www.cadanda.com
Computer-Aided Design AND APPLICATIONS

# Computing Rotation Minimizing Frames using Quaternions

David Yoon[1], Mark Narduzzi[2] and Jie Shen[3]

[1]University of Michigan - Dearborn, dhyoon@umich.edu
[2]University of Michigan - Dearborn, mnarduzzi@gmail.com
[3]University of Michigan - Dearborn, shen@umich.edu

## ABSTRACT

Frenet Frames have been employed as local frames along a trajectory. However, they behave poorly around inflection points of the path. To remedy this, we propose to compute rotation minimizing frames (RMF) using quaternions.

**Keywords:** 3D curves, Frenet frames, rotation minimizing frames.
**DOI:** 10.3722/cadaps.2012.679-690

## 1 INTRODUCTION

Three dimensional motion enjoys a wide spectrum of applications including robotics, NC machining, and 3D animation, just to name a few. It typically depicts an object moving along a path. The position and orientation of the object are represented in terms of a moving frame at a particular point on the trajectory. It is very important to select the appropriate moving frame at a particular point on the trajectory. Traditionally, the Frenet Frame has been employed as moving frames. However, over the years the drawbacks of the frame have been identified and a number of different approaches to remedy these problems have emerged. Since the trajectory and the Frenet frame form the basis for 3D motions, they are briefly introduced in the next section.

## 2 TRAJECTORY AND FRENET FRAME

Let $P(u) \in R^3$ be a $C^2$ curve, where $u \in U$. Then the position and orientation of an object is specified in terms of a particular point on the trajectory and the moving frames as follow:

$$PO(u) = [P(u), F(u)]$$

where P( ) is the trajectory and the moving frame F(u) = [r(u), s(u), t(u)] and r(), u(), and t() represent 3D orthonomal vectors [6]. Oftentimes the Frenet Frame has been employed as the moving frame as follows:

$$t(u) = \frac{P'(u)}{|P'(u)|}, \quad s(u) = \frac{P'(u)xP''(u)}{|P'(u)xP''(u)|}, \quad \text{and } r(u) = r(u) \times s(u).$$

The Frenet Frame serves as the moving frame very well as long as there is no inflection point along the trajectory, i.e. P$\acute{o}$(u) = 0. However, when there is an inflection point, the sign of s() changes and this induces undesirable consequences. To remedy this, there are a number of solutions published. In this section, some of the well-known approaches are reviewed and we will present our approach in the next section. The papers summarized below may be characterized as an effort to produce a frame in which the binormal vector does not change the sign around the inflection point. This type of a frame is known as a Rotation Minimizing Frame (RMF).

Siltanen/Woodward [10] summarize the behavior of the principle normal vector around an inflection point and propose the projection normal method. In this approach, they compute the tangent vector (in our notation t()), the normal vector (in our notation r()) and the binormal vector (in our notation s()) at each knot point. The normal vector $r_i$ at knot point $u_i$ is computed as follows:

$$r_i = r_{i-1} - (r_{i-1} \cdot t_i)\, t_{i-1}$$

where · represents dot product and $r_i$, $s_i$, and $t_i$ are normalized. This approach is intuitive and easy to implement.

Klok [3] considers moving frames in the context of sweeping. When there is no inflection point along the trajectory, the Frenet frames function as the moving frames. However, when there is an inflection point, a Rotation Minimizing Frame (RMF) has to be computed. A moving frame that does not exhibit discontinuous rotation about the tangent is called a rotation minimizing frame. Due to the RMF's smooth rotation as the moving frame crosses an inflection point, the RMF is preferred over the Frenet frame in many applications of computer graphics including sweep surface modeling [9], generalized cylinders [12], motion design and control [6]. He formulates the problem as differential equations on normal vector and binormal vector. The interested reader is referred to his paper.

Guggenheimer [1] formulates the rotation minizing problem as an integration problem rather than a differential equation problem and claims that his approach is computationally faster. Shani/Ballard's [8] approach is in a way similar to Guggenheimer's in that they view rotation minimizing problem as an integration problem.

Wang/Juttler [11] introduce the double reflection method to come up with rotation minimizing frames. As one can easily see, the authors deviate from previously mentioned researchers and formulate the problem in terms of the basic transformation reflection.

Rossignac/Kim [7] split a motion into the translational and rotational components and consider the rotational component in terms of a screw motion. They compute the parameters of a motion from two motion frames. It appears that the rotation between the two motion frames is minimized by explicitly computing the angle between them rather than relying on the two principle normal vectors.

In both refs [7,11], the authors both implicitly and explicitly compute the axis of rotation and the angle of rotation and obtain the rotation minimization frame. Generalizing these approaches, we will compute RMFs using quaternions.

## 2. 1 Quaternions

An intuitive view of a quaternion is that it represents an axis of rotation in $R^3$ along with the rotation angle. Following Hanson [2], one can consider the axis of rotation $\mathbf{n} = [n_1, n_2, n_3]$ as a unit vector with an angle of rotation $\theta$ in the polar coordinate system. Then the quaternion **q** may be represented as a 4-dimensional vector $q = (w + ix + jy + kz)$ = (w, $\vec{v}$) and its component may be expressed as follows:

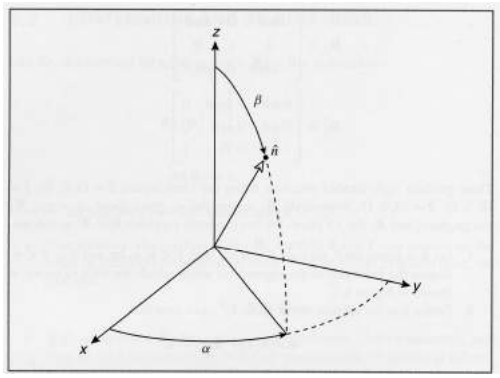


Fig. 1: The Axis of Rotation on $S^2$ [2].

$$w = \cos(\theta/2)$$
$$x = n_1 \sin(\theta/2)$$
$$y = n_2 \sin(\theta/2)$$
$$z = n_3 \sin(\theta/2)$$

Quaternions, represented by **q**, are a part of a subset of the hypercomplex numbers, with the form:

$$w + xi + yj + zk$$

where

$$i^2 = j^2 = k^2 = ijk = -1$$
$$ij = k, ji = -k$$
$$jk = i, kj = -i$$
$$ki = j\ ik = -j$$

Any vector of $\vec{v} \in \mathbf{R}^3$ can be mapped into a quaternion space:

$$\begin{bmatrix} 0 \\ \vec{v} \end{bmatrix}$$

This type of a quaternion is known as a **pure quaternion.**

## 2.2. Quaternion Reflections

Quaternions can be used to compute reflections with respect to a plane. Let $\overline{v} = [x, y, z]^T$ be a unit vector normal to the plane of reflection and $q_r = (0, \overline{v})$. To compute the reflection of $p$, compute [9]:

$$q_2 = q_r \times p \times q_r$$

To understand reflection, consider reflection with respect to the x-y plane. The basis vector $k$ is normal to the plane of reflection. From the relationship between $i$, $j$, and $k$, we have:

$$kik = i$$
$$kjk = j$$
$$kkk = -k$$

Thus only the $z$ component of a vector is inverted, the $x$ and $y$ components are unchanged.
Reflection differs from rotation in two ways. First, the reflection quaternion must be a pure quaternion. Second, multiplication on the right is by the same quaternion as on the left. For rotations, multiplication on the right is by the conjugate.

In general, it's not possible to compose an arbitrary number of reflections like rotations. It is, however, possible to combine two reflections into one rotation. Consider two pure, unit quaternions, $q_{r1}$ and $q_{r2}$. Each represents a vector normal to a plane of reflection. To reflect a point $p$ through each plane, compute:

$$q_1 = q_{r1} \times p \times q_{r1}$$

This represents the reflection of $p$ with respect to plane 1.

$$p_2 = q_{r2} \times q_1 \times q_{r2}$$

$p_2$ represents the reflection with respect to plane 2. Combining these two equations:

$$p_2 = q_{r2} \times (q_{r1} \times p \times q_{r1}) \times q_{r2} = (q_{r2} \times q_{r1}) \times p \times (q_{r1} \times q_{r2})$$

Both $q_{r1}$ and $q_{r2}$ are pure quaternions and therefore $-q_{r1} = conj(q_{r1})$ and $-q_{r2} = conj(q_{r2})$.

$$q_{r1} \times q_{r2} = (-q_{r1}) \times (-q_{r2}) = q_{r1}^* \times q_{r2}^* = (q_{r2} \times q_{r1})^*$$

Finally, for two reflections:

$$q_r = q_{r2} \times q_{r1}$$
$$p_2 = q_r \times p \times q_r^*$$

Therefore, two reflections are equivalent to one rotation. However, the product $q_{r2} \times q_{r1}$ is not necessarily a pure quaternion so we cannot compose a third reflection into a single rotation. However,

it can be shown that the composition of an even number of reflections can be written as a single rotation.

### 2. 3 Transforming the Double Reflection Method to a Single Quaternion Rotation

We now have the tools to transform the double reflection method into a set of quaternion equations. The name "double reflection" suggests the approach we take. Each reflection is represented by a quaternion and the two reflection quaternions are composed to produce one rotation quaternion. Thus $R_i$ undergoes a single rotation to produce $R_{i+1}$.

The first plane of reflection in the double reflection method is the bisecting plane between $P_i$ and $P_{i+1}$. Expressed as a pure quaternion, $q_1 = p_{i+1} - p_i$. However, to produce a reflection without scaling, $q_1$ must also be a unit quaternion. Define a function *unit(q)* that maps a non-zero quaternion to the unit hyper-sphere, $S^3$:

$$unit(q) = \frac{q}{\|q\|} = \frac{q}{\sqrt{N(q)}}$$

Thus the proper equation for the first reflection quaternion is:

$$q_1 = unit\left(P_{i+1} - P_i\right)$$

To compute the quaternion representing the second plane of reflection, we need the unit quaternion for $T_{i+1}$ **ð ò**reflection of $T_i$ in plane 1**ó**:

$$q_2 = unit\left(T_{i+1} - q_1 \times T_i \times q_1\right)$$

The first reflection of $R_i$ is computed the same as for $T_i$:

$$R_i^L = q_1 \times R_i \times q_i$$

The second reflection of $R_i$ is computed by reflecting $R_i^L$ using $q_2$:

$$R_{i+1} = q_2 \times R_i^L \times q_2$$

By substituting $R_i^L = q_1 \times R_i \times q_i$ into $R_{i+1} = q_2 \times R_i^L \times q_2$, we can compute $R_{i+1}$ directly from $R_i$ with a single rotation:

$$R_{i+1} = q_2 \times R_i^L \times q_2 = q_2 \times \left(q_1 \times R_i \times q_1\right) \times q_2 = \left(q_2 \times q_1\right) \times R_i \times \left(q_1 \times q_2\right) = \left(q_2 \times q_1\right) \times R_i \times \left(q_2 \times q_1\right)^*$$

Or more simply:

$$q_r = q_2 \times q_1$$

$$R_{i+1} = q_r \times R_i \times q_r^*$$

Here we arrive at our goal: the double reflection method can be expressed by the following set of quaternion equations:

$$q_1 = unit\left(P_{i+1} - P_i\right)$$
$$q_2 = unit\left(T_{i+1} - q_1 \times T_i \times q_1\right)$$
$$R_{i+1} = \left(q_2 \times q_1\right) \times R_i \times \left(q_2 \times q_1\right)^*$$
$$S_{i+1} = T_{i+1} \times R_{i+1}$$

### 2.4 RMF by Quaternion Rotation Implementation [5]

A pseudo code implementation of double reflection by quaternion rotation is shown in the algorithm below. All points and vectors are used as pure quaternions.

**Algorithm - RMF by quaternion rotation**

Input:

Points $P_i$, and unit tangent vectors, $T_i$, for i = 0 **é** n, and an initial reference vector $R_0$.

Output:

$U_i$ = ( $R_i$, $S_i$, $T_i$), for i = 1 **é** n, as an approximate RMF

```
for i = 0 to n-1
q1 = unit(P[i+1] - P[i])              ; first reflection quaternion
q2 = unit(T[i+1] - q1*T[i]*q1)        ; second reflection quaternion
q = q2 * q1                                 ; rotation quaternion
R[i+1] = q*R[i]*conj(q)                     ; rotate reference vector
S[i+1] = T[i+1] x R[i+1]              ; 3D vector cross product
U[i+1] = ( R[i+1], S[i+1], T[i+1] )
End
```

## 3 AN ILLUSTRATION

The system developed as part of our approach allows a user to specify a parametric curve, display the curve, the Frenet frames, and rotation minimizing frames computed using both the double reflection method and quaternion rotation. Further, this system demonstrates double reflection step-by-step, and performs a timing comparison between quaternion rotation and double reflection.

### 3.1 Command line options

The application, rmf, has several command line options as shown below:

```
~/misc/masters$rmf -?
usage: rmf -c (-q | -d | -f) -r -s -t -p filename -h -F -?

  -c            draw parametric curve
```

```
-q            draw quaternion RMF
-d            draw double-reflection RMF
-f            draw frenet frame
-p filename   load parametric equations from filename
-r            draw normal/reference vector
-s            draw s vector
-t            draw tangent vector
-h            home view (no rotation)
-F            draw each component of the parametric curve
-?            this help text
```

These options start the application and immediately draw the specified items.
The default curve is:

$$x(t) = \frac{(t)(t-\frac{1}{4})(t+\frac{1}{4})}{4} = 0.25t^3 - 0.015625t$$

$$y(t) = \frac{(t^2)(t-1)}{4} - 2 = -0.25t^3 + 0.25t - 2$$

$$z(t) = 3 - \frac{(t)(t-3)(t+2)}{2} = -0.5t^3 + 0.5t^2 + 3t - 3$$

This set of polynomials was chosen to clearly demonstrate the difference between Frenet frames and RMFs.

### 3.2 Screen captures

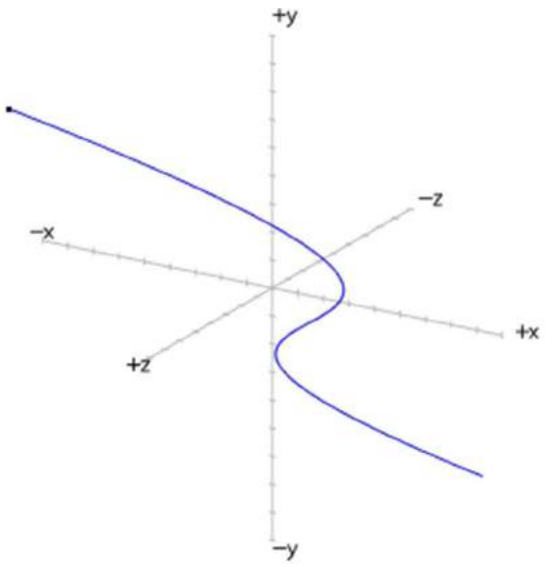


Fig. 2: Initial view with parametric curve.

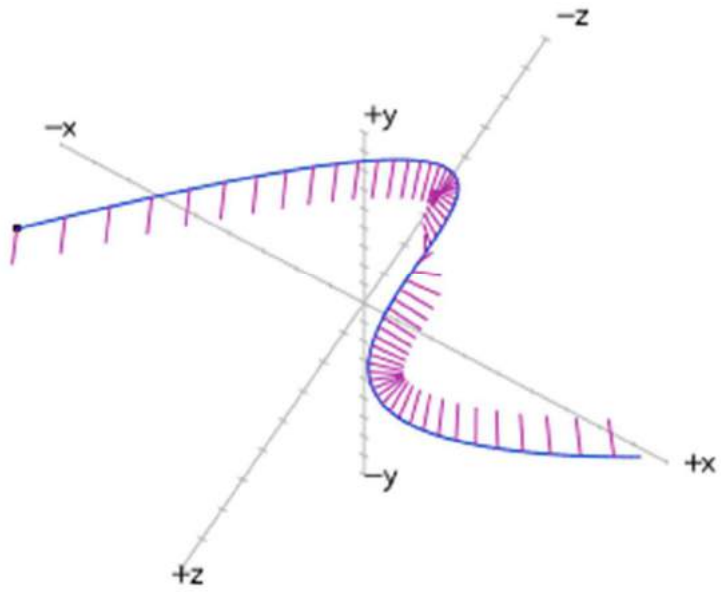


Fig. 3: Frenet frames with reference vector displayed.

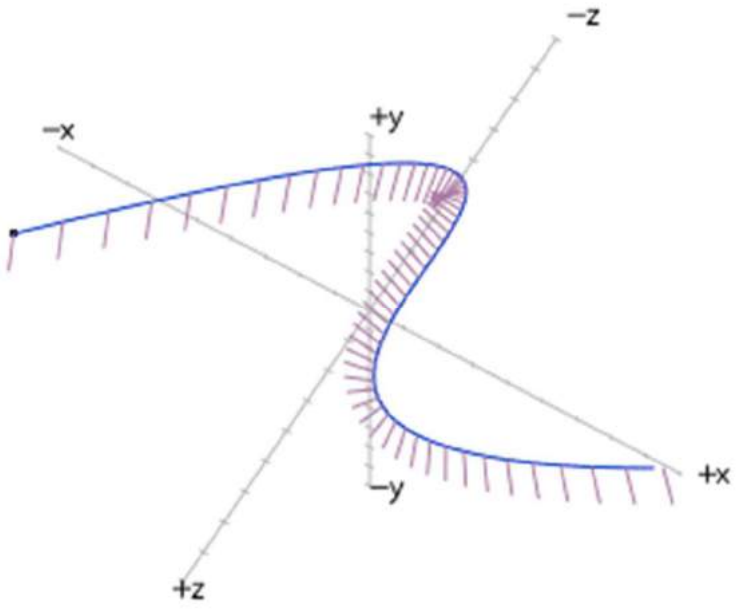


Fig. 4: Rotation minimizing frames using double reflection.

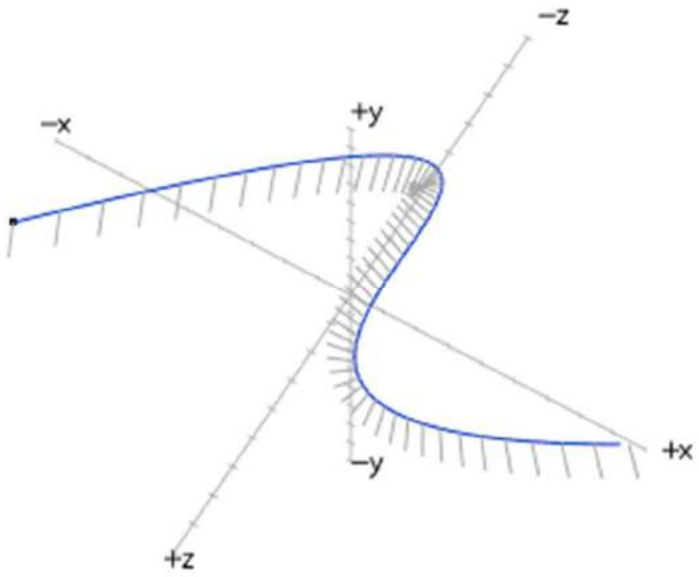


Fig. 5: Rotation minimizing frames using quaternion rotation.

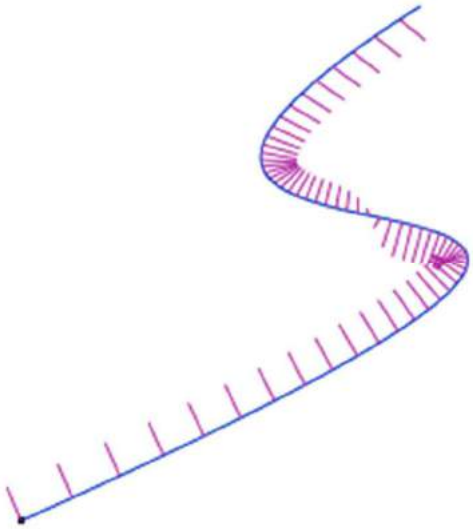

Fig. 6: Frenet frames (different perspective).

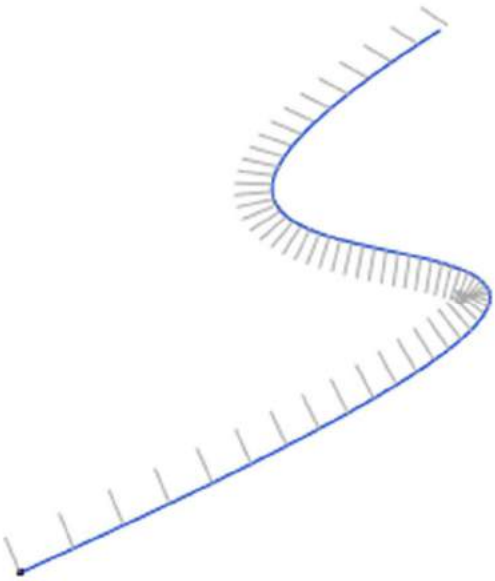

Fig. 7: Rotation minimizing frames using quaternions (different perspective).

## 4 CONCLUSION

One advantage the quaternion rotation method has over double reflection [11] is that intermediate reference vectors need not be computed. Indeed, it is possible to compute $R_N$ directly from the initial reference vector $R_0$ by composing all the rotations computed at each step. Instead of producing a series of RMFs along a curve, it is possible to generate a series of rotation quaternions and compute the RMF at a point only if required (i.e. not hidden). If the initial reference vector requires further rotation, simply compose the same rotation with the series of rotation quaternions as needed.

## REFERENCES


[1] Guggenheimer, H.: Computing Frames Along a Trajectory, Computer-Aided Geometric Design 6, 1989, 77-78.

[2] Hanson,A. J.: Visualizing Quaternions, Morgan Kaufman, 2006

[3] Klok, F., Two Moving Coordinate Frames for Sweeping Along a 3D Trajectory, Computer-Aided Geometric Design, 3, 1986, 217-229.

[4] Maths Transformations using Quaternions: http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/transforms/index.htm

[5] Narduzzi, M, ðRotation Minimizing Frames by Quarternion Rotations,ó MS Project report, CIS Department, University of Michigan ð Dearborn, 2011

[6] Pobegailo, A.P.: Design of motion along parameterized curves using B-slpines, Computer-Aided Design, 35, 2003, 1041-1046.

[7] Rossignac, J.R.; Kim, J.J.: Computing and Visualizing Pose-Interpolating 3D Motions, Computer-Aided Design 33, 2001, 279-291.
[8] Shani,U, Ballard, D.H.: Splines as Embeddings for Generalized Cylinders, Computer Vision, graphics, and Image Processing, 27, 1984, 129-156.
[9] Shoemake, K.:Quaternions, http://www.cs.caltech.edu/courses/cs171/quatut.pdf
[10] Siltanen, P.; Woodward, C.: Normal Orientation Methods for 3D Offset Curves, Sweep Surfaces and Skinning. EUROGRAPHICS ð92
[11] Wang, W.; Juttler, B.; Zheng, D.; Liu, Y.: Computation of Rotation Minimizing Frames, ACM Transactions on Graphics, 27(1), Article 2, Publication date: March 2008
[12] Yoon, D, Shen, J, Ohou, E. òGeneralized Cylinders in Reverse Engineering,ó Proc of the 2010 Int. Conf. on Computer Graphics & Virtual Reality, 83-87, July 12-15, 2010, Las Vegas, Nevada

## APPENDIX: QUATERNION ALGEBRA [4]

Let **q** and **q'** be quaternions. Then the following holds:

### *ADDITION*

$$\mathbf{q} + \mathbf{q}\delta = \mathbf{q}\delta + \mathbf{q} = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} + \begin{bmatrix} w' \\ x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} w+w' \\ x+x' \\ y+y' \\ z+z' \end{bmatrix}$$

### *MULTIPLICATION*

$$\mathbf{q}q' = \begin{bmatrix} ww' - \vec{v} \bullet \vec{v}' \\ \vec{v} \times \vec{v} + w\vec{v}' + w'\vec{v} \end{bmatrix}$$

### *MULTIPLICATION FACTS*

**(pq)p' = p(qp')**

$$1\mathbf{q} = \mathbf{q}1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}$$

$$w\delta q = qw\delta = w\delta \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} w'w \\ w'x \\ w'y \\ w'z \end{bmatrix}$$

***CONJUNCTIVE***

$$q^* = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}^* = \begin{bmatrix} w \\ -x \\ -y \\ -z \end{bmatrix}$$

***CONJUNCTIVE** FACTS*

$(q^*)^* = q$
$(pq)^* = q^*p^*$
$(p+q)^* = p^*+q^*$

*NORM* : $\mathbf{N(Q) = X^2 + Y^2 + Z^2 + W^2}$

***NORM FACTS***

$N(qq\bar{q}) = N(q)N(q\bar{q})$

***INVERSE***

$q^{-1} = q^* / N(q)$

***UNIT QUATERNION***

$q = q / N(q)$
$N(qq\bar{q}) = 1$
$q^{-1} = q^*$

***PURE QUATERNION***

$$q = \begin{bmatrix} 0 \\ x \\ y \\ z \end{bmatrix} \quad -q = q^*$$

**QUATERNION ROTATIONAL FACTS**

Quaternion rotation is well documented, and does not need to be repeated here. The equation to rotate a point v around a quaternion q is below:

$$w = qvq^*$$