Quaternions and Spherical Interpolation

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

Linear Interpolation to Blend Two Quantities

\[ Q = (1-t)Q_0 + tQ_1 \]

A Useful Concept: Spherical Interpolation

\[ Q(t) = \frac{\sin(1-t)\phi}{\sin\phi} P + \frac{\sin\phi}{\sin\phi} Q \]

where:
\[
\cos\phi = \frac{P}{|P|} \cdot \frac{Q}{|Q|} = (p_x d_1 + p_y d_2 + p_z d_3)
\]
\[
\sin\phi = \sqrt{1 - \cos^2\phi}
\]

A Review of Complex Numbers

\[ z = x + iy = r[\cos\theta + i\sin\theta] = re^{i\theta} \]

\[ z \cdot z' = (x + iy)(x' + iy') = (x_1 x_2 - y_2 y_1) + i(x_1 y_2 + x_2 y_1) \]

Note:
- \(|z||z'| = r^2\)
- \(|z'z| = 1\)
- If \( r = 1 \), then \( z^* = z \)

Quaternion Multiplication

\[ PQ = (p_1 + ip_2 + kp_3)(q_1 + iq_2 + kq_3) \]

\[ P \times Q = \frac{p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3}{\sqrt{|P|^2|Q|^2}} \]

\[ P \times Q = \frac{p_0q_0 + p_1q_1 + p_2q_2 + p_3q_3}{\sqrt{|P|^2|Q|^2}} \]

Discovered by Sir William Hamilton, 1843, while on a walk in Dublin.

Legend says that he was so excited that he took out a knife and carved
the equation into the stone of a bridge. It’s a good thing spray paint hadn’t
been invented yet…

Quaternions have 4 elements, one real and three complex:

\[ Q = (q_0, q_1, q_2, q_3) \]

By definition:
\[
ii = jj = kk = -1 \quad ij = k, jk = i, ki = j
\]
\[
jk = -k, ij = -i, ik = -j \quad ik = -1
\]

And, by definition, we always force \(|Q| = 1\). by making \(q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1\)
Performing Rotations with Quaternions

A Quaternion can record a rotation transformation by an angle \( \theta \) about an axis \( \mathbf{n} \) like this:

\[
[R(\theta, \mathbf{n})] = [I(q, \mathbf{n})]
\]

where:
\[
q = \cos\left(\frac{\theta}{2}\right)
\]
\[
\mathbf{n} = \sin\left(\frac{\theta}{2}\right)
\]

Concated Quaternion Rotations are handled like this:

\[
R(\theta, \mathbf{n}) = R_2(\theta_2, \mathbf{n}_2)R_1(\theta_1, \mathbf{n}_1)
\]

A rotation takes effect first, followed by \( R_2 \).

A Quaternion can represent a point \( P \) like this:

\[
P' = \begin{pmatrix}
q_0^+ + q_1^+ - q_2^- + q_3^- \\
q_0^+ - q_1^- + q_2^+ - q_3^+ \\
2q_1q_2 + 2q_3q_0 \\
2q_2q_3 - 2q_0q_1
\end{pmatrix}
\]

Performing Rotations with Quaternions

A rotated point, \( P' \) by one rotation is:

\[
P' = \begin{pmatrix}
q_0^+ + q_1^+ - q_2^- + q_3^- \\
q_0^+ - q_1^- + q_2^+ - q_3^+ \\
2q_1q_2 + 2q_3q_0 \\
2q_2q_3 - 2q_0q_1
\end{pmatrix}
\]

A Quaternion can record a rotation transformation by an angle \( \theta \) about an axis \( \mathbf{n} \) like this:

\[
[R(\theta, \mathbf{n})] = [I(q, \mathbf{n})]
\]

where:
\[
q = \cos\left(\frac{\theta}{2}\right)
\]
\[
\mathbf{n} = \sin\left(\frac{\theta}{2}\right)
\]

Concated Quaternion Rotations are handled like this:

\[
R(\theta, \mathbf{n}) = R_2(\theta_2, \mathbf{n}_2)R_1(\theta_1, \mathbf{n}_1)
\]

A rotation takes effect first, followed by \( R_2 \).

A Quaternion can represent a point \( P \) like this:

\[
P' = \begin{pmatrix}
q_0^+ + q_1^+ - q_2^- + q_3^- \\
q_0^+ - q_1^- + q_2^+ - q_3^+ \\
2q_1q_2 + 2q_3q_0 \\
2q_2q_3 - 2q_0q_1
\end{pmatrix}
\]

A rotated point, \( P' \) by multiple rotations is:

\[
P' = \begin{pmatrix}
q_0^+ + q_1^+ - q_2^- + q_3^- \\
q_0^+ - q_1^- + q_2^+ - q_3^+ \\
2q_1q_2 + 2q_3q_0 \\
2q_2q_3 - 2q_0q_1
\end{pmatrix}
\]

A Quaternion can record a rotation transformation by an angle \( \theta \) about an axis \( \mathbf{n} \) like this:

\[
[R(\theta, \mathbf{n})] = [I(q, \mathbf{n})]
\]

where:
\[
q = \cos\left(\frac{\theta}{2}\right)
\]
\[
\mathbf{n} = \sin\left(\frac{\theta}{2}\right)
\]

Concated Quaternion Rotations are handled like this:

\[
R(\theta, \mathbf{n}) = R_2(\theta_2, \mathbf{n}_2)R_1(\theta_1, \mathbf{n}_1)
\]

A rotation takes effect first, followed by \( R_2 \).

A Quaternion can represent a point \( P \) like this:

\[
P' = \begin{pmatrix}
q_0^+ + q_1^+ - q_2^- + q_3^- \\
q_0^+ - q_1^- + q_2^+ - q_3^+ \\
2q_1q_2 + 2q_3q_0 \\
2q_2q_3 - 2q_0q_1
\end{pmatrix}
\]

A rotated point, \( P' \) by multiple rotations is:

\[
P' = \begin{pmatrix}
q_0^+ + q_1^+ - q_2^- + q_3^- \\
q_0^+ - q_1^- + q_2^+ - q_3^+ \\
2q_1q_2 + 2q_3q_0 \\
2q_2q_3 - 2q_0q_1
\end{pmatrix}
\]
Converting from a Matrix to a Quaternion

\[
R - R' = \begin{bmatrix}
0 & -2a, \sin \theta & +2c, \sin \theta & 0 \\
+2a, \sin \theta & 0 & -2c, \sin \theta & 0 \\
-2c, \sin \theta & +2a, \sin \theta & 0 & 0 \\
0 & c & 0 & -a \\
\end{bmatrix}
\]

If we let
\[
d = \sqrt{a^2 + b^2 + c^2}, \quad n = \left( \frac{a}{d}, \frac{b}{d}, \frac{c}{d}, -\frac{d}{a} \right)
\]

\[
q_0 = \cos\left(\frac{\theta}{2}\right)
\]

\[
q = \sin\left(\frac{\theta}{2}\right) n
\]

My Own Code (Quat.h, Quat.cpp)

```cpp
float dt = 1. / (float)( N - 1 );
const int N = 10;
// try interpolating from r1 to r5:
fprintf( stderr, "Transformed point = %s\n", p2.toString() );
fprintf( stderr, "Original point    = %s\n", p1.toString() );
Point p2 = r4 * p1;
Point p1 = Point( 1., 1., 0. );
r5.printMatrix();
fprintf( stderr, "r3*r2*r1 matrix =\n" );
fprintf( stderr, "r3       = %s\n", r3.toString() );
fprintf( stderr, "r2*r1    = %s\n", r4.toString() );
fprintf( stderr, "r2       = %s\n", r2.toString() );
fprintf( stderr, "r1       = %s\n", r1.toString() );
```

Quaternions in GLM

```cpp
#include "glmnew2.hpp"
#include "glmnew3.hpp"
#include "glmnew4.hpp"
#include "glmnew5.hpp"
#include "glmnew6.hpp"
#include "glmnew7.hpp"
#include "glmnew8.hpp"
#include "glmnew9.hpp"
```

Going Back and Forth between OpenGl/Vulkan and Quaternions using GLM

```cpp
glm::quat q3 = quaternion::slerp( q0, q1, t );
glm::vec4 v' = glm::vec4( v, 0.0 );
```

My Code (Quat.h, Quat.cpp)

```cpp
return r;
r = p;
cq = sin( t *angr ) / s;
cp = sin( (1.-t)*angr ) / s;
// do the spherical interpolation:
if( s == 0. )
s = sin( angr );
angr = acos( c );
c = p.s*q.s + p.vx*q.vx + p.vy*q.vy + p.vz*q.vz;
// dot product to get the angle between p and q:
```
The GLM Quaternion \textit{mix( )} Method is actually Spherical Interpolation:

\[ Q'(t) = \frac{\sin((1-t)\phi)}{\sin\phi} P + \frac{\sin t\phi}{\sin\phi} Q \]

where:

- \( \cos\phi = P \cdot Q = (p_0 q_0 + p_x q_x + p_y q_y + p_z q_z) \)
- \( \sin\phi = \sqrt{1 - \cos^2\phi} \)}

Andrew Hanson, *Visualizing Quaternions*, Morgan-Kaufmann, 2006.