Quaternions

Introduction
Thus far we have represented a rotation using a 3x3 orthonormal matrix $R$. The columns of $R$ are the axes of the rotated reference frame. The product of $Rv$, where $v$ is the 3D point $(x, y, z)$, is $v$ transformed into the coordinate system (i.e. rotated) of $R$. We packed $R$ and a translation vector into a $4 \times 4$ matrix to form a complete coordinate frame.

A matrix is not the only possible representation of a rotation. This chapter introduces the unit quaternion, $q$, as an alternative. Unit quaternions have several advantages over rotation matrices:

- Given two rotations $q_1$ and $q_2$, it is easy to find intermediate rotations between them. This process is called spherical linear interpolation ($slerp$)
- $q$ requires only four components, less than half the space of $R$
- Numerical integration of $q$ produces less error than for $R$
- When error does accumulate, it is easy to normalize $q$
- It is easy to construct $q$ from an axis $u$ and angle $\theta$ of rotation

Quaternions also have several disadvantages. Our software and hardware infrastructure is matrix-oriented. It is therefore much more practical to transform points in and out of coordinate systems using $R$. Quaternion math is less intuitive than “vector” math because $q$ is a four-dimensional quantity, which we cannot directly visualize. Quaternions are also not unique—for every possible rotation there is only one $R$ but there are two different values of $q$. This is both confusing and a possible pitfall when comparing and interpolating between quaternions.

The quaternion representation is commonly used for physical simulation because of its robustness under numerical integration. It is also used for animation blending where the $slerp$ function is needed. Although it is possible to transform points using the quaternion directly, it is usually easiest to convert it to a rotation matrix using the equation described in this chapter and multiply the matrix by a point.

Definition
A **quaternion** is a four-component vector. The **imaginary** part $q_v$ is a 3-vector ($q_x, q_y, q_z$). The **real** part $q_w$ is scalar. A quaternion can also be written as the sum of four terms:

$$q = (q_x, q_y, q_z, q_w) = (q_v, q_w) = q_x i + q_y j + q_z k + q_w$$

**Definition of Quaternion**

In these equations, $i$ is the vector $(1, 0, 0, 0)$. It is also a square root of $-1$. Just as positive numbers have both a positive and a negative root, a negative number can have several different roots. Quaternions use three square roots of $-1$ that are related by:
Note that multiplication does not commute between these special quantities, for example, $ij \neq ji$.

Given that you already understand complex numbers, the $i, j, k$ system is not so unusual. We have just extended complex numbers from two $(a + bi)$ to four dimensions $q_xi + q_yj + q_zk + q_w$. We (or rather, Hamilton, the inventor of quaternions) skipped over three dimensions because the cyclic relationship between $i, j, k$ is necessary and does not exist in three dimensions.

You may have seen complex numbers used to represent points on the 2D plane, where the real part represented a location on the $x$-axis and the imaginary part represented a location on the $y$-axis. In other words, the value $x + yi$ is the point $(x, y)$. Note that the $i$ is implicit in the vector notation and is not part of the $y$-component itself. Euler’s relation, $e^{i\theta} = \cos \theta + i \sin \theta$

Euler’s relation in 2D

is intuitive in the complex plane because it matches the math for 2D rotation. A point at angle $\theta$ from the $x$-axis and a distance of 1 from the origin has coordinates $(\cos \theta, \sin \theta) = \cos \theta + i \sin \theta$. We can extend the idea of complex numbers as points in space to four dimensions.

Let $q$ be the four dimensional point $(q_x, q_y, q_z, q_w)$. As for 2D, the $i, j, k$ values are not part of the components, which are themselves just real scalars. We only use $i, j, k$ when writing the quaternion (or complex number) as an algebraic expression. All three-dimensional points have the form $(q_x, q_y, q_z, 0) = q_xi + q_yj + q_zk$. That is, their real part $q_w$ is zero. This is called a pure quaternion or pure vector quaternion. Although you may not have known it previously, the vector math to which you are accustomed is really just a subset of quaternion math that only considers pure quaternions. Between your understanding of complex numbers and vector math, quaternions are not a new kind of mathematics so much as putting what you already know into context.

When working with transformation matrices and projective geometry, a direction is a 4-vector with the $w$-component equal to 0. A point is a 4-vector with $w = 1$. Quaternions are also 4-vectors, but we are not considering their projections onto 3-space. Because we only consider rotations in this chapter, the origin never moves and there is no distinction between a point relative to the origin and the vector from the origin to that point. Any point or direction therefore has $q_w = 0$. It may help you to think of pure quaternions as a
totally new representation for vectors that also happens to use four components, but \( q_w \neq w \) ...although both kinds of 4D vectors really are “the same thing” mathematically and we’re applying different kinds of operations to them.

Euler’s relation holds for quaternions as well as complex numbers. Let \( u \) be a 3D vector and \( \frac{1}{2} \theta \) be a scalar. Because \( u \) is a vector, it carries the \( i, j, k \) values with it and exponentiation gives:

\[
e^{\frac{1}{2} \theta u} = (u \sin \frac{\theta}{2}, \cos \frac{\theta}{2}) = u_x i \sin \frac{\theta}{2} + u_y j \sin \frac{\theta}{2} + u_z k \sin \frac{\theta}{2} + \cos \frac{\theta}{2}
\]

Euler’s Relation in 4D

Although the equation could be written with \( \theta \) instead of \( \frac{1}{2} \theta \), we’ll see shortly why it is useful to work with the half-angle.

**Quaternion Operations**

Using the cyclic relationships established between \( i, j, \) and \( k \) we can define operations on quaternions in terms of the underlying algebra. For example, the sum of quaternions \( q \) and \( r \) is:

\[
q + r = q_x i + q_y j + q_z k + q_w + r_x i + r_y j + r_z k + r_w
\]

This expression is easier to write using the notation where \( q_v \) is the imaginary (vector) part of \( q \):

\[
q + r = \left( q_v + r_v, q_w + r_w \right)
\]

**Quaternion Sum**

The **conjugate** of a quaternion has the same real part but a negated imaginary part:

\[
q^* = (-q_v, q_w)
\]

**Quaternion Conjugate**

The **dot product** of two quaternions is the sum of the products of their components. It is always a real scalar. Notice how the 3D dot product with which you are already familiar fits within this definition for pure quaternions:

\[
q \cdot r = q_x r_x + q_y r_y + q_z r_z + q_w r_w
\]

**Quaternion Dot Product**
The **magnitude** of a quaternion is the square root of its dot product with itself (as it is for a 3D vector). This is also the 2-norm:

\[ n(q) = \sqrt{q \cdot q} \]

**2-Norm (Magnitude) of a Quaternion**

A **unit quaternion** has \( n(q) = 1 \). These are the quaternions that we use to represent rotations (a non-unit quaternion is a rotation and a scale factor). Any non-unit quaternion can be projected onto the unit hyper-sphere (remember, we are in *four* dimensions) by dividing out the magnitude:

\[ q = \frac{r}{n(r)} \]

**Normalizing a Quaternion**

The product of quaternions \( q \) and \( r \) is derived as follows:

\[ qr = (\text{see RTR page 43})... \]

\[ = (q_w r_v + q_v \times r_v + r_w q_v, q_w r_w - q_v \cdot r_v) \]

\[ qr = (q_w r_x - q_z r_y + q_y r_z + q_x r_w, \]

\[ q_z r_x + q_w r_y - q_x r_z + q_y r_w, \]

\[ -q_y r_x + q_x r_y + q_w r_z + q_z r_w, \]

\[ -q_x r_x - q_y r_y - q_z r_z + q_w r_w) \]

**Quaternion Product**

Although we encountered vectors first in this text, historically quaternions were discovered first and the cross product and dot product emerged from the quaternion product definition.

As for matrix multiplication the quaternion product does not commute (\( qr \neq rq \) in general). In fact, we can convert quaternion multiplication directly into 4×4 matrix multiplication. This is done by converting the previous equation into matrix form. The matrix \( L_q \) such that multiplying quaternion \( r \) on the left \( L_q r^T \) is equal to multiplying \( qr \) is:

\[ L_q = \begin{bmatrix} q_w & -q_z & q_y & q_x \\ q_z & q_w & -q_x & q_y \\ -q_y & q_x & q_w & q_z \\ -q_x & -q_y & -q_z & q_w \end{bmatrix}; \quad L_q r^T = qr \]

**Matrix Equivalent to Right Quaternion Multiplication**
The corresponding matrix for multiplying on the right is:

\[
R_r = \begin{bmatrix}
  r_w & r_z & -r_y & r_x \\
  -r_z & r_w & r_x & r_y \\
  r_y & -r_x & r_w & r_z \\
  -r_x & r_y & -r_z & r_w \\
\end{bmatrix}; \quad R_r^T q = qr
\]

**Matrix Equivalent to Right Quaternion Multiplication**

Note that these matrices are skew symmetric (they are equal to their transpose with off-diagonal entries negated), like the cross-product matrix. The fourth column of each is the quaternion itself. The other columns are permutations of the fourth column, with an element negated for each row swap. The only difference between the \( L \) and \( R \) matrices is which elements are negated. For each matrix, the fourth row is the conjugate of the quaternion.

Also as with matrices, we avoid explicitly defining division and instead give an expression for the inverse of a quaternion:

\[
q^{-1} = \frac{q^*}{n(q)}
\]

**Quaternion Inverse**

Division is then accomplished by multiplying by an inverse. Note that for a unit quaternion, \( n(q) = 1 \) so the inverse is the conjugate \( q^{-1} = q^* \).

**Unit Quaternions as Rotations**

Quaternions are useful for computer graphics because quaternions on the unit hypersphere (i.e., of unit length) can represent 3D orientations or rotations. Following Euler’s relation, let the rotation by angle \( \theta \) about axis \( u \) be represented by the unit quaternion:

\[
q = e^{\frac{1}{2} \theta u} = (u \sin \frac{\theta}{2}, \cos \frac{\theta}{2})
\]

**Unit Quaternion from Axis Angle**

Under this representation, a direction or a point is represented by a so-called pure quaternion with \( w = 0 \). The result of multiplying a pure quaternion \( p \) by a quaternion \( q \) and its inverse \( q^{-1} \) is another pure quaternion that is the rotation of \( p \) by \( \theta \) about \( u \):

\[
r = qpq^{-1}
\]

**Rotation of point \( p = (x,y,z,0) \) by \( q \)**

We can also convert the previous expression into a rotation matrix. Let \( q \) be a unit quaternion. Let \( L_q \) be the matrix that multiplies by \( q \) on the left and \( R_{q^*} \) be the matrix that
multiplies by \( q^* = q^{-1} \) on the right as previously defined. The matrix \( M = L_q R_q^* \) multiplies by both... it is the rotation matrix corresponding to a rotation by \( q \) about the origin!

Using the equations given in the previous section we can derive the net matrix:

\[
M = \begin{bmatrix}
1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_wq_z & 2q_zq_x + 2q_wq_y & 0 \\
2q_xq_y + 2q_wq_z & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_x - 2q_wq_x & 0 \\
2q_xq_z - 2q_wq_y & 2q_yq_z + 2q_wq_x & 1 - 2q_x^2 - 2q_y^2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}; \quad Mp^T = qpq^*
\]

Rotation Matrix from Unit Quaternion

Because the fourth row and column are \([0 \ 0 \ 0 \ 1]\), we do not need to append \( p_w = 0 \) to the point to make a pure vector quaternion. The \( w \)-component is irrelevant to this multiplication and can be our projective \( w \)-component; in other words, only the upper 3x3 submatrix is significant.

Computing a unit quaternion from a rotation matrix \( M \) is the inverse of the process of constructing the above matrix. The following routine ensures numerical stability during the computation by delaying any normalization until the end. In the routine, \( a \) is the index of the largest diagonal element of the matrix.

\[
\text{quatFromMatrix}(M) \\
a \leftarrow \text{argmax} \ M_{a,a} \\
b \leftarrow (a + 1) \mod 3 \\
c \leftarrow (a + 2) \mod 3 \\
q_x \leftarrow M_{b,b} + M_{c,c} - M_{a,a} - 1 \\
q_y \leftarrow -(M_{a,b} + M_{b,a}) \\
q_z \leftarrow -(M_{a,c} + M_{c,a}) \\
q_w \leftarrow M_{b,c} + M_{c,b} \\
\text{return } q / ||q||
\]

Spherical Linear Interpolation

The quaternion representation is well-suited to interpolation by great-circle routes around a sphere. This defines the shortest constant-velocity curve between two 3D orientations. It is useful for smoothly interpolating between joint or camera positions. The operation is called “slerp” as a contraction of spherical linear interpolation. The interpolated orientation at time \( 0 < t < 1 \) between the orientation given by quaternion \( r \) at \( t = 0 \) and quaternion \( q \) at \( t = 1 \) is given by:
\[ \text{slerp}(r, q, t) \]

\[
c ← r \cdot q
\]

if \( c < 0 \) then

\[
r = -r
\]

\[
c ← -c
\]

\[
φ ← \cos^{-1}(c)
\]

if \( φ ≥ ε \) then

\[
\text{return } \frac{r \sin((1-t)φ) + q \sin(tφ)}{\sin φ}
\]

else

\[
p ← r + (q - r) t
\]

\[
\text{return } p / ||p||
\]

The \( c \) variable is the cosine of the angle (in 4D) between the quaternions; as for angles between vectors in 2D and 3D, it is given by the dot product of the quaternions. Because every 3D orientation is represented by two different quaternions that are negatives of one another, it is possible for the same two orientations to yield either a negative or positive cosine. A negative cosine corresponds to interpolation the long way around the sphere, equivalent to interpolating from 45-degrees to 315-degrees by way of 180-degrees instead of by way of 0-degrees. To avoid this, we negate the cosine and one of the quaternions in this case.

The actual angle between the quaternions is recovered by \( \cos^{-1} \); assume that it is implemented such that the result is always \( 0 < φ < \pi \). For sufficiently large \( φ \) the slerp is given by linear interpolation on the sines of the angle normalized by the sine of \( φ \). However, when \( φ \) is small \( \sin φ \) is also small and the quotient is numerically unstable. In this case we use direct linear interpolation between the quaternions, re-normalizing after interpolation. Because the quaternions are already nearly equal, the non-linear velocity produced by linear quaternion interpolation is typically acceptable in that case.