Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2010)
M. Otaduy and Z. Popovic (Editors)

Point Cloud Glue: Constraining simulations using the Procrustes transform

Christopher D. Twigg and Zoran Kačić-Alesić

Industrial Light & Magic†

Abstract

In physical simulation, it is frequently useful to define constraints between deformable objects, ensuring that one object follows another. Existing techniques for enforcing these constraints define the relationship between the objects using barycentric coordinates, a linear combination of vertices. While simple to implement and understand, barycentric coordinates have one important drawback: for stability, weights must be non-negative, which limits the types of constraints that can be defined. We introduce the Point Cloud Glue*, which uses the nearest fit rigid rotation (the Procrustes transform) to the deformable object's particles. Our key contribution is to demonstrate that we can differentiate through this minimization in a numerically stable manner, allowing our method to be used in many constrained dynamics systems including those based on bindings/embeddings and those based on Lagrange multipliers. We demonstrate the flexibility of our method through several examples.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

1. Introduction

Simulation of deformable objects and rigid bodies has become a staple of visual effects work. The variety of situations in which physical simulations are used and the complexity of the simulated worlds is rapidly increasing, and so is the expectation of the tools' robustness and ease of use. It is often the case that geometric objects from many different sources, created using disparate design approaches and with little or no consideration for simulation requirements, are quickly thrown together into a complex scene, with the expectation that they will interact in physically believable and visually interesting ways. We thus need the simulation equivalent of duct tape: a general, simple, quick, and robust tool to keep things together wherever and whenever needed.

While methods for constraining rigid bodies in this manner are robust and widespread [MW88, BB88, WTF06], methods for constraining deforming simulations are commonly restricted to the use of linear combinations of vertices. A simple example of this is a point-to-triangle constraint, which constrains a "bound" point $\mathbf{x}_b$ to a location on a "parent" triangle $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$; see Figure 1. To make this constraint more formal, we could simply define the bound particle's position in terms of the parent particles, $\alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \alpha_3\mathbf{x}_3$; this is sometimes known as an "embedding" of the point into the triangle, or, in the terminology introduces by Sifakis and colleagues [SSIF07], we would say that the particle $\mathbf{x}_b$ is "hard bound" to the parent triangle. Alternatively, we might define a constraint function $c(\mathbf{x}_i) = \mathbf{x}_b - \sum_i \alpha_i\mathbf{x}_i = 0$ to fit into a constraint formulation that uses Lagrange multipliers [PB88, WW90].

Because methods for solving constrained dynamics invariably require the computation of the constraint Jacobian $dc/d\mathbf{x}_i$, defining constraints using barycentric coordinates has the advantage of simplicity. However, they have the limitation that weights should always be positive; if weights are negative (which would happen in our point-to-triangle constraint if the point was outside the triangle), then methods for solving constraints tend to generate poor-quality results. For example, using hard bindings we will find that any force applied to the bound particle actually drives the parent particles apart (see Figure 1, right). Similarly, if we solve constraints using Lagrange multipliers (similar to Barzel and Barr [BB88]), we will find that when we perform a least-

† {ctwigg,zoran}@ilm.com

DOI: 10.2312/SCA/SCA10/045-053

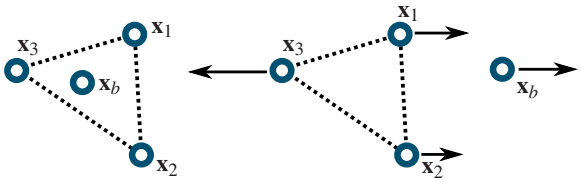

Figure 1: ***(Left)*** *Here, bound point $\mathbf{x}_b$ is constrained to lie at the location $\alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \alpha_3\mathbf{x}_3$.* ***(Right)*** *To constrain $\mathbf{x}_b$ to lie* outside *the parent triangle, we must use a negative weight for α_3, which in a bindings/embeddings frameworks means that forces applied to the bound particle will drive the parent particles apart.*

squares projection of particle velocities (accelerations) onto the space of valid velocities (accelerations), the resulting behaviors will be counterintuitive and potentially unstable.

While this might seem like a minor limitation, it in fact has significant ramifications that make linear constraints unsuitable as an all-purpose simulation duct tape. Defining constraints involving cloth, for example, becomes much more difficult when the constrained points must be *on* the cloth surface. For volumetric simulations, constrained points on one object must be entirely contained within another's tetrahedral mesh, putting an added burden on users and possibly necessitating changes to the original geometry. Ideally, our constraint system should lift these limitations, while being simple, intuitive, and very robust.

Our method, which we call "Point Cloud Glue" due to its applicability to unstructured collections of points, works by computing the nearest-fit rigid transform (the *orthogonal Procrustes transform* [Hig08]) to the particle positions. While this transform has been used previously for computing the *internal* dynamics of objects [MTG04, RJ07], it has not been used in constraint systems like those mentioned above. A key reason for this is that the closest-fit rigid transformation results from a nonlinear minimization, so it is not obvious how to compute its Jacobian besides using finite differences. Because the solution can be posed in terms of the singular value decomposition (SVD), however, we can combine earlier results on the differentiability of the individual SVD terms with new observations specific to the Procrustes problem to derive an analytical expression for the Jacobian that is both fast and robust to compute. The result is a constraint that can be used between virtually any pair of simulated objects, which is easy for users to set up and configure, and which imposes little overhead at simulation time.

1.1. Related work

There are three broad categories of methods for handling constraints in deformable simulation: penalty methods, Lagrange multipliers, and embeddings. Penalty methods are the simplest, and maintain the constraint via stiff springs. Forces can be defined in terms of the gradients of energy functions which take the form $E = \|c(\mathbf{x}_i)\|^2$ [WFB87, BW98]; differentiating E requires computing $dc/d\mathbf{x}_i$. Lagrange multipliers avoid the stiffness issues associated with penalty methods [PB88, WW90, MT92, BW92] and can maintain constraints to high accuracy, usually at the cost of solving a linear system. Any method based on Lagrange multipliers requires the computation of the constraint Jacobian. The method of Gissler and colleagues [GBT06] can be seen as a variant of Lagrange multiplier methods since it uses maximal coordinates and projects down to the constraint manifold; while the method does not explicitly use the constraint Jacobian, all the described constraints are linear, and it is likely that applying the method to constraints like ours would require the use of the constraint Jacobian to solve the resulting nonlinear equation.

The final technique commonly used for constraining deforming simulations is to "bind" or "embed" certain particles, removing them from the simulation and defining their positions in terms of other particles. This might be used to embed a high-resolution surface mesh in a lower-resolution volumetric cage for interactive performance [DDCB01, CGC*02] or to avoid difficult-to-handle "slivers" that appear in conforming meshes during fracture [MBF04]. Sifakis and colleagues [SSIF07] formalized much of the notation related to bound particles and showed that the Jacobian of the binding function is needed to transfer forces correctly. For simplicity, we will pose our constraints using bound particles (see §2); this limits somewhat the types of constraints we can define, but eases implementation. We emphasize, however, that the Jacobian calculation shown here could be used in a Lagrange multipliers framework, and this would allow certain types of constraints that cannot be expressed in a bindings framework (see also §5).

The methods listed above include many examples of "pin" constraints, point-on-circle constraints, contact constraints, and some constraints specifying internal dynamics (e.g., no-stretch constraints which fix the distance between particles). However, none of them extend between-body constraints beyond linear combinations of particles (point-to-triangle, point-tetrahedron, etc.), which means they suffer from the limitations highlighted in §1.

Using the Procrustes transformation in the context of simulation was pioneered by Müller and colleagues [MHTG05], who used it to simulate geometry of arbitrary topology. Rivers and James improved on the method's runtime by reusing computation [RJ07]. Both papers focused on shape matching for the internal dynamics of single objects. With sufficient stiffness, the earlier method could potentially be used as a penalty method constraining groups of points to move rigidly. However, this suffers from two major limitations as a constraint formulation. First, the timestepping method used is specialized for real-time simulations, while our method can be used within virtually any simulation

framework. Second, by effectively constraining points on *both* objects to move rigidly, we dramatically change the internal dynamics of each. Since we find that orthogonality between parameters improves usability, we prefer to define the constraint so that it affects the internal dynamics as little as possible.

2. Bound particle basics

As noted above, we will adopt the notion of hard-bound particles from Sifakis and colleagues [SSIF07] for its simplicity (extensions of our method to more complicated methods using Lagrange multipliers should follow naturally). In this framework the position of the "bound" particle $\mathbf{x}_b$ is defined to be a function of its parents, $\mathbf{x}_b = \phi(\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_n)$. Its velocity can be computed using the chain rule,

$$\frac{d\mathbf{x}_b}{dt} = \frac{d}{dt}\phi(\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_n) = \sum_{i=1}^{n} \frac{\partial \phi}{\partial \mathbf{x}_i}\frac{d\mathbf{x}_i}{dt} = \sum_{i=1}^{n} \frac{\partial \phi}{\partial \mathbf{x}_i}\mathbf{v}_i. \quad (1)$$

Sifakis and colleagues showed by applying the chain rule to the potential energy function that forces are distributed from child particles to parent particles via the *transpose* of the Jacobian,

$$\mathbf{f}_{\mathbf{x}_i} = \frac{\partial \phi}{\partial \mathbf{x}_i}^T \mathbf{f}_{\mathbf{x}_b}.$$

We adopt their notion of the *effective mass* m_e which measures the resistance of the bound particle to acceleration, $m_e = f_e/a_e$, and can be used to compute timestep restrictions. However, the formula must be modified slightly, since their derivation only handles the case where the binding uses scalar weights. We will need the second derivative of (1),

$$\begin{aligned}\frac{d^2\mathbf{x}_b}{dt^2} &= \frac{d}{dt}\left(\sum_{i=1}^{n} \frac{\partial \phi}{\partial \mathbf{x}_i}\mathbf{v}_i\right) \\ &= \sum_{i=1}^{n} \frac{\partial \phi}{\partial \mathbf{x}_i}\frac{d\mathbf{v}_i}{dt} + \frac{d}{dt}\left(\frac{\partial \phi}{\partial \mathbf{x}_i}\right)\mathbf{v}_i \\ &= \sum_{i=1}^{n} \frac{\partial \phi}{\partial \mathbf{x}_i}\mathbf{a}_i + \sum_{j=1}^{n}\left(\frac{\partial^2 \phi}{\partial \mathbf{x}_i \partial \mathbf{x}_j}\mathbf{v}_j\right)\mathbf{v}_i.\end{aligned}$$

Since this effective mass is used only for computing an approximate timestep restriction, a certain amount of error is acceptable, so we drop the second term here as it would be complicated to compute. Taking magnitudes where appropriate, we have,

$$\frac{\|\mathbf{f}_{\mathbf{x}_b}\|}{m_e} = \|\mathbf{a}_{\mathbf{x}_b}\| \approx \left\|\sum_{i=1}^{n} \frac{\partial \phi}{\partial \mathbf{x}_i}\mathbf{a}_i\right\| \leq \sum_{i=1}^{n}\left\|\frac{\partial \phi}{\partial \mathbf{x}_i}\mathbf{a}_i\right\| \quad (2)$$

$$= \sum_{i=1}^{n}\left\|\frac{\partial \phi}{\partial \mathbf{x}_i}\mathbf{M}_i^{-1}\mathbf{f}_{\mathbf{x}_i}\right\| = \sum_{i=1}^{n}\left\|\frac{\partial \phi}{\partial \mathbf{x}_i}\mathbf{M}_i^{-1}\frac{\partial \phi}{\partial \mathbf{x}_i}^T \mathbf{f}_{\mathbf{x}_b}\right\|. \quad (3)$$

Here, $\mathbf{M}_i$ is the diagonal mass matrix for the ith parent particle. Since we need only an upper bound on $1/m_e$ for computing the timestep restriction, we can conservatively estimate the effective mass using the largest singular value s_{max} of $d\phi/d\mathbf{x}_i$, that is, $m_e^{-1} \leq \sum_i s_{max}^2/m_i$. A consequence of (3) is that small masses for the parent particles (as for finely tesselated meshes) can lead to small timesteps, but our method can avoid this by averaging over more particles in the computation of $\phi(\mathbf{x}_i)$ (see §3).

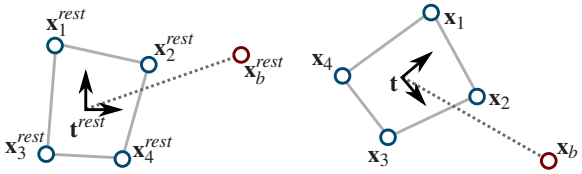

Figure 2: ***(Left)*** *Particle $\mathbf{x}_b$ is bound within the frame defined by the parent particles $\mathbf{x}_1^{rest}$, $\mathbf{x}_2^{rest}$, $\mathbf{x}_3^{rest}$, and $\mathbf{x}_4^{rest}$.* ***(Right)*** *At a subsequent frame, to determine the new location of $\mathbf{x}_b$, we find the nearest rigid transform $\{\mathbf{R}, \mathbf{t}\}$ taking $\mathbf{x}_i^{rest}$ to $\mathbf{x}_i$. We then apply the same translation and rotation to the offset vector $(\mathbf{x}_b^{rest} - \mathbf{t}^{rest})$.*

3. Defining $\phi(\mathbf{x}_i)$

First, let $\mathbf{x}_b^{rest}$ and $\mathbf{x}_i^{rest}$ be the rest positions of the bound and parent particles, respectively. The algorithm we use here is essentially the same one used for shape matching [MHTG05]. Define

$$\phi(\mathbf{x}_i) = \mathbf{R}(\mathbf{x}_b^{rest} - \mathbf{t}^{rest}) + \mathbf{t}, \quad (4)$$

where $\mathbf{t}, \mathbf{t}^{rest} \in \mathbb{R}^3$ and $\mathbf{R} \in SO(3)$ is a 3×3 rotation matrix. Intuitively, $\mathbf{t}$ and $\mathbf{R}$ are the translation and rotation that best fits the current *parent* particle positions $\mathbf{x}_i$; see Figure 2. To make this concrete, the problem is posed using least squares,

$$\mathbf{R}, \mathbf{t} = \arg\min_{\mathbf{R},\mathbf{t}} \sum_i w_i \left\|(\mathbf{R}(\mathbf{x}_i^{rest} - \mathbf{t}^{rest}) + \mathbf{t}) - \mathbf{x}_i\right\|_2^2. \quad (5)$$

Note that, as did Müller and colleagues, we have included a per-particle weight w_i to allow more control over the influence region. This can be useful because we can use a wider influence region to increase the stability of the computed transform while letting the weights fall off as a function of distance to ensure that far-off parent particles do not exert too strong an influence. We will assume that the weights have been normalized ($\sum_i w_i = 1$).

As in the earlier paper, $\mathbf{t}$ is simply the mean of the particles,

$$\mathbf{t} = \sum_i w_i \mathbf{x}_i \qquad \mathbf{t}^{rest} = \sum_i w_i \mathbf{x}_i^{rest}. \quad (6)$$

Computing the rotation matrix that minimizes (5) is an instance of the orthogonal Procrustes problem. The Procrustes problem can be solved using either the polar decomposition or the SVD [Hig08]. In graphics, the polar decomposition is commonly used as it can be computed quickly using simple-to-implement algorithms [SD92]. In our case, however, we

will need the singular values for calculating the Jacobian (§4.1), so it makes sense to compute the full SVD. Let $\mathbf{A}$ be the sum of outer products,

$$\mathbf{A} = \sum_i w_i (\mathbf{x}_i^{rest} - \mathbf{t}^{rest})(\mathbf{x}_i - \mathbf{t})^T. \tag{7}$$

Let $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ be the SVD, where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal and $\mathbf{S}$ is the diagonal matrix of positive singular values. The Procrustes transform is then $\mathbf{R} = \mathbf{U}\mathbf{V}$; however, we require that the resulting $\mathbf{R}$ have a positive determinant. Higham calls this the "rotation variant," and its solution can easily be computed from the SVD by defining the matrix $\mathbf{Z} = \mathrm{diag}(1, 1, \det(\mathbf{U}\mathbf{V}^\mathbf{T}))$ which can be used to flip the smallest singular value if the determinant of $\mathbf{U}\mathbf{V}^T$ is negative,

$$\mathbf{A} = (\mathbf{U}\mathbf{Z})(\mathbf{Z}\mathbf{S})\mathbf{V}^T \qquad \mathbf{R} = \mathbf{U}\mathbf{Z}\mathbf{V}^T.$$

A similar transformation was used by Irving and colleagues to ensure that tetrahedra in finite element simulations did not invert [ITF04]. Its effects will be discussed further in §5, but for now we will assume that any SVD that is discussed here has been adjusted such that $\mathbf{R} \in SO(3)$ is orthonormal with positive determinant.

Note that while there are fast methods for computing the polar decomposition based on Newton's method [Hig08], both Müller and colleagues [MHTG05] and Rivers and James [RJ07] instead use Jacobi iterations to diagonalize the matrix $\mathbf{A}^T\mathbf{A}$, which is more accurate for nearly-singular matrices. Our implementation uses two-sided Jacobi rotations to zero off-diagonal entries of $\mathbf{A}$ [GL96]. We do not currently take advantage of temporal coherence [RJ07], although this would be a relatively simple addition. The SVD implementation in LAPACK is a reasonable alternative [ABD*90]; while it takes roughly ten times longer to compute the 3×3 SVD than our optimized implementation, in typical simulations the number of constrained particles is small enough that computing the SVD is not a bottleneck. By contrast, a faster analytical version of the 3×3 SVD [Smi61] proved problematic in production, since accurate singular values are essential for the derivative calculation in §4.1.

4. Computing $d\phi/d\mathbf{x}_i$

Examining (4), it is clear that

$$\frac{\partial \phi(\mathbf{x}_i)}{\partial \mathbf{x}_i} = \frac{\partial \mathbf{R}}{\partial \mathbf{x}_i}(\mathbf{x}_b^{rest} - \mathbf{t}^{rest}) + \frac{\partial \mathbf{t}}{\partial \mathbf{x}_i}. \tag{8}$$

since $\mathbf{x}_b^{rest}$ and $\mathbf{t}^{rest}$ are constants. The second term is trivial to compute, since by (6) we have $d\mathbf{t}/d\mathbf{x}_i = w_i\mathbf{I}$ where $\mathbf{I}$ is the 3×3 identity matrix. The first term, however, is somewhat more challenging, since it involves differentiating through the Procrustes problem.

4.1. Differentiating the Procrustes transform

Recall that we defined the Procrustes transform using the SVD. The basics of differentiating the SVD have apparently been known for some time; Papadopoulo and Lourakis [PL00] give a very complete treatment whose notation we will borrow, but the basics can also be found in earlier works [Mat97]. We repeat the derivation here because we will need to make some important modifications specific to the Procrustes problem that ensure the method is well-behaved.

Let us simplify the problem by assuming that $\mathbf{A}(\theta)$ is a function of an arbitrary scalar parameter $\theta \in \mathbb{R}$. We can first observe that there exists a decomposition (the SVD),

$$\mathbf{A}(\theta) = \mathbf{U}(\theta)\mathbf{S}(\theta)\mathbf{V}^T(\theta).$$

where $\mathbf{U}$ and $\mathbf{V}^T$ are orthonormal ($\mathbf{U}\mathbf{U}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}$) and $\mathbf{S}$ is the diagonal matrix of singular values. Differentiating, we get

$$\frac{d\mathbf{A}}{d\theta} = \frac{d\mathbf{U}}{d\theta}\mathbf{S}\mathbf{V}^T + \mathbf{U}\frac{d\mathbf{S}}{d\theta}\mathbf{V}^T + \mathbf{U}\mathbf{S}\frac{d\mathbf{V}^T}{d\theta}.$$

Multiply on the left and right by $\mathbf{U}^T$ and $\mathbf{V}$, respectively, and by orthogonality

$$\mathbf{U}^T\frac{d\mathbf{A}}{d\theta}\mathbf{V} = \mathbf{U}^T\frac{d\mathbf{U}}{d\theta}\mathbf{S} + \frac{d\mathbf{S}}{d\theta} + \mathbf{S}\frac{d\mathbf{V}^T}{d\theta}\mathbf{V}. \tag{9}$$

Following Papadopoulo and Lourakis, we define (for convenience in notation)

$$\Omega_\mathbf{U} \stackrel{\text{def}}{=} \mathbf{U}^T\frac{d\mathbf{U}}{d\theta} \qquad \Omega_\mathbf{V} \stackrel{\text{def}}{=} \frac{d\mathbf{V}^T}{d\theta}\mathbf{V}. \tag{10}$$

Substitute (10) into (9) to get

$$\mathbf{U}^T\frac{d\mathbf{A}}{d\theta}\mathbf{V} = \Omega_\mathbf{U}\mathbf{S} + \frac{d\mathbf{S}}{d\theta} + \mathbf{S}\Omega_\mathbf{V}. \tag{11}$$

Note that everything on the left-hand side is known and the values on the right-hand side are the unknowns. There are two key points to note:

a. $\frac{d\mathbf{S}}{d\theta}$ is diagonal, and
b. $\Omega_\mathbf{U}$ and $\Omega_\mathbf{V}$ are skew-symmetric (antisymmetric).

(a) follows trivially since $\mathbf{S}$ is always diagonal. (b) results from the fact that $\mathbf{U}$ and $\mathbf{V}^T$ are orthogonal,

$$\begin{aligned}
\mathbf{U}^T\mathbf{U} &= \mathbf{I} \\
\frac{d}{d\theta}(\mathbf{U}^T\mathbf{U}) &= \mathbf{0} \\
\frac{d\mathbf{U}}{d\theta}^T\mathbf{U} + \mathbf{U}^T\frac{d\mathbf{U}}{d\theta} &= \mathbf{0} \\
\Omega_\mathbf{U}^T + \Omega_\mathbf{U} &= \mathbf{0}.
\end{aligned}$$

Since $\mathbf{R} = \mathbf{U}\mathbf{V}^T$ does not use $\mathbf{S}$, we will not need to compute $\frac{d\mathbf{S}}{d\theta}$. Since $\Omega_\mathbf{U}$ and $\Omega_\mathbf{V}$ have zeroes along the diagonal, we can ignore the diagonal entries in (11). For off-diagonal

entries $i \neq j$:

$$\left[\mathbf{U}^T \frac{d\mathbf{A}}{d\theta}\mathbf{V}\right]_{ij} = [\Omega_\mathbf{U}]_{ij}\mathbf{S}_j + \mathbf{S}_i[\Omega_\mathbf{V}]_{ij}.$$

Here, Papadopoulo and Lourakis set up three pairs of equations, one for each entry in the lower triangle (remember that $[\Omega_\mathbf{U}]_{ij} = -[\Omega_\mathbf{U}]_{ji}$ by skew-symmetry). Each has the form,

$$\begin{pmatrix}\mathbf{S}_j & \mathbf{S}_i \\ \mathbf{S}_i & \mathbf{S}_j\end{pmatrix}\begin{pmatrix}[\Omega_\mathbf{U}]_{ij} \\ [\Omega_\mathbf{V}]_{ij}\end{pmatrix} = \begin{pmatrix}\left[\mathbf{U}^T \frac{d\mathbf{A}}{d\theta}\mathbf{V}\right]_{ij} \\ -\left[\mathbf{U}^T \frac{d\mathbf{A}}{d\theta}\mathbf{V}\right]_{ji}\end{pmatrix}. \quad (12)$$

Each such pair can be solved for $[\Omega_\mathbf{U}]_{ij} = -[\Omega_\mathbf{U}]_{ji}$ and $[\Omega_\mathbf{V}]_{ij} = -[\Omega_\mathbf{V}]_{ji}$. However, we run into difficulty for repeated singular values $\mathbf{S}_j = \mathbf{S}_i = s$, as the matrix becomes singular, and with round-off error we expect high levels of error for near-repeated eigenvalues. Papadopoulo and Lourakis suggest solving this robustly using the SVD, but this risks introducing difficulties in choosing the cutoff value, and further seems counterintuitive: the case where the singular values are $(1,1,1)$ should ideally be the *easiest* to solve.

Fortunately, this turns out to be simple to remedy. Remember that what we really want to compute is the derivative of the best-fit Procrustes transform,

$$\frac{d}{d\theta}(\mathbf{R}) = \frac{d}{d\theta}\left(\mathbf{U}\mathbf{V}^T\right) = \frac{d\mathbf{U}}{d\theta}\mathbf{V}^T + \mathbf{U}\frac{d\mathbf{V}^T}{d\theta} \quad (13)$$

$$= \mathbf{U}\Omega_\mathbf{U}\mathbf{V}^T + \mathbf{U}\Omega_\mathbf{V}\mathbf{V}^T \quad (14)$$

$$= \mathbf{U}(\Omega_\mathbf{U} + \Omega_\mathbf{V})\mathbf{V}^T \quad (15)$$

Thus, we need only to compute the *sum* $\Omega_\mathbf{U} + \Omega_\mathbf{V}$, rather than the individual components. If we add the first line of (12) to the second, we get

$$(\mathbf{S}_j + \mathbf{S}_i)\left([\Omega_\mathbf{U}]_{ij} + [\Omega_\mathbf{V}]_{ij}\right) = \left(\left[\mathbf{U}^T \frac{d\mathbf{A}}{d\theta}\mathbf{V}\right]_{ij} - \left[\mathbf{U}^T \frac{d\mathbf{A}}{d\theta}\mathbf{V}\right]_{ji}\right). \quad (16)$$

We can thus compute $\Omega_\mathbf{U} + \Omega_\mathbf{V}$ whenever no two singular values sum to zero. Normally, singular values are defined to be nonnegative, so we would expect this to hold whenever two or more singular values are nonzero. As we allowed the smallest singular value to be negative in §3, however, the two smallest singular values could potentially cancel each other out. Either case is already problematic for Procrustes, so this is a reasonable criterion (if there are two near-zero singular values, the solution is unconstrained and free to rotate arbitrarily about the third axis; when there is cancellation we cannot decide which of the directions needs to be flipped to ensure a right-handed coordinate system).

Completing the derivative Now that we can differentiate $\mathbf{R}$ as a function of an arbitrary parameter θ, we can use this to differentiate $\mathbf{R}$ with respect to $\mathbf{x}_i$. Let us first define some notation to simplify things; let $\mathbf{v}_b = \mathbf{x}_b^{rest} - \mathbf{t}^{rest}$ be the bound particle's offset vector in its rest configuration and

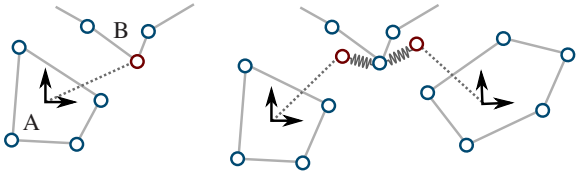

Figure 3: ***(Left)*** *If a user wants to constrain a single particle in object B to the frame defined by object A, we can use a "hard binding" in the parlance of Sifakis et al., replacing the specified particle of B with one (in red) whose position is defined by the Procrustes transform fit to A.* ***(Right)*** *If the user specifies multiple constraints for a single particle, we instead create two new bound particles and attach them to the constrained particle via stiff implicitly integrated springs, similar to the binding springs of Sifakis et al.*

$\mathbf{v}_i = \mathbf{x}_i^{rest} - \mathbf{t}^{rest}$ be the corresponding vector for the parent particle. For clarity, we can rewrite (4) in terms of individual components,

$$\phi_j(\mathbf{x}_i) = \sum_{k=1}^{3} \mathbf{R}_{jk}\mathbf{v}_{bk} + \mathbf{t}_j.$$

Now,

$$\frac{\partial\phi_j}{\partial\mathbf{x}_{i\ell}} = \sum_{k=1}^{3}\frac{\partial\mathbf{R}_{jk}}{\partial\mathbf{x}_{i\ell}}\mathbf{v}_{bk} + \frac{\partial\mathbf{t}_j}{\partial\mathbf{x}_{i\ell}}. \quad (17)$$

From (6) we have $d\mathbf{t}_j/d\mathbf{x}_{i\ell} = w_i\delta_{j\ell}$ where $\delta_{j\ell}$ is the Kronecker delta function. Now, (16) tells us how to compute $d\mathbf{R}_{jk}/d\mathbf{x}_{i\ell}$ given $d\mathbf{A}_{jk}/d\mathbf{x}_{i\ell}$. Combining (7) with (6),

$$\begin{aligned}\mathbf{A}_{jk} &= \textstyle\sum_m w_m\mathbf{v}_{mj}(\mathbf{x}_{mk} - \mathbf{t}_k) \\ &= \textstyle\sum_m w_m\mathbf{v}_{mj}(\mathbf{x}_{mk} - \sum_n w_n\mathbf{x}_{nk}),\end{aligned}$$

which can be differentiated

$$\begin{aligned}\frac{\partial\mathbf{A}_{jk}}{\partial\mathbf{x}_{i\ell}} &= \textstyle\sum_m w_m\mathbf{v}_{mj}(\delta_{im}\delta_{kl} - \sum_n w_n\delta_{in}\delta_{k\ell}) \\ &= w_i\delta_{k\ell}\left(\mathbf{v}_{ij} - \textstyle\sum_m w_m\mathbf{v}_{mj}\right).\end{aligned}$$

However, $\sum_m w_m\mathbf{v}_m = \sum_m w_m(\mathbf{x}_m^{rest} - \mathbf{t}_m^{rest}) = \sum_m w_m(\mathbf{x}_m^{rest} - \sum_n w_n\mathbf{x}_n^{rest}) = \mathbf{0}$ (since $\sum_m w_m = 1$), so we can drop the second term. As a result, each of the n Jacobian terms (one per parent particle) can be computed with only a constant amount of work. In practice, we generally limit the number of parent particles to ten or fewer, and computing the SVD accounts for the bulk of the computational cost.

4.2. Observations

At this point, it is worth taking a moment to examine how the form the derivatives take ensures that the point cloud glue avoids the problems inherent in linear bindings. Recall that we transfer forces from child to parent particles via the Ja-

cobian matrix (this is just equation (17)):

$$\frac{\partial \phi_j}{\partial \mathbf{x}_{i\ell}} = \underbrace{\sum_{k=1}^{3} \frac{\partial \mathbf{R}_{jk}}{\partial \mathbf{x}_{i\ell}} \mathbf{v}_{bk}}_{\text{rotation}} + \underbrace{\frac{\partial \mathbf{t}_j}{\partial \mathbf{x}_{i\ell}}}_{\text{translation}} . \tag{18}$$

The derivative of the translation term simply transfers the force to each parent particle i scaled by its weight w_i. We can thus think of this as applying the force to the "center of mass" defined by the weights w_j.

Now, consider the rotation term; $\partial \mathbf{R}_{jk}/\partial \mathbf{x}_{i\ell}$ must take the form given by (15),

$$\sum_{k=1}^{3} \frac{\partial \mathbf{R}_{jk}}{\partial \mathbf{x}_{i\ell}} \mathbf{v}_{bk} = \sum_{k=1}^{3} \mathbf{U} \left(\Omega_{\mathbf{U}}^{k} + \Omega_{\mathbf{V}}^{k} \right) \mathbf{V}^T \mathbf{v}_{bk}. \tag{19}$$

Here we are not concerned with the actual values of $\Omega_{\mathbf{U}}^{k}$ and $\Omega_{\mathbf{V}}^{k}$; merely that they are both skew-symmetric. Since $\mathbf{U}$ and $\mathbf{V}$ are held constant during the derivative calculation, we can factor them out,

$$\sum_{k=1}^{3} \frac{\partial \mathbf{R}_{jk}}{\partial \mathbf{x}_{i\ell}} \mathbf{v}_{bk} = \mathbf{U} \left(\sum_{k=1}^{3} \mathbf{v}_{bk} \left(\Omega_{\mathbf{U}}^{k} + \Omega_{\mathbf{V}}^{k} \right) \right) \mathbf{V}^T. \tag{20}$$

It is easy to verify that the weighted sum of skew-symmetric matrices is skew-symmetric, so this has the form $\mathbf{U} * (\text{skew-symmetric}) * \mathbf{V}^T$. When applied to a force, therefore, the (transposed) rotation term first rotates by $\mathbf{U}^T$, multiplies by a skew-symmetric matrix, and finally rotates again by $\mathbf{V}$. Since multiplying a vector $\mathbf{y}$ by a skew-symmetric matrix $[\mathbf{z}_\times]$ corresponds to computing the cross product $\mathbf{z} \times \mathbf{y}$, we can consider this first term to be analogous to the $\mathbf{r}_1 \times (\mathbf{r}_2 \times \mathbf{f})$ term that converts point forces to rigid body torques and back to point forces. Our analogy is not exact, of course, because the actual skew-symmetric matrices depend on other factors including the weights and computed singular values, but in the special case where all particles are in their "rest configuration," meaning $\mathbf{U} = \mathbf{V} = \mathbf{S} = \mathbf{I}$, we will find that

$$\sum_{k=1}^{3} \frac{\partial \mathbf{R}_{jk}}{\partial \mathbf{x}_{i\ell}} \mathbf{v}_{bk} = [(\mathbf{v}_i \times \mathbf{v}_b)_\times] \tag{21}$$

That is, transferring a force $\mathbf{f}_{\mathbf{x}_b}$ on the bound particle to a force $\mathbf{f}_{\mathbf{x}_i}$ on the parent particle i means first converting it to a "torque" $\tau = \mathbf{v}_b \times \mathbf{f}_{\mathbf{x}_b}$ and then applying the "torque" to the parent particle via $\mathbf{f}_{\mathbf{x}_i} = \mathbf{v}_i \times \tau$.

5. Results and Discussion

We implemented the point cloud glue within a simulation engine that handles cloth, flesh, hair and rigid bodies and which already supported bindings similar to those described by Sifakis and colleagues [SSIF07]. We made no changes to the engine itself, and we emphasize that this method should be usable within virtually any simulation framework. Users of our system simply specify the points to constrain and the target (parent) object(s); both constrained points and target objects can be either rigid or deformable. Because our method is topology-agnostic, constraints can be applied equally well to cloth and tetrahedral meshes without any additional implementation effort.

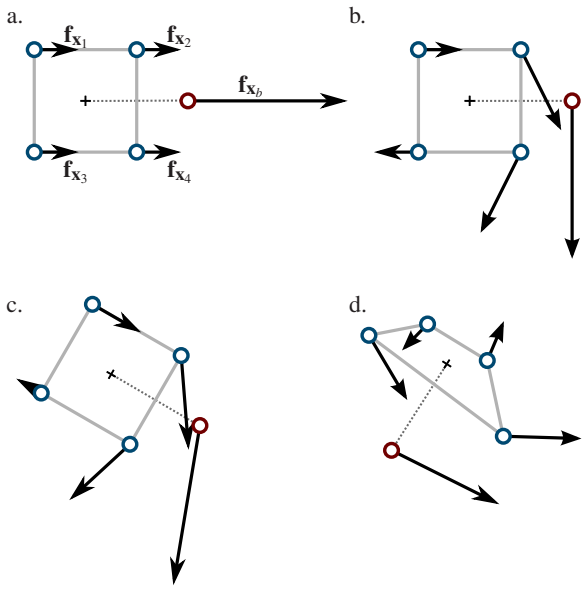

Figure 4: *Forces $\mathbf{f}_{\mathbf{x}_b}$ on the bound particle $\mathbf{x}_b$ are transferred to the parent particles $\mathbf{x}_i$ via the constraint Jacobian. In **(a)**, the force points along the "moment arm" and hence pushes all the parent particles in the same direction (this makes sense, since an acceleration in this direction only affects the translation part of the Procrustes transform). In **(b)**, the same force rotated by 90 degrees is applied to the parent particles with the characteristic "windmill" pattern caused by the skew-symmetric terms discussed in §4.2. In **(c)** we see a combination of these two terms. Of course, in real-world examples we expect to see significant deformation of the parent particles as in **(d)**; the fitted rigid frame is represented by a +. Note that while we may see shearing in the parent particles as in (b) and (d), our method never drives particles violently apart as do the linear bindings in Figure 1.*

For each constrained point, the system automatically selects enough nearby points (typically fewer than ten) on the target object to generate a sufficient basis for fitting the Procrustes transform. Users can select whether the constraint should be maintained exactly, in which case we use a "hard binding" as described in Sifakis et al. [SSIF07] (Figure 3, left), or whether a penalty-based method is preferred, in which case we substitute stiff implicitly integrated springs between $\mathbf{x}_b$ and the constrained point. Conflicting constraints on a given particle are handled using stiff springs

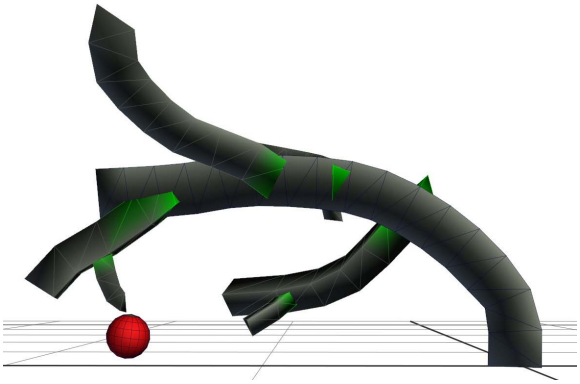

Figure 5: *"Branches" here are simulated using tetrahedral meshes; the points highlighted in green are connected to parent branches via the point cloud glue, and the rigid spherical "ornament" is attached via the same mechanism. Note that points can be outside the parent tetrahedral cage without affecting stability. This simulation runs at 0.2s/frame, of which only 1.5% is spent computing Procrustes derivatives and an additional 1.6% is spent in SVD computation.*

Figure 6: *For this shirt simulation, Point Cloud Glue is used to attach both the deforming pocket and the rigid buttons to the shirt. Point cloud glue also constrains the shirt to itself at the locations of the buttons to ensure that it stays closed. We attach the top two buttons after the first few frames of simulation to let the character's head pass through the collar by adjusting the spring stiffness on the appropriate frames; later, we detach the glue at the cuffs (reducing stiffness to zero) to simulate unbuttoning. This simulation runs at between 5 and 6 minutes per frame and is completely dominated by collision handling; SVD and Procrustes derivative computation combined account for less than 0.1% of the runtime.*

(Figure 3, right); in a system using Lagrange multipliers, least squares might be a better alternative. Points on rigid bodies are likewise constrained by attaching the constrained points to $\mathbf{x}_b$ with springs; applying the constraint exactly would require interfacing directly with the rigid constraint solver, although in principle this should be possible.

Because we always enforce a positive determinant (§3) for the fitted transform by potentially negating the smallest singular value, we only need two well-defined basis vectors for good results, which can be provided by any three points on the parent object that are not collinear. Thus, our system works well for both cloth simulations and volumetric flesh simulations, and constrained points can be anywhere in space (not just on the cloth surface). More points improve the stability of the fitted transform in the presence of any high frequency motion of the points by averaging over a larger area (essentially, leveraging the central limit theorem). For cloth, the orientation provided by the Procrustes transform ensures that constrained points remain on the correct side of the surface, which is difficult to guarantee with penalty forces.

Plant simulation While there are a variety of methods that can be used for simulating plants, it may be convenient to use tetrahedral meshes to represent branches and stems. Figure 5 shows one such simulation; we used point cloud glue to attach each branch to its parent branch and again to attach the rigid "ornament." Note that the constrained points of the child tetrahedral mesh need not be inside the parent, which simplified the setup. Figure 7 shows a more complicate production simulation; branches were attached using the glue which was disabled at the appropriate frames to allow them to fall off as the vehicle crashes through them.

Cloth simulation Figure 6 shows a shirt where rigid buttons and pockets have all been attached to the main shirt using point cloud glue. Note how we can use the glue to keep the buttoned layers slightly separated to ensure the correct Z-order; if we used a point-to-triangle constraint as described in Figure 1 we would find that the two layers would be coincident at the constrained points, and if we attached them via springs instead and relied on collision handling to keep the layers apart we would find that the the stiff springs defeat the repulsion phase of the collision handling algorithm [BFA02], necessitating reliance on later phases of the collision algorithm that tend to be slower and can produce artifacts. Note that the glue does not affect the relationships between the parent particles, meaning that clothing can still remain soft and pliable even as buttons and pockets are attached.

Embeddings Dealing with embedded meshes can sometimes be painful for users, since they must expend effort ensuring that the embedded mesh is fully contained within the volume mesh. This is especially problematic in production, since tetrahedral cages may need be regenerated from scratch due to only minor modeling changes to the surface mesh. We can avoid some of these problems by automatically detecting that points are outside the volume mesh

and switching them over from barycentric coordinate-based bindings to point cloud glue-based bindings.

Limitations A major limitation of our work is common to all methods based on the Procrustes transform: if the parent points lie in a line, we only have one well-defined direction, and the resulting transform is under-constrained. This means that it is suitable for simulations of cloth and flesh, but not when simulating hair, for example (although it could be used with hair simulations based on tetrahedral meshes [SLF08]). Similarly, even if we start with two or three well-defined directions, our parent mesh could collapse to a line during simulation, causing the fitted transform to have an extra degree of freedom. Thus, we rely on the internal dynamics of the deforming objects to maintain enough of the initial shape for a well-defined transform. In practice, we find that this is rarely a problem.

Future work As mentioned above, our method could be easily adapted for use in a constraint framework based on Lagrange multipliers. Here, constraints can be more flexible, and we could even define "joints" between deformable objects via constructions like $c(\mathbf{x}_i, \mathbf{y}_i) = \phi(\mathbf{x}_i) - \psi(\mathbf{y}_i) = 0$ (here, both ϕ and ψ represent positions defined by the Procrustes transform).

Because the Procrustes transform is a very useful construct, we suspect that the ability to differentiate it will have uses elsewhere. For example, imagine adding a term to a geometric optimization of the form "minimize the difference between a set of points and the Procrustes transform fitted to those points." Minimizing this efficiently would involve differentiating through the Procrustes transform computation.

Acknowledgements The authors would like to thank the anonymous reviewers for their comments; Don Hatch, Stephen Bowline, and Roni Goldenthal for helpful discussions; and David Deuber, Karin Cooper, and Seunghun Lee for their work testing and improving the software.

Figure 7: *In this production example from the movie Avatar, branches are again simulated as tetrahedral meshes and glued together with point cloud glue. At appropriate frames, the glue is keyed to release to produce the desired result. Simulation by Seunghun Lee. Image TM and ©2009 Twentieth Century Fox Film Corp. All rights reserved.*

References

[ABD*90] ANDERSON E., BAI Z., DONGARRA J., GREENBAUM A., MCKENNEY A., DU CROZ J., HAMMERLING S., DEMMEL J., BISCHOF C., SORENSEN D.: LAPACK: a portable linear algebra library for high-performance computers. In *Proceedings of the 1990 ACM/IEEE conference on Supercomputing* (1990).

[BB88] BARZEL R., BARR A. H.: A modeling system based on dynamic constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)* (Aug. 1988), pp. 179–188.

[BFA02] BRIDSON R., FEDKIW R. P., ANDERSON J.: Robust treatment of collisions, contact, and friction for cloth animation. *ACM Transactions on Graphics 21*, 3 (July 2002), 594–603.

[BW92] BARAFF D., WITKIN A.: Dynamic simulation of non-penetrating flexible bodies. In *Computer Graphics (Proceedings of SIGGRAPH 92)* (July 1992), pp. 303–308.

[BW98] BARAFF D., WITKIN A. P.: Large steps in cloth simulation. In *Proceedings of SIGGRAPH 98* (July 1998), pp. 43–54.

[CGC*02] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIĆ Z.: Interactive skeleton-driven dynamic deformations. *ACM Transactions on Graphics 21*, 3 (July 2002), 586–593.

[DDCB01] DEBUNNE G., DESBRUN M., CANI M.-P., BARR A. H.: Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of ACM SIGGRAPH 2001* (Aug. 2001), pp. 31–36.

[GBT06] GISSLER M., BECKER M., TESCHNER M.: Local constraint methods for deformable objects. In *Proc. of the 3rd Workshop in VR Interactions and Physical Simulation (VRIPHYS)* (2006), pp. 1–8.

[GL96] GOLUB G. H., LOAN C. F. V.: *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, 1996.

[Hig08] HIGHAM N. J.: *Functions of Matrices: Theory and Computation*. SIAM, 2008.

[ITF04] IRVING G., TERAN J., FEDKIW R.: Invertible finite elements for robust simulation of large deformation. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (July 2004), pp. 131–140.

[Mat97] MATHAI A. M.: *Jacobians of Matrix Transformations and Functions of Matrix Argument*. World Scientific Publishing Co. Pte. Ltd., 1997.

[MBF04] MOLINO N., BAO Z., FEDKIW R.: A virtual node algorithm for changing mesh topology during simulation. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 385–392.

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. *ACM Transactions on Graphics 24*, 3 (Aug. 2005), 471–478.

[MT92] METAXAS D., TERZOPOULOS D.: Dynamic deformation of solid primitives with constraints. In *Computer Graphics (Proceedings of SIGGRAPH 92)* (July 1992), pp. 309–312.

[MTG04] MÜLLER M., TESCHNER M., GROSS M.: Physically-based simulation of objects represented by surface meshes. In *Computer Graphics International 2004* (June 2004), pp. 26–33.

[MW88] MOORE M., WILHELMS J.: Collision detection and response for computer animation. In *Computer Graphics (Proceedings of SIGGRAPH 88)* (Aug. 1988), pp. 289–298.

[PB88] PLATT J. C., BARR A. H.: Constraint methods for flexible models. In *Computer Graphics (Proceedings of SIGGRAPH 88)* (Aug. 1988), pp. 279–288.

[PL00] PAPADOPOULO T., LOURAKIS M. I.: Estimating the Jacobian of the singular value decomposition: Theory and applications. In *Proceedings of the 6th European Conference on Computer Vision* (2000), pp. 554–570.

[RJ07] RIVERS A. R., JAMES D. L.: FastLSM: Fast lattice shape matching for robust real-time deformation. *ACM Transactions on Graphics 26*, 3 (July 2007), 82:1–82:6.

[SD92] SHOEMAKE K., DUFF T.: Matrix animation and polar decomposition. In *Graphics Interface '92* (May 1992), pp. 258–264.

[SLF08] SELLE A., LENTINE M., FEDKIW R.: A mass spring model for hair simulation. *ACM Transactions on Graphics 27*, 3 (Aug. 2008), 64:1–64:11.

[Smi61] SMITH O. K.: Eigenvalues of a symmetric 3 x 3 matrix. *Commun. ACM 4*, 4 (1961), 168.

[SSIF07] SIFAKIS E., SHINAR T., IRVING G., FEDKIW R.: Hybrid simulation of deformable solids. In *2007 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Aug. 2007), pp. 81–90.

[WFB87] WITKIN A., FLEISCHER K., BARR A.: Energy constraints on parameterized models. In *Computer Graphics (Proceedings of SIGGRAPH 87)* (July 1987), pp. 225–232.

[WTF06] WEINSTEIN R., TERAN J., FEDKIW R.: Dynamic simulation of articulated rigid bodies with contact and collision. *IEEE Transactions on Visualization and Computer Graphics 12*, 3 (May/June 2006), 365–374.

[WW90] WITKIN A., WELCH W.: Fast animation and control of nonrigid structures. In *Computer Graphics (Proceedings of SIGGRAPH 90)* (Aug. 1990), pp. 243–252.