A Robust Method to Extract the Rotational Part of Deformations

Matthias Müller[1] Jan Bender[2] Nuttapong Chentanez[1] Miles Macklin[1]

[1]NVIDIA Physics Research [2] RWTH Aachen University

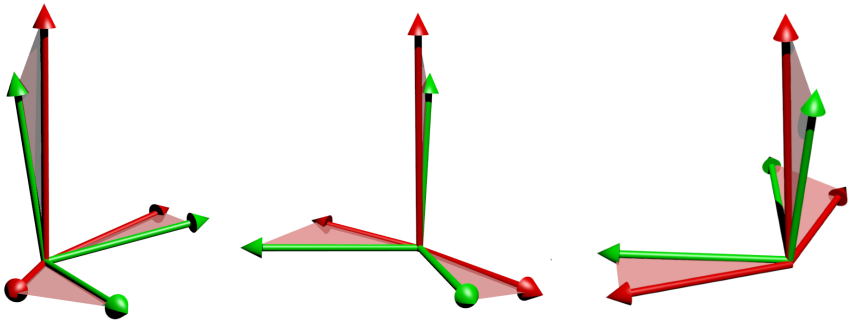

Figure 1: *The basic idea behind our method. The red arrows show the column vectors of an input matrix* **A** *and the green arrows the column vectors of the extracted rotation matrix* **R**. *The optimal rotation matrix is in an equilibrium state in which the momenta on all axes (red triangles) cancel out.*

Abstract

We present a novel algorithm to extract the rotational part of an arbitrary 3×3 matrix. This problem lies at the core of two popular simulation methods in computer graphics, the co-rotational Finite Element Method and Shape Matching techniques. In contrast to the traditional method based on polar decomposition, degenerate configurations and inversions are handled robustly and do not have to be treated in a special way. In addition, our method can be implemented with only a few lines of code without branches which makes it particularly well suited for GPU-based applications. We demonstrate the robustness, coherence and efficiency of our method by comparing it to stabilized polar decomposition in several simulation scenarios.

Keywords: rotation, shape matching, co-rotational FEM, polar decomposition

1 Introduction

There are two main challenges when simulating deformable objects, namely handling degenerate configurations and inversions. In the context of the co-rotational Finite Element Method [Müller and Gross 2004; Hauth and Strasser 2004] and simulation methods based on shape matching [Müller et al. 2005] these challenges correspond to the problem of extracting a proper rotation matrix from an arbitrary deformation gradient. More formally: Let $\mathbf{A}$ be an arbitrary 3×3 matrix. Find an ortho-normal 3×3 matrix $\mathbf{R}$ with $\det(\mathbf{R}) = +1$ which minimizes the Frobenius-norm $||\mathbf{A} - \mathbf{R}||_F^2$.

MiG '16,, October 10-12, 2016, Burlingame, CA, USA
ISBN: 978-1-4503-4592-7/16/10
DOI: http://dx.doi.org/10.1145/2994258.2994269

This problem known as the Wahba problem has been studied over several decades since the early work of Grace Wahba [1965]. It was first studied in connection with the task of deriving the orientation of a space craft from a set of measured direction vectors toward astronomical objects.

The most popular approach to solving this problem has been to extract a rotation matrix from $\mathbf{A}$ via polar decomposition $\mathbf{A} = \mathbf{RU}$ and was first proposed by Farrell et al. [1966]. The problem with this approach is that the polar decomposition is not defined if $\mathbf{A}$ is singular and yields a reflecting matrix $\mathbf{R}$ with $det(\mathbf{R}) = -1$ if $det(\mathbf{A}) < 0$. A variety of ideas have been proposed to fix $\mathbf{R}$ via reflecting it along carefully chosen directions.

Another family of methods determine $\mathbf{R}$ via a constrained optimization problem minimizing the Frobenius norm between $\mathbf{R}$ and $\mathbf{A}$ under the constraint $det(\mathbf{R}) = +1$. When $\mathbf{R}$ is parametrized by a quaterion $\mathbf{q}$, the optimization problem is reduced to finding the largest eigenvector of a 4×4 linear system. Solving a 4×4 eigenvalue problem for each element at every time step can easily become the bottle neck of a simulation, however.

In this paper we propose a new method that is both efficient and robust for any matrix $\mathbf{A}$.

2 Related Work

One of the simplest ways to extract a rotation matrix $\mathbf{R}$ from an arbitrary matrix $\mathbf{A}$ is using Gram-Schmidt orthogonalization. Here, the first column of $\mathbf{R}$ is set to the first column of $\mathbf{A}$ and normalized. The second column of $\mathbf{R}$ is set to the second column of $\mathbf{A}$ minus its projection along the first column and normalized. Finally, the third column is set to be the cross product of the first two. The resulting rotation matrix is ortho-normal with determinant +1 by construction. It also works for singular matrices as long as two directions are determined because the third column is derived from the first two directly. The main disadvantage of this method is the fact that the result depends on the order in which the columns of $\mathbf{A}$ are processed. This bias introduces ghost forces in simulations because momentum is not conserved in general.

Before we turn to polar decomposition which is it the most pop-

ular approach in computer graphics, we will mention a few other methods that have been used in other fields.

Wessner [1966] derived the formula $\mathbf{R} = \left(\mathbf{A}^T\right)^{-1}\left(\mathbf{A}^T\mathbf{A}\right)^{\frac{1}{2}}$ to extract a rotation matrix from $\mathbf{A}$. This formula can be transformed into the form $\mathbf{R} = \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-\frac{1}{2}}$ which is the common way of computing the polar decomposition as we will describe and discuss below.

Higham [1986] proposed an iterative solution. The iteration is initialized by setting $\mathbf{R} \leftarrow \mathbf{A}$. During the iteration the intermediate solution is improved by evaluating $\mathbf{R} \leftarrow \frac{1}{2}\left(\mathbf{R} + \mathbf{R}^{-T}\right)$. This process converges to an $\mathbf{R}$ with $det(\mathbf{R}) = -1$ if $det(\mathbf{A}) < 0$ and only works for a non-singular $\mathbf{A}$.

Davenport's q-method [Markley and Mortari 1999] introduced the idea of parametrizing $\mathbf{R}$ with a quaternion $\mathbf{q}$. As described in the introduction, the minimizing quaternion can be determined by finding the eigenvector corresponding to the largest eigenvalue of a 4×4 linear system. The need to solve a 4×4 eigenvalue problem makes this approach significantly slower than other methods.

The most popular methods to extract rotations from deformation gradients in computer graphics are based on the polar decomposition of the deformation gradient $\mathbf{A} = \mathbf{RS}$. Here $\mathbf{A}$ is decomposed into a rotation matrix $\mathbf{R}$ and a symmetric matrix $\mathbf{S}$. The same rotation matrix is obtained with the singular value decomposition (SVD) which splits $\mathbf{A}$ into the three parts $\mathbf{A} = \mathbf{UDV}^T$ by choosing $\mathbf{R} = \mathbf{UV}^T$. In this decomposition, the diagonal matrix $\mathbf{D}$ contains the singular values of $\mathbf{A}$ on its diagonal. Inversions and singularities manifest themselves as negative or zero singular values and can be fixed by manipulating the corresponding column vectors in $\mathbf{U}$ and $\mathbf{V}$.

Irving et al. [2004] were among the first in computer graphics to address the inversion problem. In case of an inversion, they identify the smallest singular value and invert the corresponding column vector in the resulting rotation matrix. The idea is to reflect an element along the minimal inversion direction. Schmedding et al.[2008] reflect the element in the direction for which vertices have to travel the shortest distance which, as they note, does not necessarily correspond to the minimal inversion direction, especially for badly shaped elements. As in continuous collision detection Civit-Flores and Susin [2012] determine the point in time when an element inverts by solving a cubic equation to choose how to undo the inversion. Their paper contains an excellent overview and analysis of existing methods for handling degenerate elements and inversions.

3 The Method

We start with stating the problem we want to solve again: Let $\mathbf{A}$ be an arbitrary 3×3 matrix. Our goal is to find a rotation matrix that is as close as possible to $\mathbf{A}$ or more technically, an ortho-normal 3×3 matrix $\mathbf{R}$ with $\det(\mathbf{R}) = +1$ which minimizes the Frobenius-norm $F(\mathbf{R}) = ||\mathbf{A} - \mathbf{R}||^2$.

Our novel idea is simple and effective: Instead of solving for $\mathbf{R}$ directly we assume that we already have an approximation either from the previous time step of a simulation or from the previous step of an iterative solve. We are looking for an update rule of the form

$$\mathbf{R} \leftarrow exp(\omega)\,\mathbf{R}, \tag{1}$$

where $exp(\omega)$ is the rotation matrix with axis $\omega/|\omega|$ and angle $|\omega|$ also known as the exponential map. The exponential map is guaranteed to be a proper rotation matrix, i.e. to be orthonormal with determinant +1 even for $\omega = \mathbf{0}$ in which case it assumes the identity. Therefore, if $\mathbf{R}$ is a proper rotation matrix, this property is conserved by the update.

The first question we need to answer is how to choose the direction of ω. The best choice is the one for which the Frobenius norm decreases the most. To find this direction we interpret our problem in a physical way. Let $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ be the column vectors of $\mathbf{A}$ and $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$ be the column vectors of $\mathbf{R}$. The matrix $\mathbf{A}$ is fixed and can be interpreted as a static object. The matrix $\mathbf{R}$ is only allowed to rotate about the origin and can therefore be interpreted as a rigid body. Since we want to minimize F we define the energy

$$E_F = \frac{1}{2}F(\mathbf{R}) = \frac{1}{2}\sum_i (\mathbf{r}_i - \mathbf{a}_i)^2. \tag{2}$$

We will choose ω to be the axis about which the rigid body $\mathbf{R}$ gets accelerated under the energy E_F. The forces acting at the tips of the axes of $\mathbf{R}$ are

$$\mathbf{f}_i = -\frac{\partial E_F}{\partial \mathbf{r}_i} = \mathbf{a}_i - \mathbf{r}_i. \tag{3}$$

The total torque acting on $\mathbf{R}$ due these forces is

$$\tau = \sum_i \mathbf{r}_i \times (\mathbf{a}_i - \mathbf{r}_i) = \sum_i \mathbf{r}_i \times \mathbf{a}_i. \tag{4}$$

Therefore, we choose $\omega = \alpha\tau$ for some scalar α. The introduction of τ allows a new interpretation of the matrix $\mathbf{R}$ that minimizes the Frobenius norm and the matrix $\mathbf{R}$ obtained by a polar decomposition of $\mathbf{A}$. In the appendices A and B we show that $\tau = \mathbf{0}$ for both which means they can be interpreted to be the matrices that are in an equilibrium state w.r.t. E_F (see Figure 1).

The second questions is how to choose α. We first look at a single pair $\mathbf{a}_i$ and $\mathbf{r}_i$. If we choose $\omega = \mathbf{a}_i \times \mathbf{r}_i$ then the magnitude of the rotation $|\omega|$ is $|\mathbf{a}_i||\mathbf{r}_i|\sin\phi$, where ϕ is the angle between the two vectors. However, our goal is to align $\mathbf{a}_i$ and $\mathbf{r}_i$ to make $\tau = 0$ which is accomplished if $|\omega| = \phi$.

By instead choosing

$$\omega = \frac{\mathbf{a}_i \times \mathbf{r}_i}{\mathbf{a}_i \cdot \mathbf{r}_i} \tag{5}$$

we get

$$|\omega| = \frac{|\mathbf{a}_i||\mathbf{r}_i|\sin\phi}{|\mathbf{a}_i||\mathbf{r}_i|\cos\phi} = \tan\phi \approx \phi \tag{6}$$

for small ϕs. This yields our final iterative formula

$$\boxed{\mathbf{R} \leftarrow exp\left(\frac{\sum_i \mathbf{r}_i \times \mathbf{a}_i}{|\sum_i \mathbf{r}_i \cdot \mathbf{a}_i| + \varepsilon}\right)\mathbf{R}}, \tag{7}$$

where we choose the safety parameter $\varepsilon = 10^{-9}$ in our examples. It is important to use the absolute value in the denominator to not flip the direction of the torque. Although the formula is very simple it has a number of important features. In contrast to most existing approaches, it robustly handles a rank deficient matrix $\mathbf{A}$. In this case it carries over the missing information from the previous rotation matrix $\mathbf{R}$ which is the correct solution during a simulation. This is because if an $\mathbf{a}_i$ is zero, the corresponding torque component is zero too and the orientation of $\mathbf{R}$ along that direction untouched. In the extreme case of $\mathbf{A} = \mathbf{0}$, the total torque is zero and $\mathbf{R}$ remains unchanged. If $\mathbf{a}_1$ is the only non-zero axis, our method correctly rotates $\mathbf{R}$ in the plane spawned by $\mathbf{r}_1$ and $\mathbf{a}_1$ to align $\mathbf{r}_i$ with $\mathbf{a}_1$. The determinant of $\mathbf{R}$ is guaranteed to remain +1 even if $det(\mathbf{A}) < 0$ because we treat $\mathbf{R}$ as a rigid body. In addition, the use of the previous rotation makes our method particularly efficient for simulations because warm starting is built into it.

Appendix D lists the source code of our method in C++. The *Eigen* library (eigen.tuxfamily.org) is used to perform vector and matrix

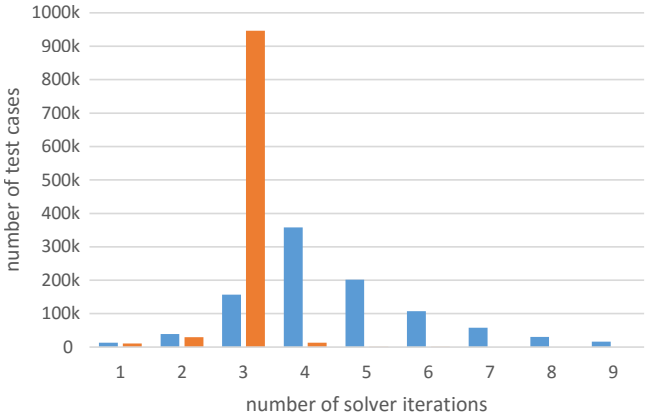

Figure 2: *Distribution of the number of solver iterations needed to reach $F < 0.001$ starting with random orientation matrices* $\mathbf{R}$ *and with* $\mathbf{A} = \mathbf{I}$ *(blue). Restricting the Euler angles of* $\mathbf{R}$ *to* $[-\pi/3, \pi/3]$ *yields the distribution shown by the orange bars.*

operations. For practical reasons we represent $\mathbf{R}$ by a quaternion $\mathbf{q}$. A comparison with the source code to implement Irving et al. [Irving et al. 2004] shown in Appendix C illustrates how much simpler the implementation of our method is. Our procedure takes as input the solution of the previous step. If such a solution is not available, we recommend to start with $\mathbf{q} = \text{Quat}(\mathbf{A})/|\text{Quat}(\mathbf{A})|$.

We have left out one important question, namely whether there are equilibria for which the torque τ is zero but for which F is not minimized, but instead maximized. Such equilibria can indeed be constructed. It is easy to verify that for $\mathbf{A} = \mathbf{I}$ and

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{8}$$

the torque is zero. Therefore, if the iteration is started with this $\mathbf{R}$ it will not converge to the true solution $\mathbf{R} = \mathbf{I}$ which minimizes the Frobenius norm. Fortunately, a small random perturbation will almost always get us out of the maxima, unless it happens to be along a ridge/plane of the maxima.

Figure 2 shows the results of an experiment in which we set $\mathbf{A} = \mathbf{I}$ and run our method a million times each time with a different, randomly chosen starting matrix $\mathbf{R}$. Our method always converged to the true solution $\mathbf{R} = \mathbf{I}$ which is expected if the equilibria form a null space. In this case, the probability to hit an equilibrium by chance is theoretically zero and numerically very small. The fact that our method never converged to a non-minimizing equilibrium in our examples is also a strong indication that the maxima is very rare. If one wishes to guarantee to avoid the maxima, when $\tau = 0$, one can add a small random rotation to $\mathbf{R}$ and check if the Frobenius norm increases or not. If it decreases, one should restart the iterations from that perturbed $\mathbf{R}$. If it increases, then rejects the rotation and you are done. If it stays the same, keeps adding another random rotation until the norm changes. One can also randomly perturbed $\mathbf{R}$ by a small ε at the beginning of the solve to get out of the maxima, if it starts being in one. In our simulations, maximum equilibria were never encountered, so we don't do these checks nor perturbation in our examples.

4 Results

We also used the experiment described in the last section to test the performance of our method. The blue bars in Figure 2 show the distribution of the number of solver iterations needed to converge to $F < 0.001$ when starting with randomly chosen rotation matrices. To emulate a warm start we restricted the Euler angles of $\mathbf{R}$ to the range $\mathbf{R}$ to $[-\pi/3, \pi/3]$. In this case, the method almost always converges after 3 iterations as the orange bars show.

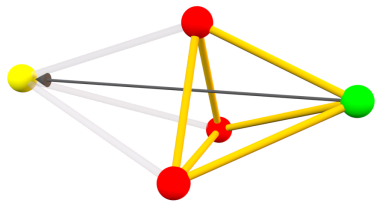

Figure 3: *A tetrahedron is inverted by moving the green vertex to the yellow position.*

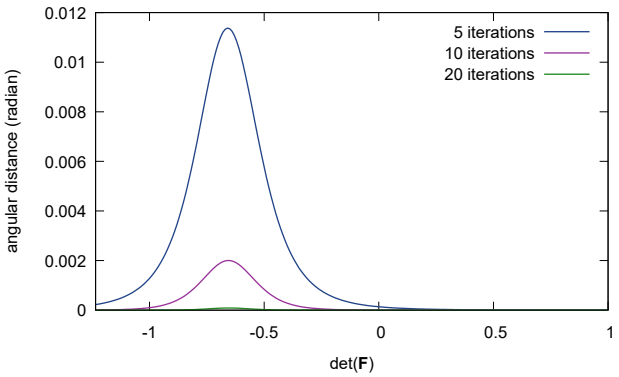

Figure 4: *Angular distance of our solution and the one of Irving et al. [2004] for different numbers of iteration. The determinant of the deformation gradient* $\mathbf{F}$ *is shown on the x-axis. A tetrahedron was deformed from its rest configuration (*$\det(\mathbf{F}) = 1$*) over a degenerate state (*$\det(\mathbf{F}) = 0$*) to a totally inverted configuration (*$\det(\mathbf{F}) = -1$*).*

We now compare our novel method with the approach of Irving et al. [2004].

First, we demonstrate that our simple algorithm converges to the same result as the method of Irving et al. For the comparison we deform a tetrahedron to an inverted state. Three points of the tetrahedron are fixed and one moves on the x-axis from the rest configuration over a degenerate state to a completely inverted configuration (see Figure 3).

Figure 4 shows the angular distance between the resulting rotation matrices of both methods when using a warm start for our algorithm. When we increase the number of iterations, both methods converge to the same solution. However, with a low number of iterations we can see that the difference between both methods is larger around the point where $\det(\mathbf{F}) \approx -0.66$. The reason for this larger difference is that at this point the method of Irving et al. has a discontinuity [Civit-Flores and Susin 2012]. Since our method only performs a rotation in each iteration, it requires more iterations to come to the same result at the discontinuity. Note that the maximum difference of approximately 0.01 radian at this point is still low.

Our approach performs better if it is started with a quaternion which is close to the solution. A comparison of the angular distance of different start configurations is shown in Figure 5. The best convergence we can see for a warm start of the algorithm with a previous

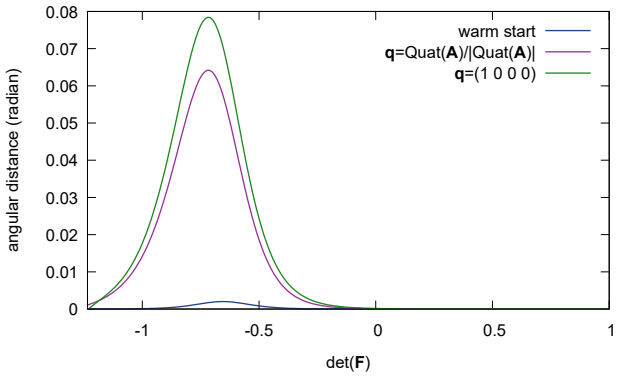

Figure 5: *Angular distance when starting our method with different quaternions* **q**. *In this comparison we used 10 iterations.*

solution **q**. If such a solution is not available, a better convergence can be achieved with the quaternion $\mathbf{q} = \mathrm{Quat}(\mathbf{A})/|\mathrm{Quat}(\mathbf{A})|$ which was recommended in the last section.

We also performed tests where we inverted the tetrahedron twice by moving two different vertices in inverted configurations. The results in these tests also converged towards the solution of Irving et al.

We tested our method in the simulation scenarios shown in Figure 6. The objects are simulated using the clustered shape matching approach. The clusters and their orientations are visualized as local coordinate frames. Even though the cluster distributions are sparse, our method allows the stable simulation of these objects. In Figure 7 after completely randomizing the vertex locations, the dragon model recovers its original shape. When using 3 iterations, which was enough in our simulation experiments to achieve good results without any visible artifacts, our method was 60 % - 80 % faster than the approach of Irving et al. Note that the usage of an optimized SVD implementation like the one of McAdams et al. [2011] could improve the performance of Irving's method so the speedup in this case would be smaller. However, a GPU implementation of such an optimized SVD is far more complex than the one of our method.

5 Conclusion

We have presented a new method to extract the rotational part of an arbitrary 3×3-dimensional matrix. The resulting algorithm is simple and short without branches and therefore ideal for the use on GPUs. It is also faster than previous approaches. We have not tested the method in the context of co-rotational FEM yet but expect it to perform equally well since it is independent of the actual application.

References

CIVIT-FLORES, O., AND SUSIN, A. 2012. Robust Treatment of Degenerate Elements in Interactive Corotational FEM Simulations. *Computer Graphics Forum 33*, 6.

FARRELL, J. L., AND STUELPNAGEL, J. C. 1966. A least squares estimate of spacecraft attitude. *SIAM Review 8*, 3, 384.

HAUTH, M., AND STRASSER, W. 2004. Corotational simulation of deformable solids. *Journal of WSCG 12*, 2–9.

HIGHAM, N. 1986. Computing the polar decomposition with applications. *SIAM J. Sci. Stat. Comput. 7*, 4, 1160–1174.

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

MARKLEY, F. L., AND MORTARI, D. 1999. How to estimate attitude from vector observations. In *AAS/AIAA Astrodynamics Specialist Conference*, vol. 103, 1979–1996.

MCADAMS, A., SELLE, A., TAMSTORF, R., TERAN, J., AND SIFAKIS, E. 2011. Computing the singular value decomposition of 3x3 matrices with minimal branching and elementary floating point operations. Tech. Rep. TR1690, University of Wisconsin.

MÜLLER, M., AND GROSS, M. 2004. Interactive virtual materials. In *Graphics Interface*, 239–246.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. In *ACM SIGGRAPH 2005 Papers*, ACM, 471–478.

SCHMEDDING, R., AND TESCHNER, M. 2008. Inversion handling for stable deformation modeling. *The Visual Computer 24*, 7, 625–633.

WAHBA, G. 1965. A least squares estimate of spacecraft attitude. *SIAM Review 7*, 3, 409.

WESSNER, R. 1966. A least squares estimate of spacecraft attitude. *SIAM Review 8*, 3, 386.

A Equilibrium and the Frobenius Norm

For the following proofs we define $\mathbf{a}_1$, $\mathbf{a}_2$ and $\mathbf{a}_3$ to be the column vectors of $\mathbf{A}$ and $\mathbf{r}_1$, $\mathbf{r}_2$ and $\mathbf{r}_3$ to be the column vectors of $\mathbf{R}$.

We show that if the Frobenius norm is minimized (or maximized) $\tau = \mathbf{0}$, i.e. the rotation matrix $\mathbf{R}$ is in an equilibrium state. First we have

$$F = \sum_i (\mathbf{r}_i - \mathbf{a}_i)^2. \tag{9}$$

If $\mathbf{R}$ minimizes or maximizes the Frobenius norm, the derivative of the Frobenius norm is zero in all directions. So we choose an arbitrary rotation axis ω. The rotation about ω by an angle ϕ can be linearized for a small ϕ and we get the angle dependent Frobenius norm

$$F(\phi) = \sum_i (\mathbf{r}_i + \phi\omega \times \mathbf{r}_i - \mathbf{a}_i)^2 \tag{10}$$

and the derivative

$$\frac{\partial F(\phi)}{\partial \phi} = 2 \sum_i (\mathbf{r}_i + \phi\omega \times \mathbf{r}_i - \mathbf{a}_i) \cdot \omega \times \mathbf{r}_i. \tag{11}$$

If $\frac{\partial F(\phi)}{\partial \phi} = 0$ for $\phi = 0$ then

$$\sum_i (\mathbf{r}_i - \mathbf{a}_i) \cdot \omega \times \mathbf{r}_i = \left(\sum_i \mathbf{r}_i \times \mathbf{a}_i \right) \cdot \omega = 0 \tag{12}$$

for an arbitrary vector ω. Therefore, $\tau = \sum_i \mathbf{r}_i \times \mathbf{a}_i = \mathbf{0}$ if $\mathbf{R}$ minimizes (or maximizes) the Frobenius norm.

Next, we show that if $\tau = \mathbf{0}$, then the Frobenius norm is minimized (or maximized). For an arbitrary rotation axis ω with angle θ, we have

$$\frac{\partial F(\phi)}{\partial \phi} = 2 \sum_i (\mathbf{r}_i + \phi\omega \times \mathbf{r}_i - \mathbf{a}_i) \cdot \omega \times \mathbf{r}_i. \tag{13}$$

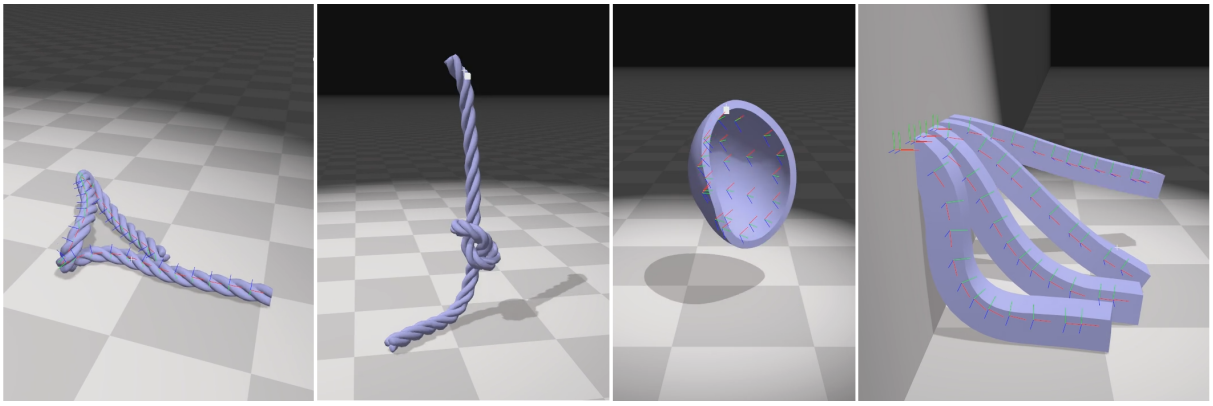

Figure 6: *Various objects simulated with the clustered shape matching approach. The clusters and their orientations are shown as local frames. Our method stably extracts the rotational part of the deformation in sparse clusters distributions.*

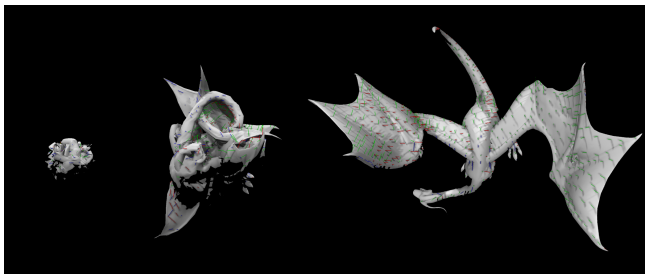

Figure 7: *After randomizing a complex model, our method is able to fully recover the original shape.*

At $\phi = 0$, we have $\frac{\partial F(\phi)}{\partial \phi} = \sum_i (\mathbf{r}_i - \mathbf{a}_i) \cdot \omega \times \mathbf{r}_i = (\sum_i \mathbf{r}_i \times \mathbf{a}_i) \cdot \omega = \tau \cdot \omega = 0$.

Therefore, $\tau = \mathbf{0}$, iff Frobenius norm is minimized (or maximized). □

B Equilibrium and the Polar Decomposition

Here we show that if $\mathbf{A} = \mathbf{RP}$ with $\mathbf{A}$ orthogonal and $\mathbf{P}$ symmetric positive semi-definite then $\tau = \mathbf{r}_1 \times \mathbf{a}_1 + \mathbf{r}_2 \times \mathbf{a}_2 + \mathbf{r}_3 \times \mathbf{a}_3 = \mathbf{0}$.

First, since $\mathbf{R}^T\mathbf{A} = \mathbf{P}$, we have $\mathbf{r}_i \cdot \mathbf{a}_j = p_{i,j}$. Moreover, since $\mathbf{R}$ is orthogonal

$$\begin{aligned} \pm\mathbf{r}_3 &= \mathbf{r}_1 \times \mathbf{r}_2 \\ \mp\mathbf{r}_2 &= \mathbf{r}_1 \times \mathbf{r}_3, \end{aligned}$$

choosing either all upper signs or all lower signs. Now

$$\begin{aligned} \mathbf{r}_1 \cdot \tau &= \mathbf{r}_1 \cdot (\mathbf{r}_1 \times \mathbf{a}_1 + \mathbf{r}_2 \times \mathbf{a}_2 + \mathbf{r}_3 \times \mathbf{a}_3) \\ &= 0 + \mathbf{r}_1 \cdot (\mathbf{r}_2 \times \mathbf{a}_2) + \mathbf{r}_1 \cdot (\mathbf{r}_3 \times \mathbf{a}_3) \\ &= \mathbf{a}_2 \cdot (\mathbf{r}_1 \times \mathbf{r}_2) + \mathbf{a}_3 \cdot (\mathbf{r}_1 \times \mathbf{r}_3) \\ &= \mathbf{a}_2 \cdot (\pm\mathbf{r}_3) + \mathbf{a}_3 \cdot (\mp\mathbf{r}_2) \\ &= \pm p_{3,2} \mp p_{2,3} \\ &= \pm p_{3,2} \mp p_{3,2} \quad (\mathbf{P} \text{ symmetric}) \\ &= 0 \end{aligned}$$

and similarly $\mathbf{r}_2 \cdot \tau = 0$ and $\mathbf{r}_3 \cdot \tau = 0$. It follows that $\tau = \mathbf{0}$. □

C Source Code - Irving et al.

```
void jacobiRotate(Matrix3d &A, Matrix3d &R, int p, int q)
{
   if (A(p, q) == 0.0)
      return;

   double d = (A(p, p) - A(q, q)) / (2.0*A(p, q));
   double t = 1.0 / (fabs(d) + sqrt(d*d + 1.0));
   if (d < 0.0)
      t = -t;
   double c = 1.0 / sqrt(t*t + 1);
   double s = t*c;
   A(p, p) += t*A(p, q);
   A(q, q) -= t*A(p, q);
   A(p, q) = A(q, p) = 0.0;

   int k;
   for (k = 0; k < 3; k++)
   {
      if (k != p && k != q)
      {
         double Akp = c*A(k, p) + s*A(k, q);
         double Akq = -s*A(k, p) + c*A(k, q);
         A(k, p) = A(p, k) = Akp;
         A(k, q) = A(q, k) = Akq;
      }
   }
   for (k = 0; k < 3; k++)
   {
      double Rkp = c*R(k, p) + s*R(k, q);
      double Rkq = -s*R(k, p) + c*R(k, q);
      R(k, p) = Rkp;
      R(k, q) = Rkq;
   }
}

void eigenDecomposition(const Matrix3d &A, Matrix3d &
    eigenVecs, Vector3d &eigenVals)
{
   const int numJacobiIterations = 10;
   const double epsilon = 1e-15;

   Matrix3d D = A;

   eigenVecs.setIdentity();
   int iter = 0;
   while (iter < numJacobiIterations)
   {
      int p, q;
      double a, max;
      max = fabs(D(0, 1));
      p = 0;
      q = 1;
      a = fabs(D(0, 2));
      if (a > max)
      {
         p = 0;
         q = 2;
         max = a;
```

```
      }
      a = fabs(D(1, 2));
      if (a > max)
      {
         p = 1;
         q = 2;
         max = a;
      }
      if (max < epsilon)
         break;
      jacobiRotate(D, eigenVecs, p, q);
      iter++;
   }
   eigenVals[0] = D(0, 0);
   eigenVals[1] = D(1, 1);
   eigenVals[2] = D(2, 2);
}

void rotationMatrixIrving(const Matrix3d &A, Matrix3d &R)
{
   Matrix3d AT_A, V;
   AT_A = A.transpose() * A;

   Vector3d S;
   eigenDecomposition(AT_A, V, S);

   const double detV = V.determinant();
   if (detV < 0.0)
   {
      double minLambda = DBL_MAX;
      unsigned char pos = 0;
      for (unsigned char l = 0; l < 3; l++)
      {
         if (S[l] < minLambda)
         {
            pos = l;
            minLambda = S[l];
         }
      }
      V(0, pos) = -V(0, pos);
      V(1, pos) = -V(1, pos);
      V(2, pos) = -V(2, pos);
   }

   if (S[0] < 0.0f)
      S[0] = 0.0f;
   if (S[1] < 0.0f)
      S[1] = 0.0f;
   if (S[2] < 0.0f)
      S[2] = 0.0f;

   Vector3d sigma;
   sigma[0] = sqrt(S[0]);
   sigma[1] = sqrt(S[1]);
   sigma[2] = sqrt(S[2]);

   unsigned char chk = 0;
   unsigned char pos = 0;
   Matrix3d U;
   for (unsigned char l = 0; l < 3; l++)
   {
      if (fabs(sigma[l]) < 1.0e-4)
      {
         pos = l;
         chk++;
      }
   }

   if (chk > 0)
   {
      if (chk > 1)
      {
         U.setIdentity();
      }
      else
      {
         U = A * V;
         for (unsigned char l = 0; l < 3; l++)
         {
            if (l != pos)
            {
               for (unsigned char m = 0; m < 3; m++)
               {
                  U(m, l) *= 1.0f / sigma[l];
               }
            }
         }

         Vector3d v[2];
         unsigned char index = 0;
         for (unsigned char l = 0; l < 3; l++)
         {
            if (l != pos)
            {
               v[index++] = Vector3d(U(0, l), U(1, l), U
                    (2, l));
            }
         }
         Vector3d vec = v[0].cross(v[1]);
         vec.normalize();
         U(0, pos) = vec[0];
         U(1, pos) = vec[1];
         U(2, pos) = vec[2];
      }
   }
   else
   {
      Vector3d sigmaInv(1.0 / sigma[0], 1.0 / sigma[1],
           1.0 / sigma[2]);
      U = A * V;
      for (unsigned char l = 0; l < 3; l++)
      {
         for (unsigned char m = 0; m < 3; m++)
         {
            U(m, l) *= sigmaInv[l];
         }
      }
   }

   const double detU = U.determinant();

   if (detU < 0.0)
   {
      double minLambda = DBL_MAX;
      unsigned char pos = 0;
      for (unsigned char l = 0; l < 3; l++)
      {
         if (sigma[l] < minLambda)
         {
            pos = l;
            minLambda = sigma[l];
         }
      }

      sigma[pos] = -sigma[pos];
      U(0, pos) = -U(0, pos);
      U(1, pos) = -U(1, pos);
      U(2, pos) = -U(2, pos);
   }

   R = U * V.transpose();
}
```

D Source Code - Our Method

```
void extractRotation(const Matrix3d &A, Quaterniond &q,
     const unsigned int maxIter)
{
   for (unsigned int iter = 0; iter < maxIter; iter++)
   {
      Matrix3d R = q.matrix();
      Vector3d omega = (R.col(0).cross(A.col(0)) + R.col
           (1).cross(A.col(1)) + R.col(2).cross(A.col(2))
           ) * (1.0 / fabs(R.col(0).dot(A.col(0)) + R.col
           (1).dot(A.col(1)) + R.col(2).dot(A.col(2))) +
           1.0e-9);
      double w = omega.norm();
      if (w < 1.0e-9)
         break;
      q = Quaterniond(AngleAxisd(w, (1.0 / w) * omega)) *
           q;
      q.normalize();
   }
}
```