EUROGRAPHICS 2021 / N. Mitra and I. Viola
(Guest Editors)

Volume 40 (2021), Number 2

Fast Updates for Least-Squares Rotational Alignment

Jiayi Eris Zhang[1] Alec Jacobson[1] Marc Alexa[2]

[1]University of Toronto, Canada [2]TU Berlin, Germany

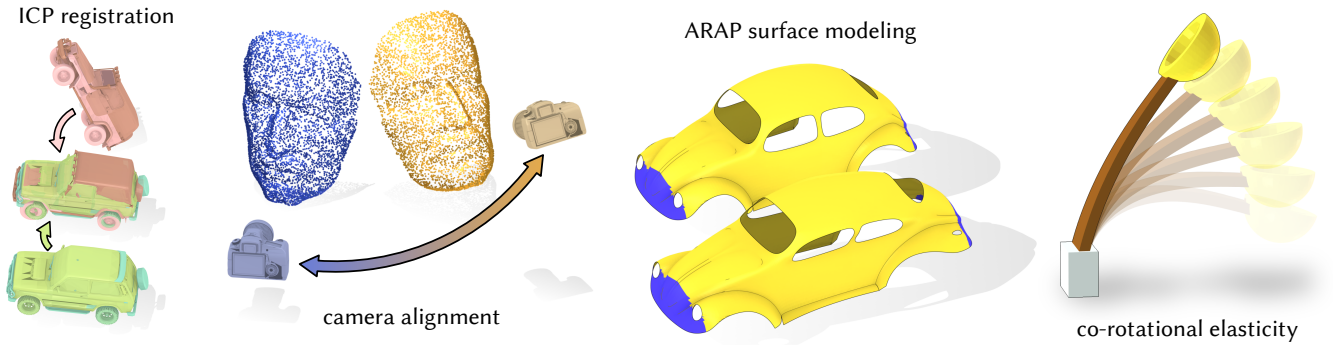


Figure 1: *Least-squares rotation fitting is a core low-level subroutine in a number of important high-level tasks in computer graphics, geometry processing, robotics and computer vision.*

Abstract

Across computer graphics, vision, robotics and simulation, many applications rely on determining the 3D rotation that aligns two objects or sets of points. The standard solution is to use singular value decomposition (SVD), where the optimal rotation is recovered as the product of the singular vectors. Faster computation of only the rotation is possible using suitable parameterizations of the rotations and iterative optimization. We propose such a method based on the Cayley transformations. The resulting optimization problem allows better local quadratic approximation compared to the Taylor approximation of the exponential map. This results in both faster convergence as well as more stable approximation compared to other iterative approaches. It also maps well to AVX vectorization. We compare our implementation with a wide range of alternatives on real and synthetic data. The results demonstrate up to two orders of magnitude of speedup compared to a straightforward SVD implementation and a 1.5-6 times speedup over popular optimized code.

1. Introduction

Finding the best rigid alignment between two sets of corresponding points is an important computational problem across essentially all areas of science and engineering. It has been introduced under various names and a variety of solutions have appeared independently in the different fields. We formally state the problem and survey common computational approaches in Section 2.

Rigid alignment has become a time-critical task in geometric modeling and animation of elastic materials. In both areas, the behavior of a material is described by non-linear energies that measure only the elastic deformation but are invariant to (local) rigid motions. It has been observed that the energies can often be minimized in a so-called local-global approach (or block-coordinate descent), a global solution of a sparse linear system and the local computation of rigid alignment [MHTG05, SA07, CPSS10]. In particular, the sparse systems are commonly constant throughout the minimization, allowing to factor the system once and solve each step by backsubstitution. In this setup, computing the local rigid transformations has become the bottleneck.

We suggest an approach to compute the local rotation, geared towards scenarios that are dominated by small rotation updates. Our main technical contribution is a careful analysis enabled by locally parameterizing rotations using the Cayley transformation (Section 3) rather than the more commonly used exponential map. The Cayley transformation leads to rational function in three variables representing the squared distances of corresponding points. Optimizing this function iteratively improves over naive Newton iterations by significantly widening the basin of attraction, while on the other hand avoiding performance penalties introduced by damping (see Section 4).

We have implemented our algorithm as well as several others. Another core contribution of our work is a comprehensive comparison among many if not all current methods for computing least squares alignment. In order to provide meaningful statistics, we collect a large number of data points from actual applications such as interactive deformations using as-rigid-as-possible deformations [SA07] and co-rotational elasticity [CPSS10]. Compared to a state of the art optimized SVD implementation [MST*11] we see a speedup of at least 1.5×, and more for smaller rotations. In the regime of approximating very small rotational updates with a single (Newton) step [KKB18], our approach (Cayley Conservative) always achieves better accuracy and suffers 73% fewer failure cases towards convergence.

2. Background

The problem of rigid alignment can be stated as follows: consider two corresponding sets of n points $\{\mathbf{x}_i \in \mathbb{R}^3\}$ and $\{\mathbf{y}_i \in \mathbb{R}^3\}$. Compute the rigid transformation that minimizes the (possibly weighted) sum of squared differences between corresponding points:

$$\underset{\mathbf{R}^\mathsf{T}\mathbf{R}=\mathbf{I},|\mathbf{R}|=1}{\arg\min} \sum_i w_i \|\mathbf{R}\mathbf{x}_i - \mathbf{y}_i + \mathbf{t}\|^2. \tag{1}$$

The optimal translation ($\mathbf{t} \in \mathbb{R}^3$) is independent of the rotation and aligns the weighted centroids of the points. For this reason it is common to assume the point sets have been translated so that their weighted centroids are in the origin and focus only on (the more difficult) problem of computing the rotation $\mathbf{R}$.

The information about the relative orientation of the points can be compactly captured in the cross-covariance matrix. For this we write the (translated) points in matrix form as $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{3\times n}$, the weights in a diagonal matrix $\mathbf{W}$, and then the (weighted) cross-covariance matrix as

$$\mathbf{M} = \mathbf{X}^\mathsf{T}\mathbf{W}\mathbf{Y}. \tag{2}$$

The prevalent approach to computing the rotation uses the SVD. By elementary transformations [SHR17], the minimization problem in Eq. 1 can be expressed in terms of the matrix trace of only the cross-covariance matrix $\mathbf{M}$ and the desired rotation $\mathbf{M}$:

$$\underset{\mathbf{R}^\mathsf{T}\mathbf{R}=\mathbf{I},|\mathbf{R}|=1}{\arg\max} \operatorname{tr}(\mathbf{M}\mathbf{R}). \tag{3}$$

This maximization problem is solved by computing the SVD $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^\mathsf{T}$ and then setting

$$\mathbf{R} = \mathbf{U}\begin{pmatrix} 1 & & \\ & 1 & \\ & & |\mathbf{U}\mathbf{V}^\mathsf{T}| \end{pmatrix}\mathbf{V}^\mathsf{T}. \tag{4}$$

The lower right entry in the diagonal matrix accounts for the case that $\mathbf{M}$ has negative determinant, in which case the product of $\mathbf{U}$ and $\mathbf{V}^\mathsf{T}$ would correspondingly contain an unwanted reflection.

This solution is known for a long time. Schönemann [Sch66] is often credited with being the first to solve the case of finding an orthogonal matrix. The solution above with forcing a special orthogonal matrix is sometimes referred to as Kabsch's algorithm [Kab76] and has been (re-)discovered multiple times in different areas of application [AHB87, Mar87, Ume91].

Based on our survey of the literature, the SVD-based approach is also the one most commonly used in graphics. This explains the enormous effort in optimizing the computation of the SVD for 3×3 matrices on current multi-core hardware. Sifakis et al. [MST*11] have analyzed the operations necessary for computing the SVD of a 3×3 matrix and greatly optimized the computation. They provide implementations exploiting current hardware using SIMD and streaming architectures using SSE and AVX. To our knowledge and consistent with our experimental findings, their code currently provides the fastest way to compute the SVD.

Other approaches for computing the rotation are available as well. Faugeras & Hebert [FH86] and Horn [Hor87] derive a direct method using a quaternion representation by computing the eigenvector of an appropriately assembled 4×4 matrix. Dual quaternions have been successfully used in graphics for representing the rigid motion (not just the rotation part); a construction for solving the rigid alignment similar to the one for quaternions exists [WSV91]. We have found claims that the quaternion based approach is faster than SVD, however, our experiments fail to support this. Interestingly, one past survey [ELF97] concluded that the dual quaternion methods were fastest by a close margin for the general rigid body fitting problem.

The conversion into an eigen problem can also be used without transforming to a quaternion-based representation [HHN88, Shu78, Mor97, YZ13]. It is claimed that the fastest way to solve the eigenproblem is by analytically solving for the quartic roots.

Our approach, in contrast, starts from the Cayley transformation for representing the rotation. This parameterization enjoys the property of only requiring 3 variables. It is known to be quite useful close to the origin (i.e., identity) and less so for large rotation angles. This nicely fits the dominant case in the iterative solvers used in graphics. The Cayley parametrization of rotation matrices appears in other contexts such as kinematics for robotics [GKWT98], as a mathematically convenient tool for deriving other quantities [Nor08, MM11], or to design a differentiable parametrization of random rotations [JHD18]. It has also been used for more general alignment problems that cannot be solved using SVD or eigendecomposition [WV06].

Newton's method has been applied to polar decomposition [Hig86, BX08], and recently Kugelstadt et al. [KKB18] also applied the Cayley transformation in this context. Their method is based on the local quadratic approximation implicitly used in the Newton step, representing rotations as quaternions. Our approach, in its simplest form, is similar except for the use of quaternions as a representation.

3. Cayley parameterization

The Cayley parameterization of rotations is based on skew-symmetric matrices. In $\mathbb{R}^3$ we can write the skew-symmetric matrix explicitly as

$$\mathbf{Z} = \begin{pmatrix} 0 & -z_2 & z_1 \\ z_2 & 0 & -z_0 \\ -z_1 & z_0 & 0 \end{pmatrix} \tag{5}$$

and represent the three unknowns as $\mathbf{z} = (z_0, z_1, z_2)$. Then any rotation matrix except for rotations by π can be expressed as

$$\mathbf{R} = (\mathbf{I}+\mathbf{Z})(\mathbf{I}-\mathbf{Z})^{-1}. \tag{6}$$

The axis $\mathbf{r}$ and angle θ of the rotation represented by $\mathbf{R}$ are parameterized as

$$\mathbf{z} = \tan\frac{\theta}{2}\,\mathbf{r} \quad \Longleftrightarrow \quad \mathbf{r} = \frac{\mathbf{z}}{\|\mathbf{z}\|},\ \theta = 2\arctan\|\mathbf{z}\|. \tag{7}$$

Compared to the more commonly used exponential map, the axis of rotation is also encoded as the direction of the 3-component vector, but the angle of rotation is a non-linear function of the length going to infinity as the angle approaches π.

Based on this we can write the trace function (Eq. 3) to be maximized as

$$g : \mathbb{R}^3 \mapsto \mathbb{R}, \quad g(\mathbf{z}) = \operatorname{tr}\left(\mathbf{M}(\mathbf{I}+\mathbf{Z})(\mathbf{I}-\mathbf{Z})^{-1}\right). \tag{8}$$

In the following, our goal is to rewrite this as a function of $\mathbf{z}$ rather than $\mathbf{Z}$.

The inverse of $\mathbf{I}-\mathbf{Z}$ has a simple expression:

$$(\mathbf{I}-\mathbf{Z})^{-1} = \frac{1}{1+\mathbf{z}^\mathsf{T}\mathbf{z}}\left(\mathbf{I}+\mathbf{z}\mathbf{z}^\mathsf{T}+\mathbf{Z}\right). \tag{9}$$

This leads to the representation for 3×3 rotations as

$$\begin{aligned}(\mathbf{I}+\mathbf{Z})(\mathbf{I}-\mathbf{Z})^{-1} &= \frac{1}{1+\mathbf{z}^\mathsf{T}\mathbf{z}}\left(\mathbf{I}+\mathbf{z}\mathbf{z}^\mathsf{T}+2\mathbf{Z}+\underbrace{\mathbf{Z}\mathbf{z}}_{=\mathbf{0}}\mathbf{z}^\mathsf{T}+\underbrace{\mathbf{Z}^2}_{=\mathbf{z}\mathbf{z}^\mathsf{T}-\mathbf{z}^\mathsf{T}\mathbf{z}\mathbf{I}}\right)\\ &= \frac{1}{1+\mathbf{z}^\mathsf{T}\mathbf{z}}\left((1-\mathbf{z}^\mathsf{T}\mathbf{z})\mathbf{I}+2\mathbf{z}\mathbf{z}^\mathsf{T}+2\mathbf{Z}\right).\end{aligned} \tag{10}$$

Plugging in this leads to

$$g(\mathbf{z}) = \frac{1}{1+\mathbf{z}^\mathsf{T}\mathbf{z}}\left((1-\mathbf{z}^\mathsf{T}\mathbf{z})\operatorname{tr}(\mathbf{M})+2\operatorname{tr}(\mathbf{M}\mathbf{z}\mathbf{z}^\mathsf{T})+2\operatorname{tr}(\mathbf{M}\mathbf{Z})\right). \tag{11}$$

For the last trace in the sum we introduce the following notation

$$\operatorname{tr}(\mathbf{M}\mathbf{Z}) = \begin{pmatrix} m_{12}-m_{21} \\ m_{20}-m_{02} \\ m_{01}-m_{10}\end{pmatrix}^\mathsf{T}\mathbf{z} = \mathbf{m}^\mathsf{T}\mathbf{z} \tag{12}$$

and we note that

$$\operatorname{tr}(\mathbf{M}\mathbf{z}\mathbf{z}^\mathsf{T}) = \mathbf{z}^\mathsf{T}\mathbf{M}\mathbf{z} = \mathbf{z}^\mathsf{T}\mathbf{M}^\mathsf{T}\mathbf{z}. \tag{13}$$

Putting everything together this leads to the following rational quadratic function to be maximized:

$$g(\mathbf{z}) = \frac{\operatorname{tr}(\mathbf{M})+2\mathbf{m}^\mathsf{T}\mathbf{z}+\mathbf{z}^\mathsf{T}\left(\mathbf{M}+\mathbf{M}^\mathsf{T}-\operatorname{tr}(\mathbf{M})\mathbf{I}\right)\mathbf{z}}{1+\mathbf{z}^\mathsf{T}\mathbf{z}}. \tag{14}$$

Note that this function is valid (and not just an approximation to the problem) whenever the optimal rotation does not require a rotation by π.

Analysis It is instructive to analyze the behavior of g on the symmetric and antisymmetric parts of $\mathbf{M}$. For this consider

$$\mathbf{M}_S = \frac{1}{2}\left(\mathbf{M}+\mathbf{M}^\mathsf{T}\right) \qquad \mathbf{M}_A = \frac{1}{2}\left(\mathbf{M}-\mathbf{M}^\mathsf{T}\right) \tag{15}$$

and denote g_S and g_A the functions resulting from restricting g to the symmetric, resp. anti-symmetric part of $\mathbf{M}$. We have

$$g_S(z) = \frac{\operatorname{tr}(\mathbf{M})+\mathbf{z}^\mathsf{T}\left(\mathbf{M}+\mathbf{M}^\mathsf{T}-\operatorname{tr}(\mathbf{M})\mathbf{I}\right)\mathbf{z}}{1+\mathbf{z}^\mathsf{T}\mathbf{z}} \quad \text{and} \tag{16}$$

$$g_A(\mathbf{z}) = \frac{2\mathbf{m}^\mathsf{T}\mathbf{z}}{1+\mathbf{z}^\mathsf{T}\mathbf{z}} \tag{17}$$

and see that $g(\mathbf{z}) = g_S(\mathbf{z}) + g_A(\mathbf{z})$ for any $\mathbf{M}$ and $\mathbf{z}$.

We can analyze the maximum of each of the functions g_S, g_A independently. In both cases it is convenient to parameterize $\mathbf{z} = \kappa\tilde{\mathbf{z}}, \|\tilde{\mathbf{z}}\| = 1$. For g_S note that $\operatorname{tr}(\mathbf{M})$ is independent of $\tilde{\mathbf{z}}$ so we can choose it as the eigenvector corresponding to the largest eigenvalue of $2\mathbf{M}_S - \operatorname{tr}(\mathbf{M})\mathbf{I}$ in order to maximize this term. Let $\lambda_0 \le \lambda_1 \le \lambda_2$ be the eigenvalues of $\mathbf{M}_S$ and recall that $\mathbf{M}_S$ and $\mathbf{M}_S + \alpha\mathbf{I}$ have the same eigenvectors. Then the largest eigenvalue of $2\mathbf{M}_S - \operatorname{tr}(\mathbf{M})\mathbf{I}$ is $2\lambda_2 - \operatorname{tr}(\mathbf{M})$. So maximizing g_S boils down to setting $\tilde{\mathbf{z}}$ to the eigenvector corresponding to $\lambda_2(\mathbf{M}_S)$ and maximizing

$$g_S(\kappa\tilde{\mathbf{z}}) = \frac{\operatorname{tr}(\mathbf{M})+(\lambda_2-\operatorname{tr}(\mathbf{M}))\kappa^2}{1+\kappa^2}. \tag{18}$$

This function takes on the value $\operatorname{tr}(\mathbf{M})$ for $\kappa = 0$ (its only possible critical point) and $\lambda_2 - \operatorname{tr}(\mathbf{M})$ for $\kappa \to \pm\infty$, and is monotone in-between. This means it is maximized at $\kappa = 0$ (corresponding to the identity rotation) if $\lambda_0 + \lambda_1 > 0$. Conversely, if $\lambda_0 + \lambda_1 < 0$ the maximal value is $2\lambda_2 - \operatorname{tr}(\mathbf{M})$ for a rotation by $\pm\pi$ around $\tilde{\mathbf{z}}$. The function is constant if $\lambda_0 + \lambda_1 = 0$. This means any rotation around $\tilde{\mathbf{z}}$ leads to the maximal value $\operatorname{tr}(\mathbf{M})$. If the situation is (numerically) close to this case it will be difficult to optimize based on the gradient, because the derivative of g can be arbitrarily close to zero.

For g_A the optimal choice of direction is $\tilde{\mathbf{z}} = \mathbf{m}/\|\mathbf{m}\|$ and we get the equation

$$g_A(\kappa\tilde{\mathbf{z}}) = \frac{2\kappa\mathbf{m}^\mathsf{T}\mathbf{m}}{\|\mathbf{m}\|\left(1+\kappa^2\right)} = \|\mathbf{m}\|\frac{2\kappa}{1+\kappa^2}, \tag{19}$$

which takes on the maximal value $\|\mathbf{m}\|$ for $\kappa = 1$.

For any $\mathbf{M}$ that is neither symmetric nor anti-symmetric g takes on its maximal value for finite non-zero $\mathbf{z}$. In the next section we discuss how to find the maximum using gradient-based iterative approaches.

4. Optimization

In order to find the optimal rotational alignment we now must determine the three variables $\mathbf{z}$ that maximize Eq. 14. Unless the function is flat, and if a good starting point is provided, iterative maximization using the gradient is the method of choice. In the following we first discuss the standard Newton approach, and then how to improve exploiting the quadratic rational representation of g.

4.1. Newton optimization with reparameterization

It is straightforward to take derivatives of g because it is a rational polynomial. The gradient and Hessian are given in the Appendix A. This immediately enables Newton optimization for the maximization.

A typical approach in the literature using Newton iterations is to reparameterize the rotations after each step [KKB18]. The main motivation appears to be the distortion introduced with the linearization of the rotations away from the origin. In our derivation we also see that the derivatives are significantly simpler at the origin (i.e. $\mathbf{z} = \mathbf{0}$):

$$\nabla g(\mathbf{0}) = 2\mathbf{m} \tag{20}$$

$$\mathbf{H}_g(\mathbf{0}) = 2\left(\mathbf{M} + \mathbf{M}^\mathsf{T} - 2\,\mathrm{tr}(\mathbf{M})\mathbf{I}\right). \tag{21}$$

Note that the resulting Newton step is identical to the one derived based on the exponential map [KKB18]: The reason for this is that the Cayley transformation and exponential map only differ in how the angle for rotation is parameterized. For the Cayley transformation we have $\|\mathbf{z}\| = \tan\theta/2$ and the identity follows from $\tan x = x + O(x^3)$.

In our setup, the idea of reparameterizing after the update leads to the following procedure:

1. Start with any guess for the optimal rotation $\mathbf{R}_0$. This could be identity, however, we will explain later where this guess could come from depending on the application.
2. In each step, apply the rotation, i.e. compute $\mathbf{M}_i = \mathbf{M}\mathbf{R}_i$.
3. Compute the optimal value $\mathbf{z}_i$. In the case of Newton optimization this means $\mathbf{z}_i = -\mathbf{H}_g(\mathbf{0})^{-1}\nabla g(\mathbf{0})$. We will later provide an improved update step.
4. Turn the Cayley representation $\mathbf{z}_i$ of the rotational step into a rotation $\mathbf{R}_{i+1}$ and go back to step 2 (until the step $\mathbf{z}_i$ converges to zero).

Each step requires only very few operations. The most expensive computation is the solution of a 3×3 linear system. Once we have computed the update in Cayley representation we need to turn it into a rotation matrix. This can be done efficiently based on Eq. 10. Using this step, we arrive at the overall algorithm detailed in pseudocode below. This procedure relies on an update step in Cayley representation. We next explain how to improve over the standard Newton step.

ALGORITHM 1: Optimization of rotational alignment.

Function `Rotation alignment` (*cross-covariance matrix* $\mathbf{M}$, *initial rotation* $\mathbf{R}$)
  repeat
    $\mathbf{M}' \leftarrow \mathbf{M}\mathbf{R}$
    $\mathbf{z} \leftarrow$ Cayley-Step($\mathbf{M}'$)
    $s \leftarrow \mathbf{z}^\mathsf{T}\mathbf{z}$
    $\mathbf{R} \leftarrow \frac{1}{1+s}\mathbf{R}\left((1-s)\mathbf{I} + 2\mathbf{z}\mathbf{z}^\mathsf{T} + 2\mathbf{Z}\right)$
  until $s < \varepsilon$
  return $\mathbf{R}$

4.2. Improved local quadratic approximation

A common problem with the Newton method is that convergence depends on how well the function is locally approximated by the quadratic function derived from the gradient and Hessian. It is generally not easy to determine from $\mathbf{M}$ if the local quadratic approximation around $\mathbf{z} = 0$ is "good enough". One observation is that we expect that the quadratic function has negative definite Hessian around the maximum. Rather than checking the definiteness of the Hessian (which would be expensive) it is customary to subtract a small multiple of the identity to ensure the desired definiteness of the Hessian and, thus the desired concavity of the local approximation around the maximum. This approach is often referred to as damping.

Rather than relying on such heuristics, it turns out that it is possible to characterize the maximum of $g(\mathbf{z})$ in a helpful way. For this we make use of an ancient technique that enables the analytical computation of the critical points of a rational quadratic function without today's knowledge of calculus [Suz05]. Our approach is inspired by this pre-calculus (i.e., "ancient") approach for finding the extreme points of a rational quadratic function: given an extreme value of the function, we can turn the problem of finding the corresponding extreme point into finding a root of a quadratic polynomial. Since the extreme point is generically unique, the root is unique, so it is not only a root but also an extreme point of the quadratic function. This leads to a linear equation characterizing the value of the extreme point. If the function is univariate this approach indeed leads to an 'analytic' solution – the same one that could be derived using calculus. In the multivariate case it leads to an eigenproblem (again, similar to the derivation based on calculus), which is what we intend to avoid. However, what we do get from this analysis is a family of quadratic fits parameterized by the (estimate of) the extreme value. In terms of the format, the resulting linear equation for locating the extreme point turns out to be identical to a damped Newton step. Conservatively estimating the extreme value leads to a modified Newton step that is generally more robust than standard Newton and faster than damped Newton with a fixed parameter towards convergence.

Assume we know that the maximal value of g is c. Since the denominator of g is strictly positive we get

$$\mathrm{tr}(\mathbf{M}) + 2\mathbf{m}^\mathsf{T}\mathbf{z} + \mathbf{z}^\mathsf{T}\left(\mathbf{M} + \mathbf{M}^\mathsf{T} - \mathrm{tr}(\mathbf{M})\mathbf{I}\right)\mathbf{z} - c\left(1 + \mathbf{z}^\mathsf{T}\mathbf{z}\right) = 0. \tag{22}$$

This is a quadratic equation in $\mathbf{z}$. Unless the situation is degenerate, there is only a single rotation that minimizes the squared distances of the points (i.e. $\mathbf{M}$ has full rank). This implies that the quadratic equation is satisfied only for a single point $\mathbf{z}^*$. This single solution has to be an extreme point of the quadratic so it is also characterized by setting the gradient to zero

$$2(\mathbf{T}\mathbf{z}^* + \mathbf{m}) + 2c\mathbf{z}^* = 0, \tag{23}$$

leading to

$$\left(\mathbf{M} + \mathbf{M}^\mathsf{T} - (\mathrm{tr}(\mathbf{M}) + c)\mathbf{I}\right)\mathbf{z}^* = -\mathbf{m}. \tag{24}$$

Note how this compares to the Newton step, for which we found $c = \mathrm{tr}(\mathbf{M})$. Indeed, if the solution to the maximization problem is $\mathbf{z} = 0$ than the optimal rotation is identity and the maximal trace is $\mathrm{tr}(\mathbf{M})$.

Our modification to the Newton step, rather than blindly adding a constant value for c, which would amount to the damping described above, is to approximate the maximal value of g. The inset shows a one-dimensional cut through g in black. The Newton iteration approximates the maximum based on the Taylor approximation in 0, shown in red.

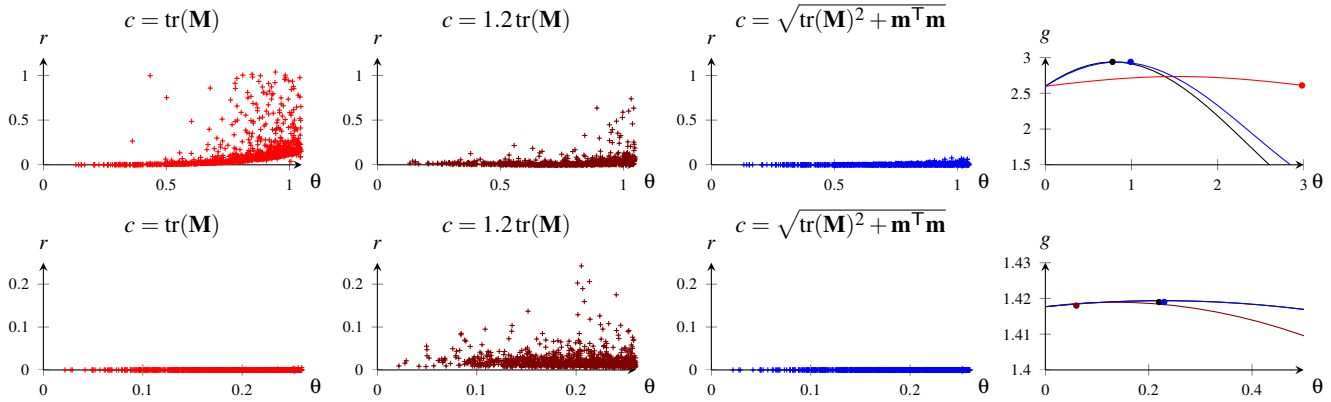
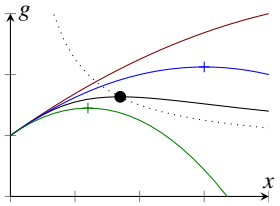


Figure 2: *Improvement of the relative distance of g to the maximal value in a single step. Upper row shows data including optimal rotations up to* $\theta = \pi/3$*, lower row up to* $\theta = \pi/12$*. Newton steps (*$c = \mathrm{tr}(\mathbf{M})$*, left) work well if the optimal rotation is small, but may fail to improve for larger angles. Damping (*$c = 1.2\,\mathrm{tr}(\mathbf{M})$*, middle left) improves the situation for larger angles, but cause reduced convergence speed for small angles. Our approximation based on* $\|\mathbf{m}\|$ *(middle right) works well in both regimes. The right column shows reparameterized cuts through* $g(\kappa\tilde{\mathbf{z}})$ *for one data point in the left scatter plots. The cut is chosen for each approximation based on the direction of z, i.e. the cuts are along different directions, and they are reparameterized based on angle; markers illustrate the updated step, which is not necessarily the maximum along the curve. The black curves, for reference, illustrate the direction of optimal rotation and the black dot indicates the maximal value of g. The upper graph illustrates how Newton steps generate local approximations that are too flat and then overshoot, the lower graph how damping reduces the step size.*



Different choices of c give rise to different local quadratic approximations of g. Underestimating c (the blue curve) causes overshooting. while overestimating (green) results in choosing $\mathbf{z}$ closer to the origin than the optimum. Unless the maximum of g is close to the origin, the approximation is more sensitive to underestimating than overestimating c – the dotted line shows c dependent on z.

Our idea for approximating c is based on the observation

$$\max_{\mathbf{z}} g(\mathbf{z}) = \max_{\mathbf{z}} \left(g_S(\mathbf{z}) + g_A(\mathbf{z})\right) \leq \max_{\mathbf{z}} g_S(\mathbf{z}) + \max_{\mathbf{z}} g_A(\mathbf{z}), \quad (25)$$

with equality only if one of the two maxima vanishes since the two maxima are attained for different values of $\mathbf{z}$. A simple analysis reveals that the factor equal maxima is bounded by $\sqrt{2}$ motivating the heuristic

$$\max_{\mathbf{z}} g(\mathbf{z}) \approx \left(\left(\max_{\mathbf{z}} g_S(\mathbf{z})\right)^2 + \left(\max_{\mathbf{z}} g_A(\mathbf{z})\right)^2\right)^{\frac{1}{2}}. \quad (26)$$

Based on our analysis of g_S and g_A we have the following approximation

$$c \approx \left(\max(\mathrm{tr}(\mathbf{M})^2, (2\lambda_2 - \mathrm{tr}(\mathbf{M}))^2 + \mathbf{m}^\mathsf{T}\mathbf{m}\right)^{\frac{1}{2}} \quad (27)$$

Computing $\mathbf{m}^\mathsf{T}\mathbf{m}$ is easy and can be readily used to improve the estimate for c. Compared to the standard Newton step it avoids overshooting and leads to better stability, yet it avoids the penalty in convergence speed associated with damping. Figure 2 demonstrates this for matrices $\mathbf{M}$ generated from points sampled uniformly in $[-1,1]^3$. We measure the effect of $\mathbf{z}$ computed from c by generating a new matrix $\mathbf{M}' = \mathbf{MR}(\mathbf{z})$. Then we consider the relative improvement of the absolute trace error $r = |\max g(\mathbf{z}) - \mathrm{tr}(\mathbf{M}')|/|\max g(\mathbf{z}) - \mathrm{tr}(\mathbf{M})|$. Newton steps work well if the angle of the best rotation is small, but the absolute error fails to improve if the necessary rotation is large, i.e. the maximum $\mathbf{z}^*$ is far from the origin. Using $c = \sqrt{\mathrm{tr}(\mathbf{M})^2 + \mathbf{m}^\mathsf{T}\mathbf{m}}$ significantly improves the situation. Damping, i.e. using $c = \kappa\,\mathrm{tr}(\mathbf{M})$ with $\kappa > 1$ helps to avoid problems for larger necessary rotations, but exhibits decreased convergence for small angles, whereas our solution behaves similar to standard Newton step also for small angles. The graphs on the right illustrate the local approximation resulting from the different approximations in the direction $\mathbf{z}$ and which point is chosen along the curve.

It remains to approximate the term $\max(\mathrm{tr}(\mathbf{M}), 2\lambda_2 - \mathrm{tr}(\mathbf{M}))$ for which we need an approximation of $\lambda_2(\mathbf{M}_S)$ that can be efficiently computed. As mentioned before, if we need λ_2, the necessary rotation is large, and it is better to overestimate to ensure convergence. For this reason we suggest to use Gershgorin disks to generate a conservative upper bound:

$$2\tilde{\lambda}_2 = \max_i \left(2m_{ii} + \sum_{j \neq i} |m_{ij} + m_{ji}|\right) \quad (28)$$

4.3. Efficient Computation

All computations involved for performing a single step require only few elementary operations. The most expensive part is solving the linear system

$$\left(\mathbf{M} + \mathbf{M}^\mathsf{T} - (\mathrm{tr}(\mathbf{M}) + c)\,\mathbf{I}\right)\mathbf{z} = \mathbf{Hz} = -\mathbf{m}, \quad (29)$$

for which we suggest to use Cramers rule. Let

$$\mathbf{H} = (\mathbf{h}_0, \mathbf{h}_1, \mathbf{h_2}), \tag{30}$$

then we find the elements of $\mathbf{z}$ as

$$z_0 = |\mathbf{H}|^{-1} |-\mathbf{m}, \mathbf{h}_1, \mathbf{h}_2| \tag{31}$$

$$z_1 = |\mathbf{H}|^{-1} |\mathbf{h}_0, -\mathbf{m}, \mathbf{h}_2| \tag{32}$$

$$z_2 = |\mathbf{H}|^{-1} |\mathbf{h}_0, \mathbf{h}_1, -\mathbf{m}|. \tag{33}$$

This requires only computing 4 determinants, each of which is composed of $6 \cdot 2$ multiplications, 4 additions, and 1 subtraction. We provide pseudocode for the solution step computed in this way as Algorithm 2 below.

ALGORITHM 2: Optimization step for rotational alignment in Cayley parameterization.

Function `Cayley-Step(M)`

$\mathbf{m} \leftarrow (m_{12} - m_{21}, m_{20} - m_{02}, m_{01} - m_{10})^\mathsf{T}$

$2\lambda_2 \leftarrow \max_i \left(2m_{ii} + \sum_{j \neq i} |m_{ij} + m_{ji}|\right)$

$t \leftarrow \mathrm{tr}(\mathbf{M})$

$\tilde{g}_S \leftarrow \max(t, 2\lambda_2 - t)$

$(\mathbf{h}_0, \mathbf{h}_1, \mathbf{h}_2) \leftarrow \mathbf{M} + \mathbf{M}^\mathsf{T} - \left(t + \sqrt{\tilde{g}_S^2 + \mathbf{m}^\mathsf{T}\mathbf{m}}\right)\mathbf{I}$

$h^{-1} \leftarrow |\mathbf{h}_0, \mathbf{h}_1, \mathbf{h}_2|^{-1}$

$d_0 \leftarrow |-\mathbf{m}, \mathbf{h}_1, \mathbf{h}_2|$

$d_1 \leftarrow |\mathbf{h}_0, -\mathbf{m}, \mathbf{h}_2|$

$d_2 \leftarrow |\mathbf{h}_0, \mathbf{h}_1, -\mathbf{m}|$

return $h^{-1}(d_0, d_1, d_2)$

5. Implementation

We implemented our method in C++ (see supplemental material for code). The starting point for our implementation does not use vectorization.

Many applications such as co-rotational elasticity or as-rigid-as-possible deformation require best-fit rotations for many (e.g., millions) of cross-covariance matrices. We further optimized our code using single-core SIMD vectorization using the AVX2 standard. Rather than attempt to utilize esoteric instructions (e.g, `_mm_permute_ps`), we follow a straight-forward best practice guideline [Pü11], which advises to rearrange the input data layout to accommodate trivial vectorization of each floating point operator in the scalar code. For example, the scalar product `a = b*c` becomes the 8-wide vectorized product `a = _mm256_mul_ps(b,c)`. AVX supports eight simultaneous single-precision floating point operations. Therefore, we expect as input a n-long list of 3×3 $\mathbf{M}$ matrices laid out in memory as $\{m^0_{0,0}, \ldots, m^7_{0,0}, m^0_{0,1}, \ldots, m^7_{0,1}, \ldots, m^0_{2,2}, \ldots, m^7_{2,2}, m^8_{0,0}, \ldots\}$, without loss of generality we assume n is divisible by eight. To best utilize this vectorization any up-/down-stream functions should be similarly rearranged to read/write data in this order. If not, a (small) reshuffling overhead is incurred.

Finally, we can further parallelize our streaming implementation using a parallel for loop (e.g., LIBIGL's `igl::parallel_for`) to take advantage of multi-core CPUs. This was omitted during performance benchmarking.

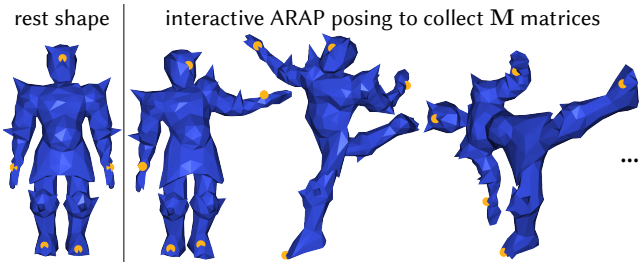
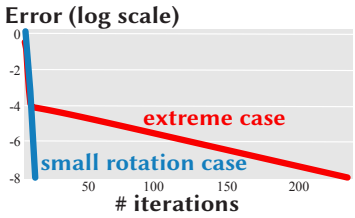


Figure 3: *Rather than test on random matrices, our performance benchmark is designed to capture the distribution of matrices that occur in common graphics applications such as as-rigid-as-possible deformation.*

We include our implementation (and those of many previous works) in our supplemental material and intend to release our optimized code open source for the community to use. https://github.com/ErisZhang/fast-rotation-fitting.git

6. Experiments & results

We now compare our approach to existing ones from the literature. Note that the actual performance of different algorithms will depend on the specific implementation, the hardware, and the data (i.e. the cross-correlation matrix). A central and long-term contribution of our work is the implementation of many different algorithms in one framework. This allows users to select the best algorithm for their platform and their data distribution. It also fosters continuing development of rigid alignment algorithms. As an example, we mainly focus on data coming from real applications, as this is likely different than random distributions. The inset figure shows the dramatic difference between fast convergence for small rotations and slow convergence for an extreme case (random cross-covariance matrix).



6.1. Cross-covariance matrix dataset

Our main objective is to speed up local rotation estimation for graphics applications. It is conceivable that the cross-covariances $\mathbf{M}$ for which the optimal rotations have to be computed have characteristic distributions in each application. As discussed, our approach may be faster for some inputs and, consequently, we want to test it on inputs that resemble the actual situation in the application.

We have selected as-rigid-as-possible (ARAP) surface deformation [SA07] and a co-rotational elasticity simulation based on the geometric model by Chao et al. [CPSS10] (see Fig. 1). For both techniques we use the implementations publicly available in LIBIGL [JP*16]. For ARAP, we have performed an interactive editing session (see Figure 3) and recorded the matrices associated with

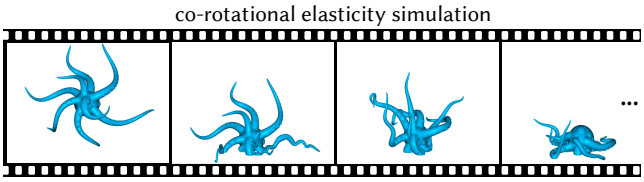


Figure 4: *The least-squares rotations fit to each element of this tetrahedral mesh change only slightly each frame of the animation, not to mention each iteration of the non-linear solver. Our one-step warm-start algorithm takes advantage of this.*

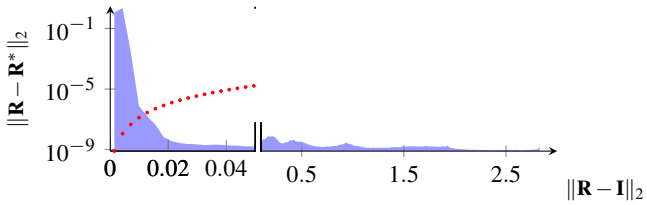


Figure 5: *We collect two datasets of co-variance matrices* $\mathbf{M}$ *as a benchmark for least-squares fitting a rotation matrix* $\mathbf{M}$*. Light blue shows a histogram over the differences of optimal rotation* $\mathbf{R}$ *and identity for data collected from an interactive ARAP session, which contained more data whose optimal rotations are further from identity. Over all our data, the left part of the plot accounts for more than 90% of the data. The red dots indicate the error of a rotation* $\mathbf{R}^*$ *based on a single Newton step.*

each of the vertex neighborhoods. For the volumetric simulation, we drop a volumetric octopus mesh above a ground plane and collected deformation gradient matrices associated with each tetrahedron (see Fig. 4).

This collection process ensures that matrices come from actual examples and represent the warm-start-able nature of the computations: consecutive frames have similar deformations and thus similar best fit rotations. This similarity is further compounded by the iterative nature of the local-global solvers employed.

Before writing the matrices into a file we have applied the best fitting rotation from the previous iteration or frame (if any), which we expect to be close to the sought best fit rotation. Recall that our approach could make direct use of this guess for potential speed up. To provide other approaches with a similar advantage, we factor out this rotation, so that the rotational component is potentially small, and all methods may profit from estimating a rotation that is close to identity.

Fig. 5 shows the proximity of the optimal rotation to identity for the case of the ARAP data – the optimal rotations for the data from the co-rotation method is even more concentrated around identity and therefore omitted. We also analyzed the condition numbers of the collected co-variance matrices. In case of ARAP they concentrated around 10^5. The reason for this is likely that ARAP collects data only from surfaces, so that the variation in normal direction is mostly small compared to the tangents. As we will see, our method is unaffected by this bad conditioning. The condition numbers for the co-rotation data are in the range $10^1 - 10^3$. So while the data sets are similar in the sense that the optimal rotations are biased towards identity, they are dissimilar in the distribution of their anisotropic scaling.

For completeness, we also verified the validity of our solver on matrices with 9 coefficients selected uniformly random in the interval $[0,1]$.

6.2. Experimental results

We tested our proposed Cayley-based methods on a variety of compilers, CPU setups, and operating systems (e.g., Linux, Mac, Windows). Based on the value of c, we name them Cayley ($c = \mathrm{tr}(\mathbf{M})$), Cayley Conservative ($c = \sqrt{\mathrm{tr}(\mathbf{M})^2 + \mathbf{m}^\mathsf{T}\mathbf{m}}$) and Cayley Gershgorin. Note that mathematically Cayley is equivalent to [KKB18] that uses a single Newton step with quarternions. In all cases, the general result is the same. As an aside, we found that *automatic* vectorization varies significantly across compilers, but in every case the manually written AVX code was fastest. Experimental reports are conducted on a Mac laptop equipped with a 2.3 GHz Core Intel i9 CPU with 16GB of memory using the default clang compiler (Apple LLVM version 12.0.0). To conduct fairer comparisons, we omit the use of multi-core parallelization unless otherwise noted.

For the data collected from the actual application examples we test the number of iterations that are necessary for convergence. In particular, we test the accuracy based on a single Newton step. Our analysis show that for over 90% of the data a single Newton step results in a Frobenius norm error of $< 10^{-5}$, This is visualized in the scatter plot in Fig. 5. In addition, in the application examples some error is acceptable, because it will be corrected for in subsequent iterations. In the following we include timings based on this idea and term this approach *warm-start*.

In Table 1, we compare our core non-vectorized methods with and without the warm-start to various baseline methods. We take the SVD computed with Eigen v3.3.7 [GJ*19] as a reference. We also implemented Horn's quaternion-based method [Hor87] using Eigen, employing its 4×4 eigen solver. We also include the publicly available implementations from Wu et al. [WLZL18], Sifakis et al. [MST*11] and Kugelstadt et al. [KKB18]. Across the different compilers we tried, Eigen's SVD performance varied substantially more than our method or the others. In the interest of fairness, we report timings on the Mac+clang configuration where Eigen's performance was best. Despite reporting relative speedups for this specific configuration in Table 1, the upshot of the comparison holds on all machines we tried on.

On the uniform random data generated from restricting the Euler angles of the rotation to $[-\pi/1.2, \pi/1.2]$, Cayley Gershgorin converges in 3 iterations on average and is $2.7\times$ faster. This is already faster than all the other methods tested. On the randomly generated cross-covariance matrices, Cayley Gershgorin converges in 5 iterations on average and is $1.6\times$ faster. However, the true performance boost comes when exploiting the existence of a good starting point, as is automatically done by selecting the data sets collecting from the real applications. In this case Cayley and Cayley Conservative are $12.9\times$ and $12.3\times$ faster than Eigen's SVD, while the timing of other methods is almost unchanged. Although Cayley Conservative

Method	Speedup over Eigen 3 × 3 SVD
[Hor87]	0.7×
[MBCM16]	1.5×
Polar Decomposition	1.5×
[WLZL18]	1.6×
Cayley Gershgorin	2.7×
Cayley Conservative + warm-start	12.3×
Cayley + warm-start	12.9×

Table 1: *Using SVD computed with Eigen [GJ*19] as a baseline (:= 1.0×), we compare the relative speedups of various methods. Our (non-vectorized) "Cayley + warm-start" implementation achieves a 12.9× speedup.*

Method (+ avx)	Speedup over Eigen	Speedup over [MST*11]
[MST*11]	10.4×	1.0×
Cayley Gershgorin	15.7×	1.5×
Cayley Conservative + warm-start	63.1×	6.0×
Cayley + warm-start [KKB18]	70.1×	6.7×

Table 2: *Many applications in graphics already take advantage of the high-performance avx-vectorized code from Sifakis et al. [MST*11] (10.4× speedup over Eigen). Treating this method as a baseline, the family of Cayley-based methods achieves superior speedups.*

is slightly slower, it always achieves better accuracy (see Figure 2) and suffers 73% fewer failure cases in our interactively collected dataset. If using Cayley Gershgorin, we found no failures cases in our experiments. However, for a local-global solver like ARAP, perfect convergence at each local step is not strictly required since as long as the energy is reduced in each step it will converge to a local minimum. We found no noticeable effect of the choice of rotation on the local minimum attended (see Figure 7).

Next, in Table 2, we compare to the SIMD vectorized SVD-method of Sifakis et al. [MST*11]. Keeping Eigen's SVD as a baseline, their method using AVX achieves a 10.4× speedup, slightly slower than our non-vectorized, warm-start Cayley code. In contrast, adding AVX vectorization to our warm-start Cayley-based methods, we run more than 60× faster than Eigen's SVD or more than 6× faster than the AVX SVD code of Sifakis et al. [MST*11].

6.3. Applications

To qualitatively ensure the correctness of our implementation we attach it as the local-rotation fitting subroutine to a number of standard graphics applications. For interactive as-rigid-as-possible deformation and co-rotational elasticity simulation, solving the corresponding sparse linear system should scale super-linearly with the number of vertices in the mesh. This suggests that this *global* step would be a bottleneck compared to the linear-complexity *local* step of fitting rotations. However, using a standard SVD-based method, the local fitting dominates the wall-clock performance. In Fig. 6, we show that using our Cayley method with the warm-start for the

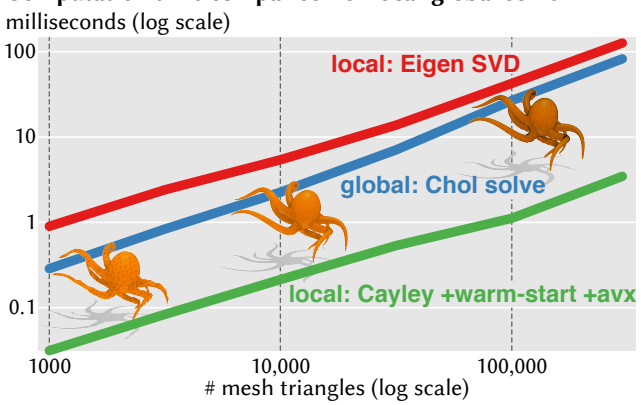


Figure 6: *Typical implementations of the local-global solver (cf. [SA07]) are bottlenecked by the least-squares rotation fitting despite the sparse back-substitution theoretically having larger computational complexity. With our fast update, this is no longer the case. The local step is an order of magnitude faster than the global step.*

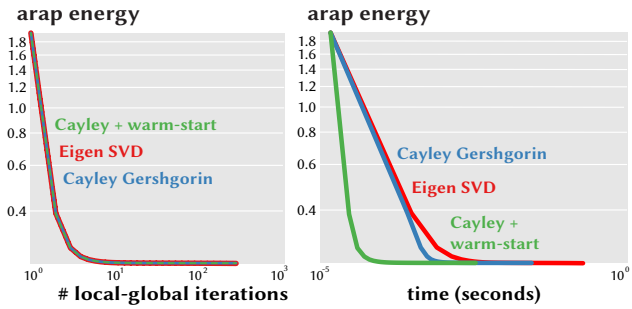


Figure 7: *The choice of rotation-fitting algorithm, while having little effect on the converged solution of a local-global ARAP solver, has a large impact on wall clock time to solution. Since the Cayley method only works for small rotations, for fair comparison, we warm-start the first local step of the Cayley method with Eigen's SVD (for maximum accuracy).*

local step, the rotation fitting is significantly faster than the global step.

7. Discussion

We have analyzed the timings based on two specific deformation techniques, one surface-based and one volumetric. The energies used and the local-global minimization approach are prototypical and represent a much wider class of non-linear deformation and simulation techniques [LBK17, BML*14, BDS*12, Vax14]. Also related problems that require quickly estimating the best fitting rotation in 3D will profit from our approach. To name a few specific areas: real-time character skinning [JBK*12, LH16], animation compression [LD12], fluid simulation [YT13], rigid registration (e.g., iterative closest point method) [BM92], pose estima-

tion from multiple cameras [Ume91], chromosome reconstruction [ZDH*18].

Our proposed Cayley-based approaches have two fundamental features:

1. It is inherently simpler than approaches that require computing the SVD of a 3×3 matrix or an eigenvector of a 4×4 matrix. The small number of elementary operations makes sure the Cayley-based approaches will likely be useful even in the light of ever changing hardware. It also eases optimized implementation for specific platforms, currently existing in the future. In particular, it requires no approximation of trigonometric functions.
2. It allows exploiting the common situation that a good estimate for the rotation is available. It appears that other approaches generally fail to provide this feature, or their degradation is catastrophic for the application, such as providing approximate rotation matrices that are not orthogonal. We tried introducing accuracy / computation time tradeoffs in the SVD approaches but failed to find a working solution. This might be an avenue for future research, as the complete SVD is useful in some applications.

8. Acknowledgements

This work is funded in part by the Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ EXC-2046/1, NSERC Discovery (RGPIN2017–05235, RGPAS–2017–507938), New Frontiers of Research Fund (NFRFE–201), the Ontario Early Research Award program, the Canada Research Chairs Program, and gifts by Adobe Systems. We thank Silvia Sellán for proofreading and all the anonymous reviewers for their helpful comments and suggestions.

References

[AHB87] ARUN K. S., HUANG T. S., BLOSTEIN S. D.: Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-9*, 5 (Sep. 1987), 698–700. 2

[BDS*12] BOUAZIZ S., DEUSS M., SCHWARTZBURG Y., WEISE T., PAULY M.: Shape-up: Shaping discrete geometry with projections. *Comput. Graph. Forum 31*, 5 (Aug. 2012), 1657–1667. 8

[BM92] BESL P. J., MCKAY N. D.: A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell. 14*, 2 (Feb. 1992), 239–256. 8

[BML*14] BOUAZIZ S., MARTIN S., LIU T., KAVAN L., PAULY M.: Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph. 33*, 4 (July 2014), 154:1–154:11. 8

[BX08] BYERS R., XU H.: A new scaling for newton's iteration for the polar decomposition and its backward stability. *SIAM Journal on Matrix Analysis and Applications 30*, 2 (2008), 822–843. 2

[CPSS10] CHAO I., PINKALL U., SANAN P., SCHRÖDER P.: A simple geometric model for elastic deformations. *ACM Trans. Graph. 29*, 4 (July 2010), 38:1–38:6. 1, 2, 6

[ELF97] EGGERT D. W., LORUSSO A., FISHER R. B.: Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications 9*, 5 (Mar 1997), 272–290. 2

[FH86] FAUGERAS O. D., HEBERT M.: The representation, recognition, and locating of 3-d objects. *Int. J. Rob. Res. 5*, 3 (Sept. 1986), 27–52. 2

[GJ*19] GUENNEBAUD G., JACOB B., ET AL.: Eigen v3.3, 2019. http://eigen.tuxfamily.org. 7, 8

[GKWT98] GUÉZIEC A., KAZANZIDES P., WILLIAMSON B., TAYLOR R. H.: Anatomy-based registration of ct-scan and intraoperative x-ray images for guiding a surgical robot. *IEEE Transactions on Medical Imaging 17*, 5 (1998), 715–728. 2

[HHN88] HORN B. K. P., HILDEN H. M., NEGAHDARIPOUR S.: Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A 5*, 7 (Jul 1988), 1127–1135. 2

[Hig86] HIGHAM N. J.: Computing the polar decomposition—with applications. *SIAM Journal on Scientific and Statistical Computing 7*, 4 (1986), 1160–1174. 2

[Hor87] HORN B. K. P.: Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A 4*, 4 (Apr 1987), 629–642. 2, 7, 8

[JBK*12] JACOBSON A., BARAN I., KAVAN L., POPOVIĆ J., SORKINE O.: Fast automatic skinning transformations. *ACM Trans. Graph. 31*, 4 (July 2012), 77:1–77:10. 8

[JHD18] JAUCH M., HOFF P. D., DUNSON D. B.: Random orthogonal matrices and the cayley transform, 2018. `arXiv:arXiv:1810.02881`. 2

[JP*16] JACOBSON A., PANOZZO D., ET AL.: libigl: A simple C++ geometry processing library, 2016. http://libigl.github.io/libigl/. 6

[Kab76] KABSCH W.: A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A 32*, 5 (1976), 922–923. 2

[KKB18] KUGELSTADT T., KOSCHIER D., BENDER J.: Fast corotated fem using operator splitting. *Computer Graphics Forum (SCA) 37*, 8 (2018). 2, 4, 7, 8

[LBK17] LIU T., BOUAZIZ S., KAVAN L.: Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Trans. Graph. 36*, 4 (May 2017). 8

[LD12] LE B. H., DENG Z.: Smooth skinning decomposition with rigid bones. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 199:1–199:10. 8

[LH16] LE B. H., HODGINS J. K.: Real-time skeletal skinning with optimized centers of rotation. *ACM Trans. Graph. 35*, 4 (July 2016), 37:1–37:10. 8

[Mar87] MARKLEY L.: Attitude determination using vector observations and the singular value decomposition. *Journal of Astronautical Sciences 38* (11 1987), 245–258. 2

[MBCM16] MÜLLER M., BENDER J., CHENTANEZ N., MACKLIN M.: A robust method to extract the rotational part of deformations. In *Proceedings of the 9th International Conference on Motion in Games* (2016), pp. 55–60. 8

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3 (July 2005), 471–478. 1

[MM11] MLADENOVA C. D., MLADENOV I. M.: Vector decomposition of finite rotations. *Reports on Mathematical Physics 68*, 1 (2011), 107 – 117. 2

[Mor97] MORTARI D.: Esoq: A closed-form solution to the wahba problem. *Journal of the Astronautical Sciences 45* (1997), 195–204. 2

[MST*11] MCADAMS A., SELLE A., TAMSTORF R., TERAN J., SIFAKIS E.: *Computing the Singular Value Decomposition of 3 × 3 matrices with minimal branching and elementary floating point operations*. Tech. Rep. TR1690, University of Wisconsin - Madison, May 2011. 2, 7, 8

[Nor08] NORRIS A.: Euler-rodrigues and cayley formulae for rotation of elasticity tensors. *Mathematics and Mechanics of Solids 13*, 6 (2008), 465–498. 2

[Pü11] PÜSCHEL M.: How to write fast numerical code, 2011. 6

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2007), SGP '07, Eurographics Association, pp. 109–116. 1, 2, 6, 8

[Sch66] SCHÖNEMANN P. H.: A generalized solution of the orthogonal procrustes problem. *Psychometrika 31*, 1 (Mar 1966), 1–10. 2

[SHR17] SORKINE-HORNUNG O., RABINOVICH M.: Least-squares rigid motion using svd, 2017. 2

[Shu78] SHUSTER M. D.: Approximate algorithms for fast optimal attitude computation. In *AIAA Guidance and Control Conference* (1978), pp. 7–9. 2

[Suz05] SUZUKI J.: The lost calculus (1637-1670): Tangency and optimization without limits. *Mathematics Magazine 78*, 5 (2005), 339–353. 4

[Ume91] UMEYAMA S.: Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*, 4 (April 1991), 376–380. 2, 9

[Vax14] VAXMAN A.: A projective framework for polyhedral mesh modelling. *Comput. Graph. Forum 33*, 8 (Dec. 2014), 121–131. 8

[WLZL18] WU J., LIU M., ZHOU Z., LI R.: Fast rigid 3d registration solution: A simple method free of svd and eigen-decomposition, 2018. arXiv:arXiv:1806.00627. 7, 8

[WSV91] WALKER M. W., SHAO L., VOLZ R. A.: Estimating 3-d location parameters using dual number quaternions. *CVGIP: Image Underst. 54*, 3 (Oct. 1991), 358–367. 2

[WV06] WEDIN P.-A., VIKLANDS T.: *Algorithms for 3-dimensional Weighted Orthogonal Procrustes Problems*. Tech. Rep. UMINF-06.06, UmeåUniversity, 2006. 2

[YT13] YU J., TURK G.: Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Trans. Graph. 32*, 1 (Feb. 2013), 5:1–5:12. 8

[YZ13] YANG Y., ZHOU Z.: An analytic solution to wahba's problem. *Aerospace Science and Technoloy 30*, 1 (2013), 46 – 49. 2

[ZDH*18] ZHU G., DENG W., HU H., MA R., ZHANG S., YANG J., PENG J., KAPLAN T., ZENG J.: Reconstructing spatial organizations of chromosomes through manifold learning. *Nucleic Acids Research 46* (2018), e50. 9

Appendix A: Derivatives

Here we provide the gradient and Hessian of the trace function in Cayley parameterization given in Eq. 14. The derivative of the denominator $1+\|\mathbf{z}\|^2$ is $2\mathbf{z}$. For the numerator we find

$$2\mathbf{m}+\left(2(\mathbf{M}+\mathbf{M}^\mathsf{T})-2\,\mathrm{tr}(\mathbf{M})\mathbf{I}\right)\mathbf{z} \tag{34}$$

The quotient rule for differentiation then yields the gradient as

$$\nabla g(\mathbf{z}) = \frac{2}{1+\mathbf{z}^\mathsf{T}\mathbf{z}}\left(\mathbf{T}\mathbf{z}+\mathbf{m}\right) - \frac{2t(\mathbf{z})}{(1+\mathbf{z}^\mathsf{T}\mathbf{z})^2}\mathbf{z} \tag{35}$$

where we have used the abbreviations

$$\mathbf{T} = \mathbf{M}+\mathbf{M}^\mathsf{T}-\mathrm{tr}(\mathbf{M})\mathbf{I} \tag{36}$$

and

$$t(\mathbf{z}) = (1-\mathbf{z}^\mathsf{T}\mathbf{z})\,\mathrm{tr}(\mathbf{M})+2\mathbf{z}^\mathsf{T}\mathbf{M}\mathbf{z}+2\mathbf{m}^\mathsf{T}\mathbf{z}. \tag{37}$$

The gradient of the latter function is

$$\nabla t(\mathbf{z}) = 2(\mathbf{T}\mathbf{z}+\mathbf{m}) \tag{38}$$

leading to the Hessian as

$$H_g(\mathbf{z}) = 2\left(1+\mathbf{z}^\mathsf{T}\mathbf{z}\right)^{-1} \left(\mathbf{T}-\frac{1}{1+\mathbf{z}^\mathsf{T}\mathbf{z}}\left(t(\mathbf{z})\mathbf{I}+2(\mathbf{T}\mathbf{z}+\mathbf{m})\mathbf{z}^\mathsf{T}\right)+\frac{2t(\mathbf{z})}{(1+\mathbf{z}^\mathsf{T}\mathbf{z})^2}\mathbf{z}\mathbf{z}^\mathsf{T}\right). \tag{39}$$

For evaluating the Hessian at $\mathbf{z}=\mathbf{0}$ we note that $t(\mathbf{0})=1$.