Linear geometric algebra rotor estimator for efficient mesh deformation

Jin Wu*, Mauricio Lopez2✉, Ming Liu*, Yilong Zhu*
1Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong, People’s Republic of China
2Buenos Aires Computer Corporation, Buenos Aires, Argentina
✉E-mail: mclopez@outlook.com

Abstract: The authors solve the problem of estimating the best rotation aligning two sets of corresponding vectors (also known as Wahba's problem or point cloud registration). The proposed method is among the fastest methods reported in recent literatures, moreover it is robust to noise, accurate and simpler than most other methods. It is based on solving the linear equations derived from the formulation of the problem in Euclidean Geometric Algebra. The authors show its efficiency in two applications: the as-rigid-as-possible (ARAP) surface modelling and the more smooth rotation enhanced ARAP mesh animation which is the only method capable of deforming surface modes with quality of tetrahedral models. Mesh deformation is a key technique in games, automated construction and robotics. The ARAP technique along with its improved variants, although have been extensively studied, can still not be achieved efficiently. Linear geometric algebra based rotor solution proposed in this study gives another perspective of the kernel problem. This, however, not only improves the real performance of the three-dimensional mesh deformation, but also provides a brand new computationally efficient solution to the Wahba's problem and point cloud registration, which has been closely related to the automation science and engineering.

1 Introduction

The concept of mesh deformation arises from the emerging needs of the highly dynamic three-dimensional (3D) computer animation, which has been extensively employed in computer games, visualisation, computer aided design and intelligent manufacturing [1]. Another similar terminology, i.e. the surface flattening has also been widely studied in automated construction [2]. In principle, the target of these approaches is using 2D meshes to construct specific 3D models. As complex 3D models own quite abundant surface details, the deformation process is time consuming. The as-rigid-as-possible (ARAP, [3]) models such process by introducing various local rigid transformations. This determines that the mathematical model of ARAP is similar with the form of point-point matching, i.e. a classical problem aligning two vector sets (may be of different numbers of points) [4]. This problem is renowned as the Wahba's problem [5] and point cloud registration [6] in aerospace engineering and robotics, respectively. Since the proposal of Wahba's problem in 1965, many robust and computationally efficient algorithms have been proposed. Early efforts have been paid by Shuster, Markley, Mortari who developed different solvers based on various attitude representations like quaternion, rotation matrix, Rodrigues vector and so on [7–9]. Recently, Wu et al. studied fast results for online determination of attitude using quaternion formulation of the Wahba's problem [10, 11]. Related applications like registration using iterative closest point also highly rely on the efficiency of matching [12].

Historically, representative solutions mainly cover the single-point batch processing of point-set alignment. However, it is required in many applications that the solution should be either accurate, robust or continuous, so that the animated meshes can be smoothly placed. A recent method using the gradient descent algorithm (GDA) of the Wahba's problem solves this problem but is not linear [13]. It has also been proposed in [14] that some sub-optimal filters can produce continuous results but the accuracy is not optimal. The target of ARAP-based mesh deformation is similar to Wahba's problem but as the amount of meshes are usually huge. Therefore, the practitioners require a new method that is not only accurate, robust, continuous but also, ARAP-friendly. Guided by this requirement, in the paper, the main contributions are:

i. A geometric algebra (GA) based formulation of the kernel problem in ARAP has been proposed, which is intuitive, linear and simple.
ii. It is able for us to construct continuous Newton algorithm to converge to the ideal solution within only several simple iterations.
iii. ARAP and its improved variants are combined with the proposed method in a friendly manner allowing real-time performance of accurate mesh animation.

We thus study the analytical forms of all the operations required. Through derivations we are able to give linear solutions with simple matrix manipulations. It has been compared with representatives of Wahba’s problem and point cloud registration and has shown that it owns the same accuracy but less computational burden. The performance of the mesh animation also proves that the accuracy belongs to the optimal ones while the computational efficiency is the highest.

This paper is then structured as follows. Section 2 provides the details of the proposed derivations and solution. Section 3 investigates the application of the method to the ARAP and its variants. Section 4 gives experimental results and comparisons and the concluding remarks are drawn in Section 5.

2 Fast rotor estimation algorithm

2.1 Preliminaries

All the rotation matrices belong to the space of special orthogonal groups, i.e. for 3 × 3 case, the 3D $\mathbf{SO}(3) := \{ R | R \in \mathbb{R}^{3 \times 3}, R^T R = I, \ \det(R) = 1 \}$. The GA allows for the geometric product of two 3 × 1 vectors $a$ and $b$ such that

$$ab = a \cdot b + a \times b$$

(1)
in which \( \cdot \) and \( \times \) are inner (dot) and outer (cross) products. Note that the outer product \( a \times b \) is a bivector also in the 3D GA space \( G_3 \). This further gives a normalised version within the scope of the even subalgebra \( G_3^+ \), which produces a group of rotor (spin) vectors

\[
SP(3) = \{ R | R \in G_3^+, \tilde{R}R = \tilde{R}R = 1 \}
\]  

(2)

where \( \tilde{R} = b \times a \) gives the reverse of \( R \) supposing that \( R = a \times b \). We use \( (R)_k \) to extract the \( k \)th element of \( R \).

### 2.2 Proposed work

Given two sets of \( n \) points in \( P = \{ p_j\}^n_{j=1}, Q = \{ q_j\}^n_{j=1} \), we try to minimise the following error:

\[
E(R) = \min_{R \in SP(3)} \sum_{j=1}^{n} c_j \| q_j - R p_j \tilde{R} \|^2
\]

(3)

where \( c_j \) is a scalar weight such that \( \sum_{j=1}^{n} c_j = 1 \). In that form, the minimisation is non-linear in \( R \). Following some manipulations, we can make it linear by multiplying by \( R \) on the right

\[
q_j R - R p_j = 0
\]

(4)

so the constraint \( R \tilde{R} = 1 \) is implicit in the energy \( E(R) \), although we will need to project \( R \) back to the manifold through normalisation. Let \( F \) be a column vector of functions which square amounts to the energy \( E(R) \)

\[
F = \begin{bmatrix} F^1 \\ \vdots \\ F^n \end{bmatrix}
\]

(5)

The energy \( E(R) \) can be expressed as the matrix product

\[
E(R) = \tilde{F}^T F
\]

(6)

where

\[
F_i(R) = \sqrt{c_j} (R p_j - q_i R)
\]

(7)

\( F_i \) is producing a multivector with the basis \( \{ e_i, e_i e_i, e_i e_i e_i \} \).

The critical points of the energy \( E(R) \) where it is minimised can be found solving \( \nabla E(R) = 0 \), where the gradient has the following form:

\[
g = \nabla E(R) = \nabla (F^T F) = 2 \tilde{J} F
\]

(8)

where \( J \) is the Jacobian matrix of \( F \). Since the energy \( E(R) \) is purely quadratic in \( R \), the solution for \( \nabla E(R) = 0 \) can be found by solving a linear system of equations. An optimal rotor is in the null space of a particular matrix derived from \( \tilde{J} F = 0 \). An easy way to obtain a solution in the null-space is using one iteration of Newton's method. Due to linearity of the equation \( \tilde{J} F = 0 \) w.r.t. \( R \), one iteration suffice for obtaining a solution. The optimal increment for \( E(R) \) is given by the Newton formula \( \Delta R = H^{-1} \nabla E(R) \), provided that the inverse of Hessian matrix \( H^{-1} \) exists. Noting that \( J \) does not depend on \( R \), i.e. constant, the Hessian matrix \( H \) takes the simple form

\[
H = \frac{\partial (2 \tilde{J} F)}{\partial R} = 2 \tilde{J}^T \frac{\partial F}{\partial R} = 2 \tilde{J}^T J
\]

(9)

where \( H \) is independent of \( R \). An optimal solution can then be found by solving a linear system

\[
R' = R_0 - H^{-1} g(R_0)
\]

(10)

where \( R_0 \) is an initial rotor and \( g(R_0) \) is the gradient of \( E \) evaluated at \( R_0 \). Of course, the choice of initial rotor \( R_0 \) has a particular effect on finding a local minimum. The energy \( E(R) \) is non-convex, its shape is similar to a sinusoidal wave with infinitely many points at minimum energy value, each one at rotor \( R_i = e^{-i \phi_i} \) for \( i \in \mathbb{N} \), being \( B \) the optimal unique attitude bivector and \( \theta \) an optimal angle with minimal absolute value. The choice of \( R_0 \) leads to find a local solution close to it, the choice \( R_0 = [1, 0, 0, 0]^T \) is optimal from the computational point of view and is also desirable because lead us to find solutions close to the identity rotor. Since there are infinite solutions, \( H \) is a singular matrix. A simple and efficient way to find its pseudo-inverse is to add a very small regularisation term to \( E(R) \). Our strategy is to introduce a new constant rotor \( R_0 \), within a new term \( \epsilon \| R - R_i \|^2 \), which we can express as matrix product

\[
F_i^T F_i
\]

(11)

with Jacobian

\[
J_i = \frac{\partial F_i}{\partial R} = \sqrt{c_i} I
\]

(12)

where \( I \) is the identity matrix. The energy now looks like this

\[
E(R) = \min_{R \in SP(3)} \sum_{i=1}^{n} c_i \| R p_i - q_i R \|^2 + \epsilon \| R - R_i \|^2
\]

(13)

where \( 0 < \epsilon < 1 \) is constant but small (we use \( \epsilon = 10^{-5} \)). The term \( \epsilon \| R - R_i \|^2 \) is a regularisation term that helps on finding a solution biased towards \( R_i \), so one can think \( R_i \) as a constant rotor which is pointing \( R \) towards a given minimum point. With this regularisation strategy we can not only find the pseudo-inverse of \( H \) but also we can make sure to obtain a stable \( R \) that is not stuck when the minimal point is close to some far angle such as \( \pm \pi \) radians. For instance by taking \( R_i \) as the previous rotor calculated with Newton iteration, we can check that \( R \) is at some minimum point, because in the next iteration we should obtain the same \( R \). In the edge cases near to \( \pm \pi \) where the \( R \) may not be at minimum, the previous solution \( R_i \) will direct the next solution to a minimum, commonly in one to three iterations. So, regularisation term can be seen as a feedback for reaching a stable minimal \( R \). We will see this more clearly in the algorithm. Keep in mind that we are solving the linear system just once (i.e. we are inverting \( H \) once) because the \( R_i \) term is affecting only the gradient. The gradient and Hessian are now

\[
g = \nabla (F^T F + F_i^T F_i) = 2 (\tilde{J}^T F + J_i^T F_i)
\]

(14)

\[
H = 2 \frac{\partial (\tilde{J}^T F + J_i^T F_i)}{\partial R} = 2 (\tilde{J}^T J + I)
\]

(15)

The optimal rotor is as before

\[
R' = R_0 - H^{-1} g(R_0)
\]

(16)

The Hessian matrix \( H \) resembles now a form of Tikhonov regularisation, whose inverse \( H^{-1} \) approaches to the Moore-Penrose pseudo-inverse as \( \epsilon \) approaches to zero. We now need to determine how to obtain \( R_0 \). An stand-alone algorithm can take \( R_i \) as the previous rotor calculated with Newton iteration. But we also can take \( R_i \) to be a previous solution in a simulation. We have used both choices in our experiments, with excellent results in both cases. We noticed in a simulation that the previous rotor helps to get the next one preserving the same sense of rotation. We now
Jacobian of

2.2.2 Form of the Jacobians: Let us recall that each term of the sum \( \sum_{j=1}^{n} \frac{\| R p_j - q R \|}{\| R \|} \) can be expressed in matrix form as function \( F^T F \) where

\[
F^i(R) = \sqrt{c_j}(R p_j - q) R
\]

(17)

\( F^i \) is producing a multivector with the basis \( \{ e_i, e_2, e_3, e_4 \} \). The Jacobian of \( F^i \) is \( \partial F^i / \partial R \)

\[
J^i = \begin{bmatrix} \frac{\partial F^i}{\partial e_1} & \frac{\partial F^i}{\partial e_2} & \frac{\partial F^i}{\partial e_3} & \frac{\partial F^i}{\partial e_4} \end{bmatrix}
\]

(18)

Solution to \( H \Delta R = g(R_0) \) looks like this

\[
\left( \sum_{j=1}^{n} J^T F^j \right) R = - \sum_{j=1}^{n} J^T F^j R' = R_0 + \Delta R
\]

(19)

where we should normalise the rotor \( R' \) for sending it back to the manifold.

2.2.2 Form of the Jacobians: The Jacobian \( J^j \) is \( 4 \times 4 \) which four columns are the directional derivatives of \( F^j = \sqrt{c_j}(R p_j - q) R \)

\[
\frac{\partial F^j}{\partial e_1} = \sqrt{c_j} (p_j - q_j) e_1
\]

(20)

\[
\frac{\partial F^j}{\partial e_2} = \sqrt{c_j} (-p_j + q_j) e_2
\]

(21)

\[
\frac{\partial F^j}{\partial e_3} = \sqrt{c_j} (p_j + q_j) e_3
\]

(22)

\[
\frac{\partial F^j}{\partial e_4} = \sqrt{c_j} (-p_j - q_j) e_4
\]

(23)

Thus, the leading symmetric matrix \( H^j = J^T J^j \) has a simple form.

The symmetric matrix \( H^j = J^T J^j \) has a simple form in (23) provided that

\[
S_i = (q_j + p_j) e_i, \quad S_j = (q_j + p_j) e_j, \quad S_k = (q_j + p_j) e_k, \quad D_i = (p_j - q_j) e_i, \quad D_j = (p_j - q_j) e_j, \quad D_k = (p_j - q_j) e_k
\]

(24)

(see (25))

Since \( H^j \) is symmetric only 10 out of 16 elements need to be actually computed. The system now looks like this

\[
\left( \sum_{j=1}^{n} H^j \right) \Delta R = - \sum_{j=1}^{n} J^T F^j
\]

(26)

Solving just one iteration of the Newton suffice for solving the system, i.e. 19. In the iteration the symmetric matrix \( H = \sum_{j=1}^{n} H^j \) is constant, but the gradient at right-hand-side depends on \( R \) since \( F^j \) depends on it.

We now can proceed to optimise a little more our method. We incorporate the fixed initial guess \( R_0 = 1 \) into the gradient

\[
F^j = \sqrt{c_j}(R p_j - q) R^j F^j = \sqrt{c_j}(p_j - q_j)
\]

(27)

\[
D_i + D_j + D_k
\]

(28)

\[
g^j = J^i F^j = e_i
\]

(29)

Notice that the first row of \( H^j \) is equal to \( g^j \), so \( g^j \) does not need to be explicitly calculated. Now we introduce the regularisation term \( F_2 \) and proceed to replace again \( R_0 \) on it, which look like this

\[
F_2 = \sqrt{c}(R_2 - R) F_2 = \sqrt{c}(1 - \{ R \})
\]

(30)

The final iteration is reduced to

\[
\left( \sum_{j=1}^{n} H^j + \epsilon I \right) \Delta R = - \sum_{j=1}^{n} g^j - d
\]

(31)

Notice that all terms are constant except \( d \) which depends on the previous iteration. Note that the matrix \( \sum_{j=1}^{n} H^j + \epsilon I \) does not depend on \( R \) and its inverse can be precomputed. Since the matrix \( \sum_{j=1}^{n} H^j + \epsilon I \) and the vector \( \sum_{j=1}^{n} g^j \) are constant, the inner loop amounts to compute a cheap matrix–vector multiplication. Also notice that since the matrix \( \sum_{j=1}^{n} H^j \) is symmetric, only ten components of \( J \) need to be actually computed and since the matrix \( \sum_{j=1}^{n} H^j + \epsilon I \) is symmetric its inverse is symmetric and also cheap to compute.

3 Applications

3.1 ARAP

Given two meshes \( P \) and \( Q \) consisting of vertices \( p_i \) and \( q_j \), respectively, and directed edges \( p_{ij} \) and \( q_{ij} \), the discrete ARAP energy is defined as

\[
E(P, Q) = \sum_{i=1}^{m} \sum_{(i,j) \in \epsilon_i} c_{ij} \| q_{ij} - R_i p_{ij} \|^2
\]

(32)

where \( R_1, \ldots, R_m \in SO(3) \) are optimal local rotations, \( \epsilon_1, \ldots, \epsilon_m \) are their corresponding set of 1-ring edges and \( c_{ij} \) are weighting coefficients, typically the familiar cotangent weights.

The main idea of this method is breaking the surface into overlapping cells \( \epsilon_i \) and seek for keeping the cells transformations as rigid as possible in the least squares sense. Overlap of the cells is necessary to avoid surface stretching or shearing at the boundary of the cells. The vertices of mesh \( P \) are in original position while vertices of mesh \( Q \) are the deformed vertices and the matrix \( R_i \) is the best rigid transformation, in the least squares sense, relating the original and the deformed vertices. This is a non-linear optimisation problem that can be efficiently solved by a simple iterative method that solves two linear least squares sub-problems on each iteration.
\[
H' = j^\dagger j = D^j D^\dagger + S^j S^\dagger
\]

The first step is to consider the vertices of \( P \) and \( Q \) constant and obtain the best rigid transformation \( R_i \) for each cell \( \xi_i \). The second step is to consider the rotations \( R_i \) constant and computing the optimal deformed vertices \( q_i \) in the least squares sense. This can be achieved by taking the gradient of \( E(P, Q) \) w.r.t. \( Q \) and equating the result to zero, which leads to formulate a linear system of equations which can be solved by constraining the position of at least two vertices of \( Q \). The main source of inefficiency in this method is that the first step typically involves solving a series of singular value decomposition (SVD) problems, which is slow even using the optimised solver for 3 × 3 matrices of McAdams et al. We show how a GA approach can speed up the technique. We first change the energy \( E(P, Q) \) for using rotors instead of rotation matrices

\[
E(P, Q) = \sum_{i=1}^{n} \sum_{j \in \xi_i} c_{ij} \| q_{ij} - R_i p_j \tilde{R}_i \|^2 \quad \text{s.t.} \quad R_i \tilde{R}_i = 1, \forall i \tag{33}
\]

We first solve the best rotors incrementally first by applying the algorithm developed above. That is an incremental version of our algorithm that only require to solve a linear system for computing the next best rotor.

The second step is computing the optimal vertices \( q_i \). Taking the partial derivatives of \( E(P, Q) \) w.r.t. \( q_i \) and equating the result to zero, we obtain

\[
\sum_{(i,j) \in \xi_i} c_{ij} \partial q_{ij} = \sum_{(i,j) \in \xi_i} \bar{c}_{ij} (R_i p_j \tilde{R}_i + R_i p_j \tilde{R}_i) \tag{34}
\]

Equation (34) can be expressed in matrix form as \( LQ = C \), where \( L \) is the symmetric Laplace-Beltrami operator, \( Q \) is the column of target positions and \( C \) is the right hand side of (34). That is, a Poisson equation. Constraints of the form \( q_i = c_i \) are incorporated into the system by substituting the corresponding variables, i.e. erasing respective rows and columns from \( L \) and updating the right-hand side with the values \( c_i \). The system is then solved in the least squares sense

\[
(L^\dagger L)Q = L^\dagger C \tag{35}
\]

Mesh deformations are achieved by repositioning the constrained vertices \( q_i = c_i \), solving the linear subproblem for rotors \( R_i \) updating the right-hand-side of (34) and solving the LLS system for \( q_i \). Having a good initial guess, the convergence is typically achieved in less than ten iterations (three to four iterations already provides compelling results).

The performance improvements over the ARAP surface modelling technique are significant as can be seen in the comparison. The Euclidean rotors are as efficient as quaternions (indeed they are quite the same). Rotors require less storage than rotation matrices (just four numbers) and operations such as scalar multiplication, composition of transformations and addition are more efficient than matrices.

### 3.2 Smooth rotation ARAP (SR-ARAP) energy

The SR-ARAP [15] energy for a smooth map between two 2-manifolds \( f: P \rightarrow Q \) is

\[
E_{SR}(f) = \int_P \min_{R \in SO(3)} \left( \| df - R \|_F + \alpha \hat{A} \| dR \|_F \right) \tag{36}
\]

The first term in the integral is the membrane energy, and the second term is a bending energy that penalises the difference between rotations. \( \alpha \hat{A} \) is a weighting scalar, where \( \hat{A} \) is the area of the whole surface. Normally, the differential of a mapping of a 2-manifold is a 2 × 2 matrix, which maps tangent vectors from the parametric domain to a tangent plane at a surface point. Here, \( df \) is a 3 × 3 matrix, which maps the 3D embedding of the tangent vectors from one surface to another (which is simply the Jacobian matrix of \( f: \mathbb{R}^3 \rightarrow \mathbb{R}^3 \)). The discretisation is

\[
E_{SR}(P, Q) = \min_{R_1, \ldots, R_n \in SO(3)} \left( \sum_{(i,j) \in \xi_i} w_{ij} \| R_i - R_j \|_F^2 \right) + \alpha \hat{A} \sum_{(i,j) \in \xi_i} w_{ij} \| R_i \|_F^2 \tag{37}
\]

where \( R_1, \ldots, R_n \in SO(3) \) are optimal local rotations associated with the vertices; \( \xi_i \) is the set of 1-ring edges which are neighbours of \( i \)-th vertex; \( w_{ij} \) are scalar weights, and \( \hat{A} \) is the triangle mesh area, which is used to make the energy scale invariant. Here the regularisation is conducted in the form of the chordal distance \( \| R_i - R_j \|_F \) between \( R_i \) and \( R_j \). The first term, the membrane energy, is similar to the ARAP discretisation, and the second term, the bending energy, penalises the difference between an edge set rotation and the rotations of neighbouring edge sets. The objective of the membrane term is to lower the distortion of an edge set (resists stretching and shearing), by keeping the map differential close to rigid. The objective of the bending term is to keep the variation in the rotations in an edge set neighbourhood low, such that the neighbourhood would transform as a unit, as much as possible.

#### 3.3 Geometric-algebra SR-ARAP (GA SR-ARAP) energy

The GA SR-ARAP energy is shown as follows:

\[
E_{SR}(P, Q) = \min_{R_1, \ldots, R_n \in SP(3)} \left( \sum_{(i,j) \in \xi_i} w_{ij} \| q_{ij} - R_i p_j \tilde{R}_i \|^2 \right) + \alpha \hat{A} \sum_{(i,j) \in \xi_i} w_{ij} \| R_i \|_F^2 \tag{38}
\]

Notice that we are expressing rotations using rotors instead of 3 × 3 rotation matrices, also notice that in the second term, the bending energy, we are using the Euclidean \((L)\) norm instead of the Frobenius norm. Those changes are not affecting the resulting surface deformation in any noticeable way but are very convenient since the resulting optimisation problem can be solved with a much more efficient algorithm as we will show.

#### 3.4 Local step

In the local step we consider vertices of \( P \) and \( Q \) as constants and solve for the best rotation of each cell \( \xi_i \), independently by minimising...
Let $\tilde{R}_i$ be the vector of functions which square equals to the constraint $E_i = 2$, where the constraint $\tilde{R} = 1 \tilde{R}_i$ is now implicit, although we will project $\tilde{R}_i$ back to the rotor manifold through normalisation.

Following Perwass, we can make it linear by multiplying by $R_i$ to the rotor manifold through normalisation.

Notice that the first row of $H^t$ is equal to $g^t$, so $g^t$ does not need to be calculated. Now the right-hand-side does not depend on $R_i$ (see (49)). The final system to solve is

$$
\Delta R = -\left( \sum_{i=1}^{n} H^t + \alpha \tilde{A} \right)^{-1} \left( \sum_{i=1}^{n} (g^t + d^t) \right)
$$

3.5 Global step

The global step is computing the optimal vertices $\{q_i\} \in Q$

$$
E_{SB}(P, Q) = \min_{R_1, \ldots, R_n \in S^{3d}} \sum_{i=1}^{n} \sum_{(i, j) \in E_i} c_{ij} || q_i - R_pq_j ||^2
$$

Taking the partial derivatives of $E_{SB}(P, Q)$ w.r.t. $q_i$ and equating the result to zero lead us to obtain the linear system of (34)

$$
\sum_{(i, j) \in E_i} c_{ij} q_j = \sum_{(i, j) \in E_i} c_{ij} (R_{ij}pq_j + R_{ij}pq_{ij})
$$

which can be expressed in matrix form as $LQ = C$, where $L$ is the symmetric Laplace-Beltrami operator, $Q$ is the column of target positions and $C$ is the right hand side of (34). Constraints of the form $q_i = c_i$ are incorporated into the system by substituting the corresponding variables, i.e. erasing respective rows and columns from $L$ and updating the right-hand side with the values $c_i$. The solution is $(L^tL)Q = L^tC$, as described previously.

3.5.1 Local relaxation: In the SR-ARAP method, mesh deformations are obtaining by repositioning the constrained vertices $q_i = c_i$ solving local step subproblem for rotors $R_i$, updating the right-hand-side of (34) and solving the global step subproblem system for $q_i$. Unlike the ARAP surface, the optimised

$$
\begin{align*}
\arg \min_{R_i \in S^{3d}} & \sum_{(i, j) \in E_i} c_{ij} || R_pq_i - R_pq_j ||^2 \\
+ \alpha \tilde{A} & \sum_{(i, j) \in E_i} w_{ij} || R_i - R_j ||^2, \quad \text{s.t.} \quad R_i = 1
\end{align*}
$$

Like in the Wahba problem, in the first term the minimisation is non-linear in $R_i$. Following Perwass, we can make it linear by multiplying by $R_i$ on the right

$$
\begin{align*}
q_iR_i - R_pq_iR_i &= 0 \\
q_iR_i - R_pq_{ij} &= 0
\end{align*}
$$

This amounts to minimise

$$
\begin{align*}
E(R_i) &= \sum_{(i, j) \in E_i} c_{ij} || q_iR_i - R_pq_{ij} ||^2 \\
+ \alpha \tilde{A} & \sum_{(i, j) \in E_i} w_{ij} || R_i - R_j ||^2
\end{align*}
$$

where the constraint $\tilde{R}_i = 1$ is now implicit, although we will need to project $\tilde{R}_i$ back to the rotor manifold through normalisation.

Let $F(R_i)$ be the vector of functions which square equals to the energy $E(R_i) = F^t F$. The column vector $F$ can be split in two blocks so that

$$
F = [(F_1^t, F_2^t, \ldots, F_n^t)^T]
$$

where $n$ is the degree of the $i$th vertex, i.e. $n = |E_i|$. The energy $E(R_i)$ can be expressed as the matrix product

$$
E(R_i) = F^t F
$$

where $F(R_i) = \sqrt{c_{ij}(R_pq_{ij} - q_iR_i)}F_i(R_i) = \sqrt{w_{ij}}(R_i - R_j)$ in which the factor $w_{ij}$ is absorbing the whole factor $(\alpha \tilde{A} n)w_{ij}$ for the sake of notation simplicity. $F_i$ is producing a multivector with the basis $\{e_1, e_2, e_3, e_{12}, e_{13}, e_{23}\}$ and $F_i$ is producing a rotor in the basis $\{1, e_2, e_3, e_{12} \}$. The critical points of $E(R_i)$ are solutions of the equation $\nabla E(R_i) = 0$

$$
\nabla E(R_i) = \frac{\partial F^t F}{\partial R_i} = 2 \sum_{j=1}^{n} J_i^T F_i^j + 2 \sum_{j=1}^{n} J_i^T F_i^j
$$

Since $J_i$ and $J_j$ are constant, the equation $\nabla E(R_i) = 0$ is linear in $R_i$ and the solution amounts to solve the null-space of a linear system. As before, we can solve it using only one iteration of Newton method. The Newton increment is given by $\Delta R = -H^{-1} \nabla E(R_i)$. The Hessian matrix $H$ is

$$
H = 2 \left( \sum_{j=1}^{n} J_i^T + \sum_{j=1}^{n} J_i^T J_i^T \right)
$$

So, the full system looks like this

$$
\left( \sum_{j=1}^{n} J_i^T J_i^j + \sum_{j=1}^{n} J_i^T F_i^j \right) \Delta R = - \sum_{j=1}^{n} J_i^T F_i^j - \sum_{j=1}^{n} J_i^T F_i^j
$$

For some initial guess $R_0$, that we will choose to be $R_0 = 1$ as in the previous section. The rest of the work focuses on the computation of Jacobian and Hessian which is in the similar form shown in aforementioned contents. Here we define

$$
g^t = J_i^T F_i^j = \begin{pmatrix} D_i^2 + D_j^2 + D_{i j}^2 \\
D_{i j} - D_i S_i - D_j S_j \\
S_i S_j - D_i D_j \\
S_i S_j - D_i D_j \\
S_i S_j - D_i D_j \\
S_i S_j - D_i D_j
\end{pmatrix}
$$

$$
d^t = J_i^T F_i^j = \begin{pmatrix} 1 - \{R_j\}_b \\
- R_j \cdot e_2 \\
- R_j \cdot e_3 \\
- R_j \cdot e_3
\end{pmatrix}
$$
rotations in the local step for SR-ARAP are codependent. Since we optimise each rotation independently, while fixing the others, the local step can be considered as a relaxation, thus more than one local iteration should be executed before performing the global step. At least two steps of local relaxations should be done for each global iteration, although for better convergence we should perform more local relaxations for each global iteration.

3.6 Real-time SR-ARAP

The local relaxation used to be a serious bottleneck for the SR-ARAP method. However our fast rotor estimation SR-ARAP can be used to compute as many local relaxation steps as needed at the same cost as the simple ARAP method (i.e. without slowing down the solver). Also we show how real-time performance can be achieved on SR-ARAP by adding a simple multiresolution step.

3.6.1 Rotor relaxation: Recall that rotor estimation for SR-ARAP amounts to solve one linear system

$$R' = R_0 - H^{-1}(g + d)$$

(53)

where $H = \sum_{j=1}^{g} \sum_{i=1}^{d} |H_{ij} + \alpha \Delta_I|$, $g = \sum_{j=1}^{g} |d|$ and $d = \sum_{j=1}^{d} |d|$. The only term involving the neighbour rotations is $\sum_{j=1}^{g} |d|$. So, for the entire local relaxation we can precompute $H^{-1}$ and $g$. So, the computational cost of solving each relaxation step amounts to do a matrix multiplication.

3.6.2 Multiresolution SR-ARAP: For achieving real-time performance, we optimise the SR-ARAP energy in a low resolution version of the input mesh and then we transfer that solution to the full resolution mesh. To obtain the low resolution mesh, we simplify the mesh using half edge collapses (i.e. the simplified mesh is a triangulation of subset of the original vertices) while minimising the Quadrics error metric. After we obtained the optimal deformed shape on the simplified mesh, we transfer the optimised rotations to the full resolution mesh and solve the linear system of (34) using as position constraints the optimised vertices of the low resolution mesh.

The transference of rotations from the low-res mesh to the high-res mesh is equivalent to clustering rotations in the high resolution mesh. The clustering of rotations is based on the connectivity graph of the simplified mesh. Having the a injective map $f: Q_{low} \rightarrow Q_0$ that maps rotations from low-res mesh $Q_{low}$ to hi-res mesh $Q_0$ we generate the surjective map $c: Q_0 \rightarrow Q_{low}$ which is mapping several (equal) rotations of $Q_0$ to one corresponding rotation of $Q_{low}$. Our algorithm to construct the surjection $c$ is a simple diffusion process: We start by mapping the rotations from low-res mesh to hi-res mesh using the injection $f$, then we iteratively copy the assigned rotations to the neighbouring vertices that still does not have a rotation, repeating the process until there are no vertices with neighbours without a rotation assigned. That process is used to precompute the surjection $c: Q_0 \rightarrow Q_{low}$ which is used later for efficient transferring of rotations. Let $f^{-1}$ denote the inverse image of the injection $f$. So, the range of $f^{-1}$ is the entire set of vertices in $Q_{low}$ and the domain of $f^{-1}$ is the image of $f$.

4 Experimental results

In this section, we use some datasets to conduct registration and mesh deformation using our proposed method and other existing algorithms. In the first study, we show the accuracy consistency of the proposed method with representative methods. We use the following model to generate the point cloud data:

$$\mathcal{L} = \sum_{i=1}^{n} \| b_i - R_{i} \|^2$$

(54)

in which $b_i$ is the reference vector and $h_i$ stand for the measurement vector in the body frame. We use the ground truth data of $R$ and reference data as shown in Table III in [10]. Here various algorithms for registration are employed for assembled test, they are SVD method by Horn [16], Fast Linear Attitude Estimator (FLAE) [10], GDA [13], Fast Symbolic 3D Registration (FS3R) [11], Fast Analytical 3D Registration (FA3R) [17]. Each of them has been proven to be accurate, robust and computationally efficient.

We evaluate various representatives using the loss function value $\mathcal{L}$ and computation time on a personal computer of MacBook Pro 2017 with i7-CPU and MATLAB environment. The required parameters of these algorithms are:

- Proposed: relative stop criteria $1 \times 10^{-12}$.
- SVD: no parameter.
- FLAE: no parameter.
- GDA: descent step length $1 \times 10^{-3}$; relative stop criteria $1 \times 10^{-5}$; maximum iteration number 100.
- FS3R: singularity check factor: $1 \times 10^{-3}$.
- FA3R: relative stop criteria $1 \times 10^{-12}$.

With 12 classical cases in [10], we are able to generate the results shown in Table 1. One may observe that the accuracy of compared methods are almost identical. The reason is that these solutions are all based on the framework of (54) so the resulted analytical formulations achieve equivalent mathematical errors.

The developed method has been combined with ARAP and its variants in the aforementioned contents. Based on the energy functions defined using the GA, we are able to implement the proposed algorithm. Here, the libigl-2.1.0 library [18, 19] has been invoked for high-efficiency implementation of the mesh deformation via parallel computation through the BLASX library [20]. We use the decimated-knight and cactus models to

| Case | Proposed | SVD Horn 1987 [16] | FLAE 2018 [10] | GDA 2018 [13] | FS3R 2019 [11] | FA3R 2020 [17] |
|------|----------|-------------------|----------------|-------------|---------------|----------------|
| 1    | $4.9709 \times 10^{-11}$ | $4.9709 \times 10^{-11}$ | $4.9709 \times 10^{-11}$ | $4.9709 \times 10^{-11}$ | $4.9709 \times 10^{-11}$ | $4.9709 \times 10^{-11}$ |
| 2    | $2.4983 \times 10^{-11}$ | $2.4983 \times 10^{-11}$ | $2.4983 \times 10^{-11}$ | $2.4983 \times 10^{-11}$ | $2.4983 \times 10^{-11}$ | $2.4983 \times 10^{-11}$ |
| 3    | $5.0554 \times 10^{-10}$ | $5.0554 \times 10^{-10}$ | $5.0554 \times 10^{-10}$ | $5.0554 \times 10^{-10}$ | $5.0554 \times 10^{-10}$ | $5.0554 \times 10^{-10}$ |
| 4    | $2.4957 \times 10^{-10}$ | $2.4957 \times 10^{-10}$ | $2.4957 \times 10^{-10}$ | $2.4957 \times 10^{-10}$ | $2.4957 \times 10^{-10}$ | $2.4957 \times 10^{-10}$ |
| 5    | $5.0278 \times 10^{-11}$ | $5.0278 \times 10^{-11}$ | $7.1999 \times 10^{-10}$ | $5.0278 \times 10^{-11}$ | $5.0444 \times 10^{-11}$ | $7.1999 \times 10^{-10}$ |
| 6    | $4.9635 \times 10^{-13}$ | $4.9635 \times 10^{-13}$ | $4.9635 \times 10^{-13}$ | $4.9635 \times 10^{-13}$ | $4.9635 \times 10^{-13}$ | $4.9635 \times 10^{-13}$ |
| 7    | $2.4841 \times 10^{-12}$ | $2.4841 \times 10^{-12}$ | $2.4841 \times 10^{-12}$ | $2.4841 \times 10^{-12}$ | $2.4841 \times 10^{-12}$ | $2.4841 \times 10^{-12}$ |
| 8    | $4.7623 \times 10^{-11}$ | $4.7623 \times 10^{-11}$ | $4.7623 \times 10^{-11}$ | $4.7623 \times 10^{-11}$ | $4.7623 \times 10^{-11}$ | $4.7623 \times 10^{-11}$ |
| 9    | $2.5273 \times 10^{-11}$ | $2.5273 \times 10^{-11}$ | $2.5273 \times 10^{-11}$ | $2.5273 \times 10^{-11}$ | $2.5273 \times 10^{-11}$ | $2.5273 \times 10^{-11}$ |
| 10   | $1.5007 \times 10^{-12}$ | $1.5007 \times 10^{-12}$ | $1.5007 \times 10^{-12}$ | $1.5007 \times 10^{-12}$ | $1.5007 \times 10^{-12}$ | $1.5007 \times 10^{-12}$ |
| 11   | $4.9458 \times 10^{-13}$ | $4.9458 \times 10^{-13}$ | $9.5140 \times 10^{-13}$ | $4.9458 \times 10^{-13}$ | $4.9471 \times 10^{-13}$ | $9.5140 \times 10^{-13}$ |
| 12   | $5.0559 \times 10^{-13}$ | $5.0559 \times 10^{-13}$ | $3.4190 \times 10^{-11}$ | $5.0559 \times 10^{-13}$ | $7.5501 \times 10^{-11}$ | $3.4190 \times 10^{-11}$ |

IET Cyber-syst. Robot., 2020, Vol. 2 Iss. 2, pp. 88-95

This is an open access article published by the IET and Zhejiang University Press under the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/).
evaluate the various ARAP variants under the kernel energy of the proposed geometric-algebra solution.

The model of knight contains 1500 vertices and simulated control points are generated using ground truth rotation. Fig. 1 shows the refined ARAP results of various algorithms. Gradient descent is introduced for classical implementation where the descent factor is set to $5 \times 10^{-2}$. Seen from the figure, it is observed that the GA supervised ARAP can achieve quite accurate deformation after 20 iterations. However, the GDA according to its nature of convergence problem, cannot fully converge to the ground truth value. That is to say in engineering, we may need more iterations or improved Newton methods to converge to a satisfactory result. However, we need to note that this is not computationally efficient and thus should be compared with the proposed GA algorithm.

The cactus model is employed for mesh deformation and the result of GA SR-ARAP is shown in Fig. 2. We can see that under high noise level the algorithm is still capable of achieving good deformation results. This indicates the accuracy of the GA-based point cloud registration. Now we need to compare the computational efficiency of various algorithms. All these algorithms are then implemented in an unoptimised (compilation level) manner guaranteeing the fairness of all the run-time results. The termination criteria of all the algorithms is that the relative difference of the minimised energy of two successive iterations reaches $1 \times 10^{-8}$. The details of the execution time for the cactus model with different numbers of vertices are summarised in Table 2. The GA-based ARAP and variants are more time-efficient than other representatives. We need to point out that although FLAE is analytical and computationally more efficient than the iterative GA algorithm proposed in this paper, it is not so friendly

![Fig. 1 Knight deformation: left: reference model of the knight; middle: deformation using the proposed method combining with ARAP; right: deformation using the GDA [13] combining with ARAP](image1)

![Fig. 2 GA SR-ARAP clusters: left: Hi-res cactus model with Low-res points marked. Clusters formed around vertices are coloured with same intensity. Middle and right: deformation of Low-res model transferred to Hi-res model](image2)

| Case | ARAP | SR-ARAP | GA-ARAP | GSR-ARAP | GAARAP | FLAEARAP |
|------|------|---------|---------|----------|--------|----------|
| 100  | $1.234960 \times 10^{-2}$ s | $1.369739 \times 10^{-2}$ s | $7.786490 \times 10^{-3}$ s | $2.870024 \times 10^{-2}$ s | $8.915073 \times 10^{-3}$ s | $8.915073 \times 10^{-3}$ s |
| 199  | $3.714611 \times 10^{-2}$ s | $9.279175 \times 10^{-3}$ s | $1.155189 \times 10^{-2}$ s | $1.613069 \times 10^{-2}$ s | $1.144537 \times 10^{-3}$ s | $1.144537 \times 10^{-3}$ s |
| 398  | $5.28854 \times 10^{-3}$ s | $1.070500 \times 10^{-2}$ s | $2.611447 \times 10^{-2}$ s | $3.654026 \times 10^{-2}$ s | $1.647663 \times 10^{-3}$ s | $1.647663 \times 10^{-3}$ s |
| 794  | $1.889311 \times 10^{-3}$ s | $4.307448 \times 10^{-3}$ s | $5.891055 \times 10^{-3}$ s | $1.047712 \times 10^{-2}$ s | $6.138786 \times 10^{-3}$ s | $6.138786 \times 10^{-3}$ s |
| 1584 | $1.846733 \times 10^{-3}$ s | $3.815289 \times 10^{-3}$ s | $6.808507 \times 10^{-3}$ s | $8.913603 \times 10^{-3}$ s | $5.572538 \times 10^{-3}$ s | $5.572538 \times 10^{-3}$ s |
| 3162 | $3.78262 \times 10^{-3}$ s | $7.035758 \times 10^{-3}$ s | $1.392477 \times 10^{-2}$ s | $1.800131 \times 10^{-2}$ s | $1.169642 \times 10^{-1}$ s | $1.169642 \times 10^{-1}$ s |
| 6309 | $5.08200 \times 10^{-3}$ s | $8.757848 \times 10^{-3}$ s | $1.776170 \times 10^{-2}$ s | $2.162810 \times 10^{-2}$ s | $1.457190 \times 10^{-1}$ s | $1.457190 \times 10^{-1}$ s |

Bold values indicate results from our method.
combining with ARAP. Therefore, the convergence of the FLAE-based ARAP will require more computational burden.

For illustration of the proposed method for large numbers of meshes, the Armadillo model has been chosen which contains over 43,000 vertices and 80,000 edges [21]. For ARAP, it is hard to implement real-time animation of the Armadillo model for satisfactory update frequency. Using the proposed GA-ARAP, it is able to speed up the mesh deformation to a large extent. Fig. 3 shows the details of the reference and the deformation result from the proposed GA-ARAP.

The developed algorithm maintains good accuracy and can achieve very high visualization frequency of 60 fps on a typical personal computer with i7-4core CPU. For conventional ARAP, since the computational burden is relatively higher, it can only reach the update rate of 12 fps at most. The good computational behaviour of the proposed method provides the possibility of performing highly real-time mesh deformation such as games and high-speed visualisation.

5 Conclusion

In this paper, we solve the problem of estimating the best rotation for the alignment of two sets of corresponding 3D points. It is based on solving the linear equations derived from the formulation of the problem in Euclidean GA. The method is fast, robust to noise, accurate, simple and GPU-friendly. Applications and experimental validation of mesh deformation are presented for description of the performance of the proposed algorithm. The results show that the slightly losing accuracy of the proposed method leads to advance in execution time consumption. The designed algorithms are ARAP-friendly and may be helpful for real-time mesh deformation in related applications.

6 Acknowledgment

J. Wu and M. Lopez contributed equally to the content of this paper.

7 References

[1] Yi, R., Wu, C., Liu, Y.J., et al.: ‘Delta DLP 3-D printing of large models’, IEEE Trans. Autom. Sci. Eng., 2018, 15, (3), pp. 1193–1204
[2] Zhang, Y., Wang, C.C.L.: ‘Wirewrapping+: robust and flexible surface flattening with length control’, IEEE Trans. Autom. Sci. Eng., 2011, 8, (1), pp. 205–215
[3] Sorkine, O., Alexa, M.: ‘As-rigid-as-possible surface modelling’, Proc. of the Fifth Eurographics Symp. on Geometry Processing. SGP ’07, Aire-la-Ville, Switzerland, 2007, pp.109–116
[4] Anan, K.S., Huang, T.S., Blountin, S.D.: ‘Least-squares fitting of two 3-D point sets’, IEEE Trans. Pattern Anal. Mach. Intell., 1987, PAMI-9, (5), pp. 698–700
[5] Wahba, G.: ‘A least squares estimate of satellite attitude’, SIAM Rev., 1965, 7, (3), pp. 409–409
[6] Best, P.J., McKay, N.D.: ‘A method for registration of 3-D shapes’, IEEE Trans. Pattern Anal. Mach. Intell., 1992, 14, (2), pp. 239–256
[7] Shuster, M.D., Oh, S.D.: ‘Three-axis attitude determination from vector observations’, J. Guid. Control Dyn., 1981, 4, (1), pp. 70–77
[8] Markley, F.L.: ‘Attitude determination using vector observations and the singular value decomposition’, J. Astronaut. Sci., 1980, 36, (3), pp. 245–258
[9] Mortari, D., Markley, F.L., Singla, P.: ‘Optimal linear attitude estimator’, J. Guid. Control Dyn., 2007, 30, (6), pp. 1619–1627
[10] Wu, J., Zhou, Z., Gao, B., et al.: ‘Fast linear quaternion attitude estimator using vector observations’, IEEE Trans. Autom. Sci. Eng., 2018, 15, (1), pp. 307–319
[11] Wu, J., Liu, M., Zhou, Z., et al.: ‘Fast symbolic 3D registration solution’, IEEE Trans. Autom. Sci. Eng., 2018, PP, pp. 1–10
[12] Ying, S., Peng, J., Du, S., et al.: ‘A scale stretch method based on ICP for 3D data registration’, IEEE Trans. Autom. Sci. Eng., 2009, 6, (3), pp. 559–565
[13] Wu, J., Zhou, Z., Fourati, H., et al.: ‘Generalized linear quaternion complementary filter for attitude estimation from multisensor observations: an optimization approach’, IEEE Trans. Autom. Sci. Eng., 2019, 16, (3), pp. 1330–1343
[14] Wu, J., Zhou, Z., Fourati, H., et al.: ‘Recursive linear continuous quaternion attitude estimator from vector observations’, IET Radar Sonar Navig., 2018, 12, (11), pp. 1196–1207
[15] Levi, Z., Gottman, C.: ‘Smooth rotation enhanced as-rigid-as-possible mesh animation’, IEEE Trans. Visual. Comput. Graph., 2014, 21, (2), pp. 264–277
[16] Horn, B.K.P.: ‘Closed-form solution of absolute orientation using unit quaternions’, J. Opt. Soc. Am. A, 1987, 4, (4), pp. 629–642
[17] Wu, J.: ‘Rigid 3D registration: a simple free of SVD and Eigen-decomposition’, IEEE Trans. Instrum. Meas., 2020, pp. 1–1
[18] Jacobson, A., Panozzo, D.: ‘Libigl: prototyping geometry processing research in C++’, SIGGRAPH Asia 2017 Courses, Bangkok, Thailand, 2017, pp. 1–172
[19] Jacobson, A., Panozzo, D., Schuller, C., et al.: ‘Libigl: a simple C++ geometry processing library’, 2016. Retrieved from http://libigl.github.io/libigl, 18 October 2017
[20] Wang, L., Wu, W., Xu, Z., et al.: ‘BLASX: a high performance level-3 BLAS library for heterogeneous multi-GPU computing’. Proc. of the 2016 Int. Conf. on Supercomputing, Istanbul, Turkey, 2016, pp. 1–11
[21] Lavoué, G.: ‘A roughness measure for 3D mesh visual masking’, ACM Int. Conf. Proceeding Set., Freiburg, Germany, 2007, vol. 253, pp. 57–60