A High-Performance Orientation Estimator in Geometric Algebra

Mauricio Cele Lopez Belon

Abstract. Modern microprocessors supporting Advanced Vector Extensions (AVX) can double the speed of calculations on floating point data. By leveraging the available 4×4 matrix multiplication routines using AVX registers, we developed a high-performance estimator of the best rotation aligning two sets of corresponding vectors (also known as Wahba's problem). On modern micro-processors, the proposed method outperform the fastest methods reported in literature and on old microprocessors it is still among the fastest methods, moreover it is robust to noise, accurate and simpler than most other methods. Our solution is based on Geometric Algebra representation of rotations which is isomorphic to Hamilton's quaternions.

Mathematics Subject Classification (2010). Parallel algorithms 68W10; Clifford algebras, spinors 15A66.

Keywords. Geometric Algebra, Rotation Estimation, Wahba Problem.

1. Introduction

Wahba's problem has been studied for over half a century since 1965 [12]. The problem looks for the optimal rotation between two sets of corresponding vectors. Many effective algorithms have been developed to solve the problem [1, 4, 6, 9, 14, 15] using quaternions, the Singular Value Decomposition (SVD) and recently geometric algebra representation [3, 8]. Applications of Wahba's solutions are diverse from aerospace engineering computation of spacecraft attitude [14] to mesh deformation in computer graphics [10, 11] for accurate motion construction and restoration [7, 8].

In engineering fields requiring high performance vector alignment our method can lowered down the time in more than 50% by leveraging the Advanced Vector Extensions (AVX) support available in modern microprocessors. In this work we propose a fast, robust to noise, simple and accurate method for solving the Wahba's problem. It is based on minimizing a convex least squares error formulated in geometric algebra $\mathbb{G}_3$. The proposed method

allow us to exploit the available AVX instructions to accelerate the 4×4 matrix multiplication of symmetric matrix. Obtaining a performance boost not present in other numerical algorithms when execute on modern hardware.

Few work has been done on studying this problem using geometric algebra. Geometric algebra rotors are closely related to quaternions (quaternion algebra can be regarded as a geometric algebra defined on a set of imaginary basis vectors) we find geometric algebra to be a more natural choice for studying this problem since it is defined over a Euclidean vector space $\mathbb{R}^3$, where original data is defined. We derive our algorithm from Wahba's problem defined on bivectos instead of vectors for the sake of mathematical simplicity. Due to mathematical duality of vector and bivectors in $\mathbb{G}_3$ our formulation is also valid for vectors. The implementation of our algorithm does not require access to any geometric algebra library, we present an implementation based on standard matrix and quaternion library.

This paper is arranged as follows: Section II introduces the geometric algebra $\mathbb{G}_3$, Section III includes the presentation of our fast rotor estimation algorithm. Section IV demonstrates the experimental results.

2. Geometric Algebra $\mathbb{G}_3$

A geometric algebra $\mathbb{G}_3$ is constructed over a real vector space $\mathbb{R}^3$, with basis vectors $\{e_1, e_2, e_3\}$. The associative geometric product is defined so that the square of any vector is a scalar $aa = a^2 \in \mathbb{R}$. From the vector space $\mathbb{R}^3$, the geometric product generates the geometric algebra $\mathbb{G}_3$ with elements $\{X, R, A...\}$ called multivectors.

For a pair of vectors, a symmetric inner product $a \cdot b = b \cdot a$ and antisymmetric outer product $a \wedge b = -b \wedge a$ can be defined implicitly by the geometric product $ab = a \cdot b + a \wedge b$ and $ba = b \cdot a + b \wedge a$. It is easy to prove that $a \cdot b = \frac{1}{2}(ab + ba)$ is scalar, while the quantity $a \wedge b = \frac{1}{2}(ab - ba)$, called a bivector or 2-vector, is a new algebraic entity that can be visualized as the two-dimensional analogue of a direction, that is, a planar direction. Similar to vectors, bivectors can be decomposed in a bivector basis $\{e_{12}, e_{13}, e_{23}\}$ where $e_{ij} = e_i \wedge e_j$.

The outer product of three vectors $a \wedge b \wedge c$ generates a 3-vector also known as the pseudoscalar, because the trivector basis consist of single element $e_{123} = e_1 \wedge e_2 \wedge e_3$. Similarly, the scalars are regarded as 0-vectors whose basis is the number 1. It follows that the outer product of k-vectors is the completely antisymmetric part of their geometric product: $a_1 \wedge a_2 \wedge ... \wedge a_k = \langle a_1 a_2 ... a_k \rangle_k$ where the angle bracket means k-vector part, and k is its grade. The term grade is used to refer to the number of vectors in any exterior product. This product vanishes if and only if the vectors are linearly dependent. Consequently, the maximal grade for nonzero k-vectors is 3. It follows that every multivector X can be expanded into its k-vector parts and the entire

algebra can be decomposed into k-vector subspaces:

$$\mathbb{G}_3 = \sum_{k=0}^{n} \mathbb{G}_3^k = \{X = \sum_{k=0}^{n} \langle X \rangle_k\}$$

This is called a *grading* of the algebra.

Reversing the order of multiplication is called reversion, as expressed by $(a_1 a_2 ... a_k)\tilde{} = a_k ... a_2 a_1$ and $(a_1 \wedge a_2 \wedge ... \wedge a_k)\tilde{} = a_k \wedge ... \wedge a_2 \wedge a_1$, and the reverse of an arbitrary multivector is defined by $\tilde{X} = \sum_{k=0}^{n} \langle \tilde{X} \rangle_k$.

Rotations are even grade multivectors known as rotors. We denote the subalgebra of rotors as $\mathbb{G}_3^+$. A rotor R can be generated as the geometric product of an even number of vectors. A reflection of any k-vector X in a plane with normal n is expressed as the sandwitch product $(-1)^k nXn$. The most basic rotor R is defined as the product of two unit vectors a and b with angle of $\frac{\theta}{2}$. The rotation plane is the bivector $B = \frac{a \wedge b}{\|a \wedge b\|}$.

$$ab = a \cdot b + a \wedge b = \cos\left(\frac{\theta}{2}\right) + B \sin\left(\frac{\theta}{2}\right).$$

Rotors act on all k-vectors using the sandwitch product $X' = RX\tilde{R}$, where $\tilde{R}$ is the reverse of R and can be obtained by reversing the order of all the products of vectors.

3. Geometric Algebra Rotor Estimation

Given two sets of n corresponding bivectors $P = \{p_j\}_{j=1}^n$ and $Q = \{q_j\}_{j=1}^n$, we attempt to minimize the following error function:

$$E(R) = \min_{R \in \mathbb{G}_3^+} \sum_j c_j \|q_j - R p_i \tilde{R}\|^2$$

$$s.t.\ R\tilde{R} = 1$$

where $\{c_j\}_{j=1}^n$ are scalar weights such that $\sum_j^n c_j = 1$. In that form, the minimization is a non-linear least squares problem with a non-linear constraint in R. Notice that the error term $q_j - R p_i \tilde{R}$ is equivalent to $q_j R - R p_j$ by multiplying by R on the right and using the fact that $R\tilde{R} = 1$. The equivalent problem is:

$$E(R) = \min_{R \in \mathbb{G}_3^+} \sum_j c_j \|R p_j - q_j R\|^2$$

$$s.t.\ R\tilde{R} = 1$$

which is a linear least squares problem with a non-linear constraint in R.

We define the commutator product of two bivectors p_j and q_j as $p_j \times q_j = \frac{1}{2}(p_j q_j - q_j p_j)$. The commutator product of bivectors in $\mathbb{G}_3$ can be interpreted as a cross product of bivectors in the sense that the resulting bivector $B = p_j \times q_j$ is orthogonal to both p_j and q_j.

For a pair of unit bivectors, A and B, the rotor R aligning them can be defined as $R = A\frac{(A+B)}{\|A+B\|}$. Define the unit bivector $C = \frac{A+B}{\|A+B\|}$ the rotor

R is also $R = AC = w + L$ where $w = A \cdot C = \cos(\theta/2)$ and $L = A \times C = \sin(\theta/2)\frac{A \times B}{\|A \times B\|}$.

The constraint $Rp_j - q_j R = 0$ can be rewrite as $(w+L)p_j - q_j(w+L) = (w + L) \cdot (p_j - q_j) + L \times (p_j + q_j)$. Since the inner product $L \cdot (p_j - q_j) = 0$, the constraint can be seen as:

$$\begin{aligned} w(p_j - q_j) + L \times (p_j + q_j) &= 0 \\ L \times (p_j + q_j) &= -w(p_j - q_j) \end{aligned}$$

The above constraint equation holds when the input data is not noisy, since all terms $(p_j - q_j) \cdot L$ cancels out. However in a realistic case the input data would have some level of noise so it is better to keep the term $(p_j - q_j) \cdot L$ in the equation and let the optimization find its minimum possible value:

$$(p_j - q_j) \cdot (w + L) + L \times (p_j + q_j) = 0$$

In matrix language we can define the following matrix system $M_j R = 0$:

$$\begin{bmatrix} 0 & -d^T \\ d & [s]_\times^T \end{bmatrix} \begin{bmatrix} w \\ L \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$d = p_j - q_j \quad s = p_j + q_j$$

where M_j is skew-symmetric 4×4 real matrix, so that $M_j^T = -M_j$. The rotor R is represented as 4×1 column vector with the real scalar w and a vector $L \in \mathbb{R}^3$ holding the bivector components. The 3×3 skew-symmetric matrix $[p_j + q_j]_\times$ is representing the cross-product matrix as defined for vectors in $\mathbb{R}^3$.

Let define the function F^j representing the linear transformation:

$$F^j = \sqrt{c_j} M_j R$$

Let F be a column vector of n stacked functions F^j

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

such that the energy $E(R)$ can be expressed as matrix product

$$E(R) = F^T F = \begin{bmatrix} F^{1T} & \cdots & F^{nT} \end{bmatrix} \begin{bmatrix} F^1 \\ \vdots \\ F^n \end{bmatrix}$$

We can express $E(R)$ as the following quadratic form:

$$E(R) = \min_R R^T M R$$
$$s.t.\ R^T R = 1$$

where $M = \sum_j^n c_j M_j^T M_j$. Note that since M_j is skew-symmetric, the product $M_j^T M_j$ is symmetric and positive semi-definite. Consequently the matrix M is also symmetric positive semi-definite. It follows that all eigenvalues of M are real and $\lambda_i \geq 0$.

$$M_j^T M_j = \begin{bmatrix} \|d\|^2 & (s \times d)^T \\ s \times d & dd^T - [s]_\times^2 \end{bmatrix}$$
$$d = p_j - q_j \quad s = p_j + q_j$$

Using the spectral theorem it is easy to show that the minimizer of $E(R)$ is the eigenvector of M associated with the minimum eigenvalue. Typically it can be obtained with great precision using SVD or QR factorization. Nevertheless, in this work we seek for an efficient solution which allow us to take advantage of nowadays computing hardware.

The first row of M_j is encoding the product $(p_j - q_j) \cdot L$ which is zero when the rotor R aligns perfectly the input data. When that happen, R is in the null space of M_j, the matrix $M_j^T M_j$ is singular as at least one eigenvalue is zero $\lambda_{\min} = 0$ (two eigenvalues will be zero for a single pair of corresponding bivectors). That implies that 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 $R_i = e^{-(\theta + i\pi)L}$ for $i \in \mathbb{N}$. That means that the solution is in general non-unique.

4. Fast Geometric Algebra Rotor Estimation

The most time consuming task of the estimation is to compute the eigenvector of M associated with the minimum eigenvalue. Sound numerical algorithms for computing eigenvectors of a real symmetric matrix are SVD, QR factorization and Jacobi iteration among others. These algorithms find all eigenvectors of a matrix. Since we are interested in finding an eigenvector with least eigenvalue, our best choice is to use the Inverse Power Method. Which have the following form:

$$R_{i+1} = \epsilon(M + \epsilon I)^{-1} R_i$$

Where ϵ is a small real value $0 < \epsilon < 1$. The use of Tikhonov regularization makes M symmetric positive-definite matrix, making the energy $E(R)$ convex. Note that the Hessian matrix of $E^j = R^T M_j^T M_j R$ is $H^j = 2M_j^T M_j$, so the Hessian matrix $H = \sum_j M_j^T M_j$ of $E(R)$ is equal to the matrix M. The Tikhonov regularization makes the Hessian matrix positive-definite.

The Newton update formula shade some light on the shape of the energy near the rotor R_i. The Newton's update formula is defined as:

$$R_{i+1} = R_i - (H + \epsilon I)^{-1}(J^T F)R_i$$

where J is the Jacobian matrix of $E(R)$ and the gradient is $\nabla E(R) = (J^T F)R$. Notice that the Jacobian matrix J^j of F^j is $J^j = M_j$, so $J^T F = \sum_j M_j^T M_j$. The Newton's update depends only on the Hessian, since it is shaping the curvature of the 3-sphere manifold.

$$R_{i+1} = R_i - (H + \epsilon I)^{-1} H R_i$$

The Newton's iteration is just aligning R_{i+1} towards the dominant eigenvector of $(H+\epsilon I)^{-1}$. Numerical experiments shown that the column of $(H+\epsilon I)^{-1}$ with higher L1-norm is already close to the dominant eigenvector, specially in the absence of noise.

Newton's method convergence is *quadratic*. However, it has been established that convergence of power iteration is linear or *geometric*, depending on the ratio of the largest eigenvalue to the second eigenvalue $\lambda_2/\lambda_{\max}$ i.e., the method converges slowly if there is an eigenvalue close in magnitude to the largest eigenvalue.

A well-known algorithm for finding dominant eigenvectors of matrices consists of repeatedly squaring a matrix, renormalizing the matrix after each squaring. See [13]. In [5] it is showed that the convergence of the repeated squaring is *quadratic* and for real symmetric $N \times N$ matrices the average number of iterations required is $\mathrm{O}(\ln N + \ln \| \ln \epsilon \|)$. Moreover, the following upper bound is provided $\frac{1}{\ln 2}[\ln(3\sqrt{2}(N(N-1)/2+N)) + 1 + \ln(2\ln N + \| \ln \epsilon \|)] + 1$. Thus for a symmetric 4×4 matrix, taking a small $\epsilon = 10^{-13}$ the average number of iterations is 6 and upper bound is 12. That can be exploited for creating deterministic algorithms.

Several mathematicians have proved bounds on eigenvalues in terms of the value of the matrix. A simple test in [2], is based on the L1-norm of the columns nd rows. Let a_{ij} be the elements, $R_i = \sum_j |a_{ij}|$ be the L1-norm of the ith row and $C_j = \sum_i |a_{ij}|$ the L1-norm of the jth column. Let $\mathcal{R}$ be the largest R_i and $\mathcal{C}$ the largest C_j. Then for all eigenvalues $|\lambda| \leq \min(\mathcal{R}, \mathcal{C})$. Since our matrix is symmetric the upper bound is the same for rows and columns. Going back to the power method, we see that after a small number of repeated matrix squaring, the dominant eigenvector correspond to the jth column having the largest C_j value.

The convergence of our method can be accelerated by squaring the matrix $(H + \epsilon I)^{-1}$ a given number of times before doing the power iteration. We show that in computers with vector instructions our method outperforms previous methods.

5. Optimal Computation of the Hessian

The symmetric matrix $H^j = M_j^T M_j$ has a simple form:

$$S_1 = \langle q_j + p_j \rangle_{12} \ , \ S_2 = \langle q_j + p_j \rangle_{13} \ , \ S_3 = \langle q_j + p_j \rangle_{23}$$
$$D_1 = \langle p_j - q_j \rangle_{12} \ , \ D_2 = \langle p_j - q_j \rangle_{13} \ , \ D_3 = \langle p_j - q_j \rangle_{23}$$

$$H^j = M_j^T M_j =$$
$$\begin{bmatrix} D_1^2 + D_2^2 + D_3^2 & D_3S_2 - D_2S_3 & D_1S_3 - D_3S_1 & D_2S_1 - D_1S_2 \\ D_3S_2 - D_2S_3 & D_1^2 + S_3^2 + S_2^2 & D_1D_2 - S_2S_1 & D_1D_3 - S_3S_1 \\ D_1S_3 - D_3S_1 & D_1D_2 - S_2S_1 & D_2^2 + S_3^2 + S_1^2 & D_2D_3 - S_3S_2 \\ D_2S_1 - D_1S_2 & D_1D_3 - S_3S_1 & D_2D_3 - S_3S_2 & D_3^2 + S_2^2 + S_1^2 \end{bmatrix}$$

Since H^j is symmetric only 10 out of 16 elements need to be actually computed. Notice that the trace of H^j is $Tr(H^j) = 2\|p_j - q_j\|^2 + 2\|p_j + q_j\|^2$ and $Tr(H) = 2\sum_j (\|p_j - q_j\|^2 + \|p_j + q_j\|^2)$.

6. Algorithms

Although our algorithm is based on the inverse power method, which is an iterative algorithm, it always performs a fixed number of matrix squaring operations. Since the upper bound of iterations has been established and numerical tests confirms that no more iterations are needed to achieve the required accuracy. Opposite to other iterative algorithms, we don't need a random guess for R_0. The best R_i is always calculated by matrix squaring.

The proposed method is shown in Algorithm 1.

The C++ code using the Eigen library can be found in Listing 1.

7. Comparisons

We select several representative methods e.g. FLAE [14], SVD [4] and QUEST [9] to implement the algorithm. The Eigen library is employed to implement the SVD and QUEST. Another version of QUEST using the Newton iteration is also added for comparison. The tests are ran on a MacBook Pro 13' 2017 computer with the CPU of Intel 3.1GHz 4-core i5. The Visual Studio 2019 C++ compiler is used in our experiments.

8. Conclusion

In this paper, we solve the problem of estimating the best rotation for the alignment of two sets of corresponding 3D vectors. It is based on solving the linear equations derived from the formulation of the problem in Geometric Algebra. The method is fast, robust to noise, accurate and simpler than most other methods. Experimental validation of the performance of the proposed algorithm is presented. The results show that the slightly losing accuracy of the proposed method is well worth the huge advance in execution time consumption. For many applications the vector datasets are very huge i.e. the

Algorithm 1 Fast Rotor Estimation

Require: $P = \{p_j\}_{j=1}^{n}, Q = \{q_j\}_{j=1}^{n}, C = \{c_j\}_{j=1}^{n}$

$H = 0_{4\times4}, \epsilon = 1 \times 10^{-13}$
for $j = 1$ **to** n **do**
  $S_1 = (q_j + p_j) \cdot e_{21}\ ,\ S_2 = (q_j + p_j) \cdot e_{31}\ ,\ S_3 = (q_j + p_j) \cdot e_{32}$
  $D_1 = (p_j - q_j) \cdot e_{21}\ ,\ D_2 = (p_j - q_j) \cdot e_{31}\ ,\ D_3 = (p_j - q_j) \cdot e_{32}$
  $H = H +$

$$c_j \begin{bmatrix} D_1^2 + D_2^2 + D_3^2 & D_3S_2 - D_2S_3 & D_1S_3 - D_3S_1 & D_2S_1 - D_1S_2 \\ D_3S_2 - D_2S_3 & D_1^2 + S_3^2 + S_2^2 & D_1D_2 - S_2S_1 & D_1D_3 - S_3S_1 \\ D_1S_3 - D_3S_1 & D_1D_2 - S_2S_1 & D_2^2 + S_3^2 + S_1^2 & D_2D_3 - S_3S_2 \\ D_2S_1 - D_1S_2 & D_1D_3 - S_3S_1 & D_2D_3 - S_3S_2 & D_3^2 + S_2^2 + S_1^2 \end{bmatrix}$$

end for
$H_1^+ = (H + \epsilon I)^{-1}$
for $i = 1$ **to** 12 **do**
  $H_i^2 = H_i^+ H_i^+$
  $H_{i+1}^+ = \frac{1}{Tr(H_i^2)} H_i^2$
end for
$\mathcal{C}_j = \max_j \sum_k^4 |H_{12}^+(k,j)|, j = 1, ..., 4$
$R = normalize(\mathcal{C}_j)$
return $R(0) + R(1)e_{12} + R(2)e_{13} + R(3)e_{23}$

computation time would be more important than slight change of accuracy. Therefore, we hope that the proposed algorithm would be of benefit to such applications.

9. C++ code

Listing 1. C++ code for rotor estimation

```
Quaterniond GAFastRotorEstimatorAVX(
    const vector<Vector3d>& P,
    const vector<Vector3d>& Q,
    const vector<double>& w)
{
    Matrix4d H;
    Vector4d R;
    Vector3d S, D;
    double S0, S1, S2, S3, D1, D2, D3, wj;
    const double epsilon = 1e-13;
    const size_t N = P.size(), MAX_STEPS = 12;
    const double* DP = nullptr;

    // Compute H = sum_j M_j*M_j
    H.setZero();
    for (register size_t j = 0; j < N; ++j) {
        wj = w[j];
        S = Q[j] + P[j];
        D = P[j] - Q[j];
        S1 = S.x(); S2 = S.y(); S3 = S.z();
        D1 = D.x(); D2 = D.y(); D3 = D.z();
        H(0, 0) += wj*(D1*D1 + D2*D2 + D3*D3);
        H(1, 0) += wj*(D3*S2 - D2*S3);
```

```
    H(2, 0) += wj*(D1*S3 - D3*S1);
    H(3, 0) += wj*(D2*S1 - D1*S2);
    H(1, 1) += wj*(D1*D1 + S3*S3 + S2*S2);
    H(2, 1) += wj*(D1*D2 - S2*S1);
    H(3, 1) += wj*(D1*D3 - S3*S1);
    H(2, 2) += wj*(D2*D2 + S3*S3 + S1*S1);
    H(3, 2) += wj*(D2*D3 - S3*S2);
    H(3, 3) += wj*(D3*D3 + S2*S2 + S1*S1);
}
// H is lower triangular, complete it to be symmetric
H.selfadjointView<Eigen::Lower>().evalTo(H);
// H = (H + epsilon*I)^-1
H(0, 0) += epsilon; H(1, 1) += epsilon;
H(2, 2) += epsilon; H(3, 3) += epsilon;
H = H.inverse().eval();
// Compute the dominant eigenvector of H using AVX instructions
for (register size_t j = 0; j < MAX_STEPS; ++j) {
    // H *= H
    // H /= Tr(H)
    wj = 1.0 / H.trace();
    __m256d C1 = _mm256_setr_pd(H(0, 0), H(1, 0), H(2, 0), H(3, 0));
    __m256d C2 = _mm256_setr_pd(H(1, 0), H(1, 1), H(2, 1), H(3, 1));
    __m256d C3 = _mm256_setr_pd(H(2, 0), H(2, 1), H(2, 2), H(3, 2));
    __m256d C4 = _mm256_setr_pd(H(3, 0), H(3, 1), H(3, 2), H(3, 3));
    __m256d weights = _mm256_setr_pd(wj, wj, wj, wj);
    // Perform 4 dot products in parallel
    __m256d xy0 = _mm256_mul_pd(C1, C1);
    __m256d xy1 = _mm256_mul_pd(C1, C2);
    __m256d xy2 = _mm256_mul_pd(C1, C3);
    __m256d xy3 = _mm256_mul_pd(C1, C4);
    __m256d temp01 = _mm256_hadd_pd(xy0, xy1);
    __m256d temp23 = _mm256_hadd_pd(xy2, xy3);
    __m256d swapped = _mm256_permute2f128_pd(temp01, temp23, 0x21);
    __m256d blended = _mm256_blend_pd(temp01, temp23, 0b1100);
    __m256d dotproduct = _mm256_mul_pd(
        _mm256_add_pd(swapped, blended), weights);
    DP = (double*)& dotproduct;
    H(0, 0) = DP[0]; H(1, 0) = DP[1]; H(2, 0) = DP[2]; H(3, 0) = DP[3];
    // Perform 4 dot products in parallel
    xy0 = _mm256_mul_pd(C2, C2);
    xy1 = _mm256_mul_pd(C2, C3);
    xy2 = _mm256_mul_pd(C2, C4);
    xy3 = _mm256_mul_pd(C3, C3);
    temp01 = _mm256_hadd_pd(xy0, xy1);
    temp23 = _mm256_hadd_pd(xy2, xy3);
    swapped = _mm256_permute2f128_pd(temp01, temp23, 0x21);
    blended = _mm256_blend_pd(temp01, temp23, 0b1100);
    dotproduct = _mm256_mul_pd(
        _mm256_add_pd(swapped, blended), weights);
    DP = (double*)& dotproduct;
    H(1, 1) = DP[0]; H(2, 1) = DP[1]; H(3, 1) = DP[2]; H(2, 2) = DP[3];
    // Perform 2 dot products in parallel
    xy0 = _mm256_mul_pd(C3, C4);
    xy1 = _mm256_mul_pd(C4, C4);
    temp01 = _mm256_mul_pd(_mm256_hadd_pd(xy0, xy1), weights);
    __m128d hi128 = _mm256_extractf128_pd(temp01, 1);
    __m128d dotproduct2 = _mm_add_pd(
        _mm256_castpd256_pd128(temp01), hi128);
    DP = (double*)& dotproduct2;
    H(3, 2) = DP[0]; H(3, 3) = DP[1];
}
// H is lower triangular, complete it to be symmetric
H.selfadjointView<Eigen::Lower>().evalTo(H);
// Choose the column with higher L1 norm
S0 = H.col(0).lpNorm<1>();
S1 = H.col(1).lpNorm<1>();
S2 = H.col(2).lpNorm<1>();
S3 = H.col(3).lpNorm<1>();
```

```
    wj = std::max(S3, std::max(S2, std::max(S0, S1)));
    if (wj == S0) R = H.col(0);
    else if (wj == S1) R = H.col(1);
    else if (wj == S2) R = H.col(2);
    else R = H.col(3);
    R.normalize();
    return Quaterniond(R(0), R(1), R(2), R(3));
}
```

References

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

[2] A. Brauer. Limits for the characteristic roots of a matrix. *Duke Mathematical Journal*, 13(3):387–395, 09 1946.

[3] L. Dorst and J. Lasenby, editors. *Guide to Geometric Algebra in Practice*. Springer, 2011.

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

[5] E. Kostlan. Statistical complexity of dominant eigenvector calculation. *Journal of Complexity*, 1991.

[6] D. Mortari. EULER-q algorithm for attitude determination from vector observations. *Journal of Guidance, Control, and Dynamics*, 21(2):328–334, 1998.

[7] J. R. Nieto and A. Susín. Cage based deformations: A survey. *Deformation Models: Tracking, Animation and Applications*, 7:75, 2012.

[8] C. B. U. Perwass. *Geometric algebra with applications in engineering*, volume 4 of *Geometry and Computing*. Springer, Berlin; Heidelberg, 2009.

[9] M. D. Shuster and S. D. Oh. Three-axis attitude determination from vector observations. *Journal of Guidance, Control, and Dynamics*, 4(1):70–77, 1981.

[10] F. Sin, D. Schroeder, and J. Barbic. Vega: Non-linear fem deformable object simulator. *Comput. Graph. Forum*, 32(1):36–48, 2013.

[11] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the fifth Eurographics symposium on Geometry processing*, SGP '07, pages 109–116, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[12] G. Wahba. A Least Squares Estimate of Satellite Attitude, 1965.

[13] J. H. Wilkinson, editor. *The Algebraic Eigenvalue Problem*. Oxford University Press, Inc., New York, NY, USA, 1988.

[14] J. Wu, Z. Zhou, B. Gao, R. Li, Y. Cheng, and H. Fourati. Fast Linear Quaternion Attitude Estimator Using Vector Observations. *IEEE Transactions on Automation Science and Engineering*, X(X):1–13, 2017.

[15] Y. Yang. Attitude determination using Newton's method on Riemannian manifold. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 229(14):2737–2742, 2015.

Mauricio Cele Lopez Belon
Buenos Aires, Argentina
e-mail: mclopez@outlook.com