T\mathbf{C}$ are

$$\left(\mathbf{C}^T\mathbf{C}\right)_{ij} = \begin{cases} 2\sum\limits_{k\in n(i)} w_{ik}^2 & \text{if } i = j\\ -2w_{ij}^2 & \text{if } i \in n(j) \wedge j \in n(i)\\ -w_{ij}^2 & \text{if } i \in n(j) \oplus j \in n(i)\\ 0 & \text{otherwise.}\end{cases} \quad (12)$$

This matrix is symmetric and diagonally dominant, but if all the neighborhoods are symmetric will have a null-space containing the constant vector (corresponding to global translations). We explicitly remove this by adding a row to $\mathbf{C}$ constraining the first particle to maintain its current embedded position. Note that the system is *not* invariant to global rotations, so no special handling is required. Because the system changes structure as neighborhoods change or particles are split or merged, prefactoring is not possible.

While these embedded space positions provide a globally consistent reference configuration, there will still be discrepancy between the rest and embedded spaces. The vectors $\tilde{\mathbf{u}}_{ij}$ become the rest space vectors $\mathbf{u}_{ij}$ for the next timestep.

It is natural to compare our embedding to the elastic-energy minimization approach of Wicke and colleagues [2010]. Interestingly, our linear embedding does not capture some rotational changes to the rest space. However, world space dynamics are nearly identical compared to the more expensive nonlinear optimization used by Wicke and colleagues, as shown in Figure 2. This behavior is because rotations typically do not change the neighbors with the highest smoothing weights (those closest to the particle), and the rest-space vectors for those particles are preserved exactly. In our experiments, our linear solve was nearly twice as fast as the elastic-energy minimization approach of Wicke and colleagues.

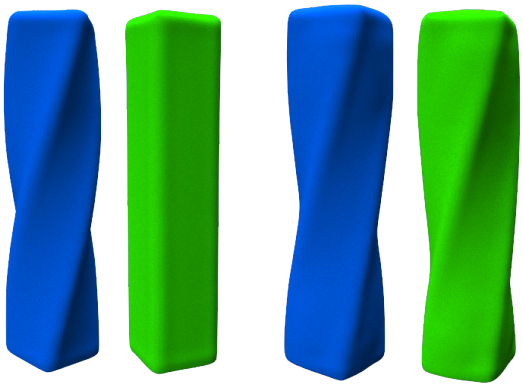

Fig. 2: *Comparison between our linear embedding (left) and nonlinear embedding (right) for a twisted plastic bar. The world space behavior (blue) is nearly identical, even though the linear embedding captures very little rotation. Though small, changes in the linear embedding did cause neighborhoods to change in this example.*

Another difference between our approach and that of Wicke and colleagues [2010] is in how we store rest space. Wicke and colleagues store a single 3×3 matrix per tetrahedron, which they call a *plastic offset*, that maps from embedded space to rest space. Instead, we store rest space as about 30 rest-space neighbor vectors per particle. While our approach requires more memory, we found storing a single matrix per particle problematic as a simple least-squares fit introduced too much error into the mapping from embedded to rest space. If memory consumption is a significant concern, higher order fitting techniques and/or compression may prove effective.

3.4 Updating Neighborhoods

After plastic flow, a particle's neighbors may have moved far away, and no longer provide useful information about the deformation gradient. We therefore update each particle's neighborhood by finding the nearest neighbors in the embedded space. We expect that neighborhoods in embedded space are a good approximation of neighborhoods in rest space, and they can be queried efficiently using a KD-tree. The only information lost in this process is the difference between the rest space vector, $\mathbf{u}_{ij}$, and the embedded space vector, $\mathbf{e}_{ij}$, when a particle j moves out of particle i's neighborhood. When a new particle enters a neighborhood, we initialize $\mathbf{u}_{ij} = \mathbf{e}_{ij}$. This process does lose some information about the rest state (and, consequently, the internal stress), but the lost information is the least useful as it comes from the particles that are farthest away.

3.5 Particle Resampling

To improve computational efficiency and stability, we enable particle resampling in our simulator. We selectively split and merge particles when neighborhood sampling is either too dense, or too sparse. We quantify this by computing a basis matrix for each particle as

$$\mathbf{B}_i = \sum_j \frac{\mathbf{u}_{ij}\mathbf{u}_{ij}^T}{||u_{ij}||^4}, \tag{13}$$

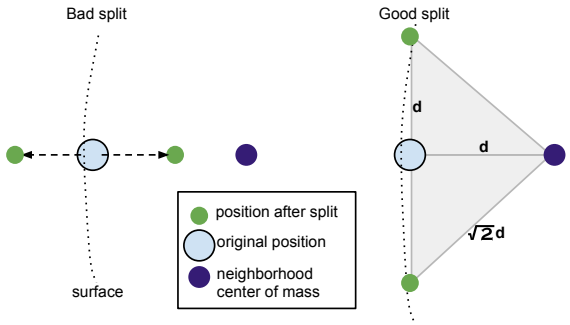

Fig. 3: *We cancel splits that are likely to cause popping artifacts near the object surface.*

where the index j runs over the neighbors of particle i, which may vary over time. Note that this is simply a weighted covariance matrix, where the 4th power in the denominator results in a $1/r^2$ falloff away from particle i.

We perform an eigendecomposition of this matrix and examine its eigenvalues to decide to split or merge particles. If the maximum eigenvalue is small, this indicates that there are too few particles nearby, and the particle should be split. We split the particle into two particles and place them along the eigenvector with the minimum eigenvalue. They are offset from the particle's original position by half the average distance to the particles in the neighborhood.

Splitting particles near the surface can potentially cause rendering artifacts on the object's surface, which we attempt to mitigate in two ways. First, we chose the middle eigenvalue as a splitting direction because for surface particles, the smallest eigenvalue is typically perpendicular to the surface, and the largest is already well sampled. The middle eigenvector corresponds to a direction tangent to the surface that is poorly sampled. Second, we reject splits that are likely to cause surface artifacts. We compute the distance from the original particle to the center of mass of its neighborhood. If either split particle is more than a factor of $\sqrt{2}$ away from the center of mass, the split is cancelled. Intuitively, this condition attempts to eliminate splits that are not tangent to the surface (Figure 3). We also cancel splits that would place a new particle inside of an obstacle. We classify particles as "surface particles" if the distance from the particle to it's neighborhood center of mass is greater than a user-defined threshold, as interior particles likely have a uniformly distributed neighborhood, while surface particles will have a neighborhood skewed away from the surface.

If the minimum eigenvalue is too large, then there are too many particles nearby and the particle should be merged with its nearest neighbor. The merged particle is placed halfway between the two original particles. The split and merge thresholds are a user-specified parameter, but reasonable values can be chosen by examining minimum and maximum basis matrix eigenvalues from the object's initial configuration.

While much more complex resampling methods exist [Adams et al. 2007; Ando et al. 2012], we consider the ease of implementation of our technique to be a major benefit.

3.6 Implementation Details

For our weighting kernels, we use the standard SPH kernels of Müller and colleagues [2003]: The *poly6* kernel, Equation (14), for all weights except for viscosity, where we use the SPH viscosity kernel, Equation (15).

$$W_{\text{poly6}}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \le r \le h \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

$$\nabla^2 W(\mathbf{r}, h) = \frac{45}{\pi h^6} (h - r) \tag{15}$$

We initialize each particle's smoothing radius by finding its nearest 32 neighbors, and setting its radius to twice the average neighbor distance. During the simulation, a particle can have a maximum of 32 neighbors, which must be within its radius in embedded space. This constraint limits the number of neighbors and the number of non-zeroes in our matrices.

Rendering. Generating visually appealing surfaces from particle data for rendering is a difficult problem. In the examples in this paper, we used two methods: the skinning method of Bhattacharya and colleagues [2011], and a simple surface mesh embedding technique. When embedding a surface mesh, we update mesh vertex positions with the weighted average of the displacements of nearby particles from the first frame of animation. The weights are computed using the *poly6* kernel with a user defined, constant support radius. This approach is sufficient for simulations with largely elastic deformations, but breaks down for large plastic flows. Including more robust surface tracking and using the embedded space to update weights are interesting directions for future work.

4. RESULTS AND DISCUSSION

We have used our proposed method to simulate materials exhibiting a wide range of elastoplastic behavior as shown in our figures and included videos. Timing results are shown in Table I and Table II. All examples were run on a dual-2.8 GHz Intel Xeon processor machine using up to 12 cores. Trivially parallelizable loops were multithreaded using the Intel TBB library.

Figures 2 and 5 demonstrate the importance of the two components of our mapping to rest space: plastic offsets and embedded space. Without plastic offsets, the final world pose of the bar in Figure 2 would not include any twist. Without neighborhood updates, a particle's neighborhood may become degenerate, leading to an ill conditioned basis matrix and instability. By using embedded space to update neighborhoods, we maintain well sampled neighborhoods and increase stability (see Figure 5). Resampling also improves stability by merging particles that become extremely close in embedded space.

Figures 6 and 7 compare the world space and embedded space deformations of a bunny dropped onto obstacles. The embedded space captures the key features of the global plastic deformation. The high frequency details of the original mesh are preserved as the material flows plastically.

Figure 8 demonstrates our method's robust handling of uneven sampling. Our new volume estimate allows us to simulate objects with dramatically varying particle densities. Previous work has required expensive fully adaptive sampling techniques [Adams et al.

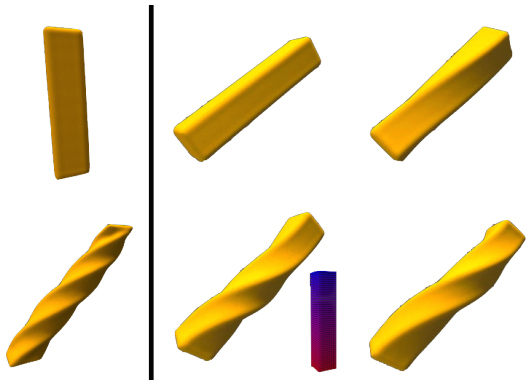

Fig. 4: *A bar is twisted and sheared (left), then released. The final world space configurations for (clockwise from top, left) elastic, slightly plastic, highly plastic, and varying plasticity materials. For the nonuniform bar, the plastic flowrate varies from high(red) to low(blue) along the bar.*

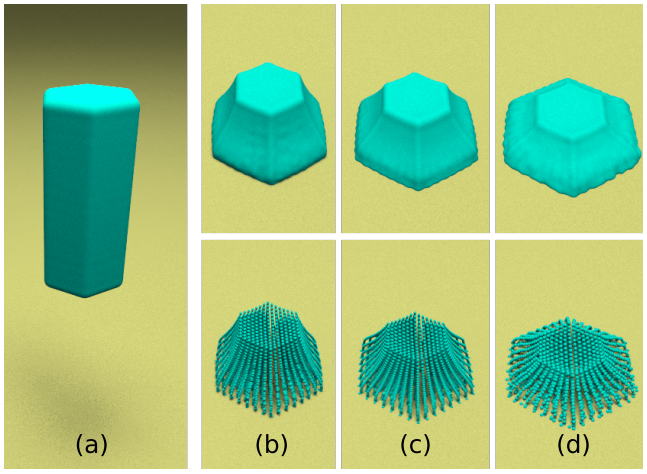

Fig. 5: *A plastic bar is dropped onto the ground(a). Without neighborhood updates (b), the simulation becomes unstable after the object is significantly flattened. Our embedding allows neighborhood updates which improve stability (c), but this also becomes unstable. Adding resampling (d) preserves stability through the entire scene.*

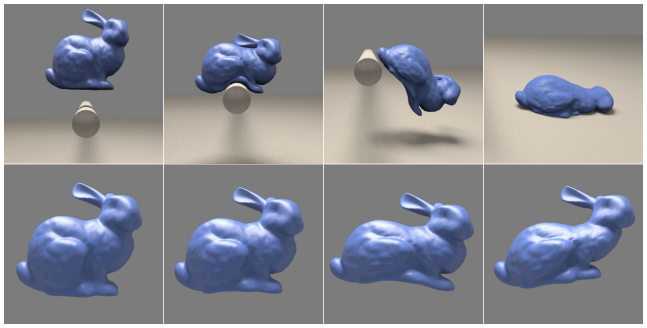

Fig. 6: *World space (above) and embedded space (below) of bunny dropped on a rigid bar.*

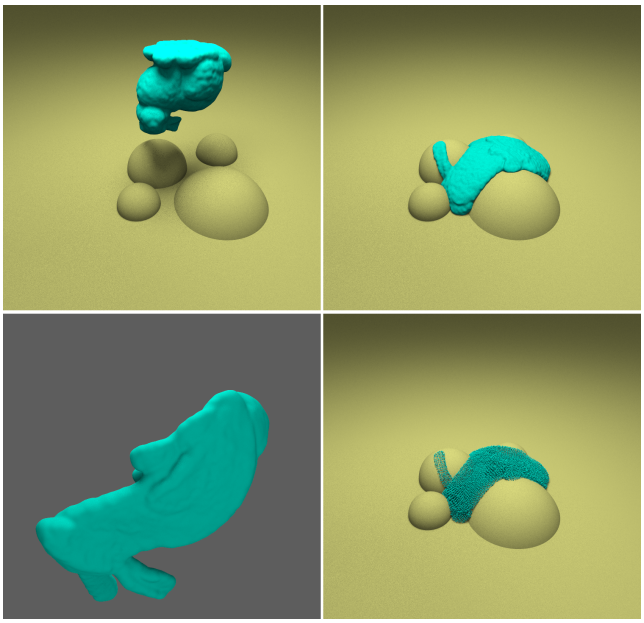

Fig. 7: *A bunny is dropped on a set of spheres. Clockwise from top left: Initial configuration, world (skinned), world(particles), and embedded spaces after impact.*

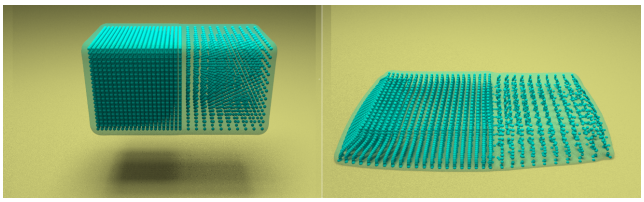

Fig. 8: *A plastic block with dramatically different sampling densities flows when dropped.*

2007] or special handling of boundaries where regions of varying particle density meet [Solenthaler and Gross 2011].

As the "upset fowl" collides with the rigid obstacles in Figure 10, it undergoes significant elastic and plastic deformation. As it is crushed, it reacts elastically, lifting the upper block before succumbing to creep and flowing plastically.

As shown in Figure 9, our volume estimate closely tracks both the "ground truth" volume computed using the skinned, embedded space mesh, and the estimate used by SPH based approaches.

Example	#Particles	Δt	sec/frame
Figure 2 linear	5,000	.001	2.36
Figure 2 nonlinear	5,000	.001	4.30
Figure 5	5,408	.001	1.2
Figure 7	36,675	.0004	278
Figure 8	10,470	.005	2.39
Figure 10	21,708	.0001	92

Table I. : *Timing results for pictured examples. Time is given in seconds to produce one 30 Hz frame of animation.*

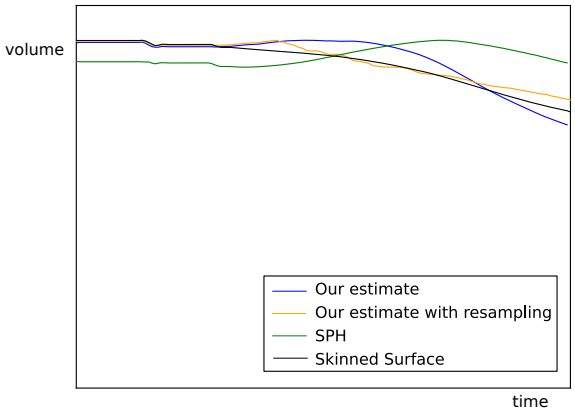

Fig. 9: *Estimated total volume computed using our new approach, SPH, and the skinned embedded space mesh for Figure 5.*

Step	% computation time
Compute basis matrices	3.54
Viscosity solve	17.19
Compute forces	4.55
Explicit integration step	0.36
Embedding solve	41.70
Resampling	2.24
Neighborhood updates	22.91
Compute plastic offsets	3.73
Handle collisions	0.19

Table II. : *Timing detail for Figure 5*

Limitations and Future Work. In practice, we found our method to be robust and stable for bulk motion. Typical failure cases are a single particle or a small group of particles drifting away from the bulk material, or getting caught on an obstacle (we colloquially refer to such particles as "jerk particles"). These artifacts are usually induced by extreme or violent deformations or sharp corners of rigid obstacles. In a production environment, these troublesome particles can be easily deleted before skinning or rendering. Using a robust least squares solve, or minimizing the $\mathcal{L}^1$ norm in our embedding may eliminate these problems, albeit at significant additional cost.

In the current implementation, we allow particles to apply forces through solid obstacles, which can lead to unnatural and undesirable behavior near sharp corners. Also, we do not check for collisions between particles nearby in world-space, but far apart in embedded space, which can result in interpenetration artifacts. We suspect both problems can be efficiently resolved with an additional spatial partitioning scheme in world space.

Although our simple resampling scheme works to improve stability, our splitting method can cause surface artifacts. Increasing the model resolution reduces these artifacts, but a more robust resampling or skinning approach is a possible direction of future work. Additionally, aggressively coarsening regions on the object interior could lead to improved performance while preserving surface detail.

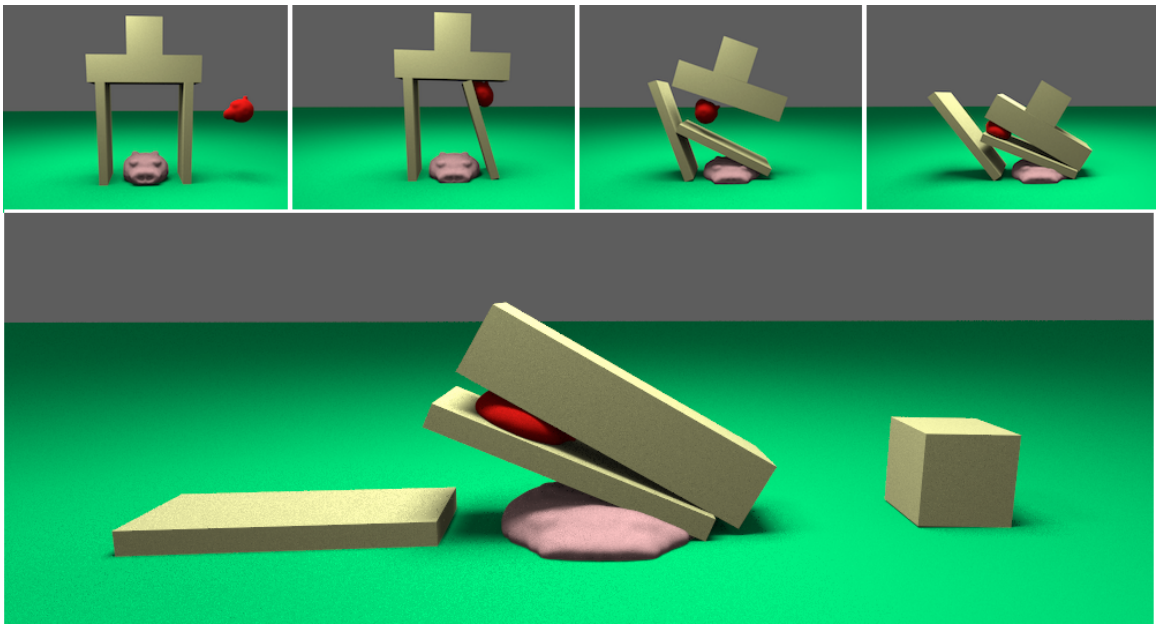

Fig. 10: *An "upset fowl" destroys a pig's house.*

It is common for materials to become brittle after work hardening and become prone to fracture. In our implementation, such topological changes happen by accident, when particle neighborhoods change and parts of the material stop interacting. By intentionally causing such topological changes, our method could be adapted to simulate ductile fracture. Relatedly, we do nothing in particular to address "fusing" in embedded space, but did not notice this to be a problem in our examples. One could address such issues using an embedded surface mesh and ray casting when computing neighborhoods. We also note that many previous techniques, both point-based [Gerszewski et al. 2009] and mesh-based [Bargteil et al. 2007], allow such fusing and sometimes consider it a "feature."

Because points behave nearly independently, this method is well suited to parallelization on multi-core architectures or GPUs. Most computations only output results for a single particle, and read only from nearby particles. We employed multithreading to parallelize these independent loops on the CPU, however these access patterns are also well suited for the memory hierarchy on current GPU architectures. Preliminary GPU implementation efforts resulted in an order of magnitude speedup for simple elastic examples.

We have presented a simple to implement point-based method for animating elastoplastic materials. By maintaining a globally optimal fit of the material's reference configuration we are able to simulate materials undergoing simultaneous and extreme elastic and plastic deformations. Like all point-based methods, our approach trades the numerical advantages of discretizing the domain into disjoint elements (better conditioning, sparser matrices, etc.) for the significant implementation advantage and simplicity of avoiding computing such a discretization altogether.

REFERENCES

ADAMS, B., PAULY, M., KEISER, R., AND GUIBAS, L. J. 2007. Adaptively sampled particle fluids. *ACM Trans. Graph. 26,* 3, 48.

ANDO, R., THUREY, N., AND TSURUNO, R. 2012. Preserving fluid sheets with adaptively sampled anisotropic particles. *IEEE Trans. Vis. Comp. Graph. 18,* 8, 1202–1214.

BARGTEIL, A. W., WOJTAN, C., HODGINS, J. K., AND TURK, G. 2007. A finite element method for animating large viscoplastic flow. *ACM Trans. Graph. 26,* 3.

BHATTACHARYA, H., GAO, Y., AND BARGTEIL, A. W. 2011. A level-set method for skinning animated particle data. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

BRACKBILL, J. U. AND RUPPEL, H. M. 1986. Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *J. Comput. Phys. 65*, 314–343.

BRIDSON, R., MARINO, S., AND FEDKIW, R. 2003. Simulation of clothing with folds and wrinkles. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 28–36.

CLAUSEN, P., WICKE, M., SHEWCHUK, J. R., AND O'BRIEN, J. F. 2013. Simulating liquids and solid-liquid interactions with lagrangian meshes. *ACM Trans. Graph. 32,* 2 (Apr.), 17:1–15. Presented at SIGGRAPH 2013.

CLAVET, S., BEAUDOIN, P., AND POULIN, P. 2005. Particle-based viscoelastic fluid simulation. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 219–228.

FLEISCHER, K., WITKIN, A., KASS, M., AND TERZOPOULOS, D. 1987. Cooking with kurt. *Animation in ACM SIGGRAPH Video Review* 36.

GERSZEWSKI, D., BHATTACHARYA, H., AND BARGTEIL, A. W. 2009. A point-based method for animating elastoplastic solids. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

GOKTEKIN, T. G., BARGTEIL, A. W., AND O'BRIEN, J. F. 2004. A method for animating viscoelastic fluids. *ACM Trans. Graph. 23,* 3, 463–468.

GOKTEKIN, T. G., REISCH, J., PEACHEY, D., AND SHAH, A. 2007. An effects recipe for rolling a dough, cracking an egg and pouring a sauce. In *ACM SIGGRAPH 2007 sketches*. 67.

HARLOW, F. AND WELCH, J. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. *The Physics of Fluids 8*, 2182–2189.

HARLOW, F. H. 1963. The particle-in-cell method for numerical solution of problems in fluid dynamics. *Experimental arithmetic, high-speed computations and mathematics*.

IRVING, G., TERAN, J., AND FEDKIW, R. 2004. Invertible finite elements for robust simulation of large deformation. In *The Proceedings of the ACM/Eurographics Symposium on Computer Animation*. 131–140.

KEISER, R., ADAMS, B., GASSER, D., BAZZI, P., DUTRÉ, P., AND GROSS, M. 2005. A unified Lagrangian approach to solid-fluid animation. In *The Proceedings of the Symposium on Point-Based Graphics*. 125–133.

LOSASSO, F., SHINAR, T., SELLE, A., AND FEDKIW, R. 2006. Multiple interacting liquids. *ACM Trans. Graph. 25,* 3, 812–819.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 154–159.

MÜLLER, M. AND CHENTANEZ, N. 2011. Solid simulation with oriented particles. In *ACM Trans. Graph.* Vol. 30. ACM, 92.

MÜLLER, M., KEISER, R., NEALEN, A., PAULY, M., GROSS, M., AND ALEXA, M. 2004. Point based animation of elastic, plastic and melting objects. In *The Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 141–151.

NARAIN, R., PFAFF, T., AND O'BRIEN, J. F. 2013. Folding and crumpling adaptive sheets. *ACM Trans. Graph. 32,* 4 (July), 51:1–8.

O'BRIEN, J. F., BARGTEIL, A. W., AND HODGINS, J. K. 2002. Graphical modeling and animation of ductile fracture. *ACM Trans. Graph. 21,* 3, 291–294.

RUILOVA, A. 2007. Creating realistic CG honey. In *ACM SIGGRAPH 2007 posters*. 58.

SOLENTHALER, B. AND GROSS, M. 2011. Two-scale particle simulation. *ACM Trans. Graph. 30,* 4 (July), 81:1–81:8.

SOLENTHALER, B., SCHLÄFLI, J., AND PAJAROLA, R. 2007. A unified particle model for fluid-solid interactions. *Journal of Visualization and Computer Animation 18,* 1, 69–82.

STOMAKHIN, A., SCHROEDER, C., CHAI, L., TERAN, J., AND SELLE, A. 2013. A material point method for snow simulation. *ACM Trans. Graph. 32,* 4, 102:1–102:10.

SULSKY, D., CHEN, Z., AND SCHREYER, H. L. 1994. A particle method for history-dependent materials. *Computer Methods in Applied Mechanics and Engineering*, 179–196.

TERZOPOULOS, D. AND FLEISCHER, K. 1988. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *The Proceedings of ACM SIGGRAPH*. 269–278.

TERZOPOULOS, D., PLATT, J., AND FLEISCHER, K. 1989. Heating and melting deformable models (from goop to glop). In *The Proceedings of Graphics Interface*. 219–226.

TSIKNIS, D. 2006. Better cloth through unbiased strain limiting and physics-aware subdivision. M.S. thesis, University of British Columbia.

WICKE, M., RITCHIE, D., KLINGNER, B. M., BURKE, S., SHEWCHUK, J. R., AND O'BRIEN, J. F. 2010. Dynamic local remeshing for elastoplastic simulation. *ACM Trans. Graph. 29*, 49:1–49:11.

WOJTAN, C. AND TURK, G. 2008. Fast viscoelastic behavior with thin features. *ACM Trans. Graph. 27*, 47:1–47:8.