



1


Quaternions



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



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/)

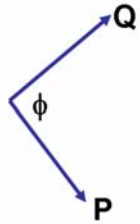


Oregon State
University
Computer Graphics

Quaternions.pptx mjb - February 21, 2018

2

A Useful Concept: Spherical Linear Interpolation




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

$$0 \leq t \leq 1.$$

where:

$$\cos \phi = P \cdot Q = (p_x q_x + p_y q_y + p_z q_z)$$

$$\sin \phi = \sqrt{1 - \cos^2 \phi}$$



Oregon State
University
Computer Graphics

mjb - February 21, 2018

A Review of Complex Numbers

3

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

$$z_1 z_2 = (x_1 + i y_1)(x_2 + i y_2) = r_1 e^{i\theta_1} r_2 e^{i\theta_2} = r_1 r_2 e^{i(\theta_1 + \theta_2)}$$

(i.e., multiply the r's and add the θ 's)

Since we are adding the θ 's, we see that complex multiplication is really just rotation by θ_2 if $r_2 = 1$.

Complex conjugate:

$$z^* = r e^{-i\theta} = x - iy$$

Complex inverse:

$$z^{-1} = \frac{1}{r} e^{-i\theta}$$

Note:

- $(z)(z^*) = r^2$
- $(z)(z^{-1}) = 1$.
- If $r = 1$., then $z^* = z^{-1}$



mjb - February 21, 2018

Quaternion Background

4

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.

Good thing spray-paint hadn't been invented yet...

Quaternions have 4 elements, one real and three complex:

$$Q = q_0 + i q_1 + j q_2 + k q_3 = (q_0, q_1, q_2, q_3) = (q_0, \bar{q})$$

By definition, 1 $ii = jj = kk = -1$ 2 $ij = k, jk = i, ki = j$

3 $ji = -k, kj = -i, ik = -j$ 4 $ijk = -1$



Oregon State University Computer Graphics

And, by definition, we always force $\|Q\| = 1$. by making $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$

mjb - February 21, 2018

Quaternion Multiplication


5

$$PQ = (p_0 + ip_1 + jp_2 + kp_3)(q_0 + iq_1 + jq_2 + kq_3)$$

$$-\bar{p} \cdot \bar{q}$$

$$PQ = \begin{pmatrix} p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ p_1q_0 + p_0q_1 + p_2q_3 - p_3q_2 \\ p_2q_0 + p_0q_2 + p_3q_1 - p_1q_3 \\ p_3q_0 + p_0q_3 + p_1q_2 - p_2q_1 \end{pmatrix} \begin{matrix} i \\ j \\ k \end{matrix}$$

$q_0\bar{p}$ $p_0\bar{q}$ $\bar{p} \times \bar{q}$



mjb - February 21, 2018

Performing Rotations with Quaternions

6

A Quaternion can record a rotation transformation by an angle θ about an axis \hat{n} like this:

$$[R(\theta, \hat{n})] = [Q(q_0, \bar{q})]$$

where:

$$q_0 = \cos(\theta/2)$$

$$\bar{q} = \sin(\theta/2)\hat{n}$$


Concatenated Quaternion Rotations are handled like this:

$$R'(\theta, \hat{n}) = R_2(\theta_2, \hat{n}_2) * R_1(\theta_1, \hat{n}_1) \quad \leftarrow R_1\text{'s rotation takes effect first, followed by } R_2\text{'s}$$

A Quaternion can represent a point P like this:

$$P = Q(0, \bar{p}) = (0, p_x, p_y, p_z)$$

A rotated point, P' by one rotation is: **A rotated point, P' by multiple rotations is:**

$$P' = [R] * \{P\} * [R]^{-1} \quad \quad P' = [R_2][R_1] * \{P\} * [R_1]^{-1}[R_2]^{-1}$$


mjb - February 21, 2018

Converting from a Quaternion to a Matrix

7

$$R(Q) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

From the GLM Notes:

8

The Rotation Matrix for an Angle (θ) about an Arbitrary Axis (A_x, A_y, A_z)

$$[M] = \begin{bmatrix} A_x A_x + \cos \theta (1 - A_x A_x) & A_x A_y - \cos \theta (A_x A_y) - \sin \theta A_z & A_x A_z - \cos \theta (A_x A_z) + \sin \theta A_y \\ A_y A_x - \cos \theta (A_y A_x) + \sin \theta A_z & A_y A_y + \cos \theta (1 - A_y A_y) & A_y A_z - \cos \theta (A_y A_z) - \sin \theta A_x \\ A_z A_x - \cos \theta (A_z A_x) - \sin \theta A_y & A_z A_y - \cos \theta (A_z A_y) + \sin \theta A_x & A_z A_z + \cos \theta (1 - A_z A_z) \end{bmatrix}$$

For this to be correct, A must be a unit vector

Converting from a Quaternion to a Matrix

9

$$R(Q) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

$$R = \begin{bmatrix} c + n_x^2(1 - c) & n_x n_y(1 - c) - s n_z & n_x n_y(1 - c) + s n_z \\ n_y n_x(1 - c) + s n_z & c + n_y^2(1 - c) & n_y n_z(1 - c) - s n_x \\ n_z n_x(1 - c) - s n_y & n_z n_y(1 - c) + s n_x & c + n_z^2(1 - c) \end{bmatrix}$$

where: $c = \cos\theta$
 $s = \sin\theta$

Note that the sum of the Trace (diagonal) elements is: $3c + (1 - c) = 1 + 2\cos\theta$



mjb - February 21, 2018

Converting from a Matrix to a Quaternion

10

$$R(Q) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

Note that the sum of the Trace (diagonal) elements is: $3c + (1 - c) = 1 + 2\cos\theta$

Letting $t = \text{Trace}(R) = 1 + 2\cos\theta$, then:

$$\cos\theta = \frac{1}{2}(t - 1)$$

$$\sin\theta = \sqrt{1 - \cos^2\theta}$$



mjb - February 21, 2018

Notes from Converting from a Quaternion to a Matrix

11

$$R = \begin{bmatrix} c + n_x^2(1-c) & n_x n_y(1-c) - s n_z & n_x n_y(1-c) + s n_y \\ n_y n_x(1-c) + s n_z & c + n_y^2(1-c) & n_y n_z(1-c) - s n_x \\ n_z n_x(1-c) - s n_y & n_z n_y(1-c) + s n_x & c + n_z^2(1-c) \end{bmatrix}$$

$$R^T = \begin{bmatrix} c + n_x^2(1-c) & n_y n_x(1-c) + s n_z & n_z n_x(1-c) - s n_y \\ n_x n_y(1-c) - s n_z & c + n_y^2(1-c) & n_z n_y(1-c) + s n_x \\ n_x n_y(1-c) + s n_y & n_y n_z(1-c) - s n_x & c + n_z^2(1-c) \end{bmatrix}$$

$$R - R^T = \begin{bmatrix} 0 & -2s n_z & +2s n_y \\ +2s n_z & 0 & -2s n_x \\ -2s n_y & +2s n_x & 0 \end{bmatrix}$$



$$c = \cos\left(\frac{\theta}{2}\right)$$

$$s = \sin\left(\frac{\theta}{2}\right)$$

mjb - February 21, 2018

Converting from a Matrix to a Quaternion

12

$$R - R^T = \begin{bmatrix} 0 & -2n_z \sin \theta & +2n_y \sin \theta \\ +2n_z \sin \theta & 0 & -2n_x \sin \theta \\ -2n_y \sin \theta & +2n_x \sin \theta & 0 \end{bmatrix} = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix}$$

If we let

$$d = \sqrt{a^2 + b^2 + c^2}, \text{ then } \hat{n} = \left(\frac{a}{d}, \frac{b}{d}, \frac{c}{d} \right)$$

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

$$\bar{q} = \sin\left(\frac{\theta}{2}\right)\hat{n}$$



mjb - February 21, 2018

Quaternions in GLM

13

```

#include "glm/vec2.hpp"
#include "glm/vec3.hpp"
#include "glm/mat4x4.hpp"
#include "glm/gtc/matrix_transform.hpp"
#include "glm/gtc/matrix_inverse.hpp"
#include "glm/gtc/quaternion.hpp"
#include "glm/gtx/quaternion.hpp"

glm::quat rot1 = glm::angleAxis( glm::radians(45.f), glm::vec3( 0.707f, 0.707f, 0. ) );
glm::quat rot2 = glm::angleAxis( glm::radians(90.f), glm::vec3( 1. , 0. , 0. ) );

glm::quat combinedRots = rot2 * rot1;
glm::vec4 v = glm::vec4( 1., 1., 1., 1. );
glm::vec4 vp = combinedRots * v;

glm::mat4 rotMatrix = glm::toMat4( combinedRots );
glm::vec4 vpp = rotMatrix * v; // same result as vp

```

Converting from a Quaternion to OpenGL

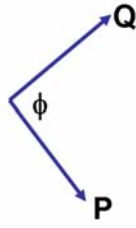
14

```
glRotate3f(  $\theta^\circ$ , nx, ny, nz );
```

```
glm::rotate( glm::quat const & q,  $\theta^R$ , glm::vec3( nx, ny, nz ) );
```

A Useful Concept: Spherical Linear Interpolation, where P and Q are Quaternions

15



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

$$0 \leq t \leq 1.$$

where:

$$\cos \phi = P \cdot Q = (p_0q_0 + p_xq_x + p_yq_y + p_zq_z)$$

$$\sin \phi = \sqrt{1 - \cos^2 \phi}$$



mjb - February 21, 2018

My Code (Quat.h, Quat.cpp)

16

```

Rotate
Slerp( float t, const Quat& p, const Quat& q )
{
    float angr;           // angle between p and q in radians
    float c, s;           // cosine and sine of the angle between p and q
    float cp, cq;        // coefficients to multiply quaternions p and q
    Rotate r;

    // dot product to get the angle between p and q:
    c = p.s*q.s + p.vx*q.vx + p.vy*q.vy + p.vz*q.vz;
    angr = acos( c );

    // sine of that angle:
    s = sin( angr );

    // if the sine is 0., then p == q:
    if( s == 0. )
    {
        r = p;
        return r;
    }

    // do spherical interpolation:
    cp = sin( (1.-t)*angr ) / s;
    cq = sin( t*angr ) / s;

    r = cp*p + cq*q;
    return r;
}

```



mjb - February 21, 2018

17

My Code (Quat.h, Quat.cpp)

```

int
main( int argc, char *argv[ ] )
{
    Rotate r1 = Rotate( 45.*D2R, 0., 0., 1. );
    Rotate r2 = Rotate( 90.*D2R, 0., 0., 1. );
    Rotate r3 = Rotate( 90.*D2R, 1., 0., 0. );
    Rotate r4 = r2 * r1;
    Rotate r5 = r3 * r2 * r1;

    fprintf( stderr, "r1   = %s\n", r1.toString() );
    fprintf( stderr, "r2   = %s\n", r2.toString() );
    fprintf( stderr, "r2*r1 = %s\n", r4.toString() );


    fprintf( stderr, "r3   = %s\n", r3.toString() );
    fprintf( stderr, "r3*r2*r1 = %s\n", r5.toString() );
    fprintf( stderr, "\nr3*r2*r1 matrix =\n" );
    r5.printMatrix();

    Point p1 = Point( 1., 1., 0. );
    Point p2 = r4 * p1;

    fprintf( stderr, "Original point = %s\n", p1.toString() );
    fprintf( stderr, "Transformed point = %s\n", p2.toString() );

    // try interpolating from r1 to r5:
    const int N = 10;
    float dt = 1. / (float)( N - 1 );
    float t = 0.;
    for( int i = 0; i < N; i++, t += dt )
    {
        Rotate r15 = Slerp( t, r1, r5 );
        fprintf( stderr, "%2d %5.3f %s\n", i, t, r15.toString() );
    }
}

```



mjb - February 21, 2018

18

My Code (Quat.h, Quat.cpp)

```


r1   = 45.000 degrees about ( 0.000 0.000 1.000 )
r2   = 90.000 degrees about ( 0.000 0.000 1.000 )
r2*r1 = 135.000 degrees about ( 0.000 0.000 1.000 )
r3   = 90.000 degrees about ( 1.000 0.000 0.000 )
r3*r2*r1 = 148.606 degrees about ( 0.281 -0.678 0.679 )

r3*r2*r1 matrix =
-0.707 -0.707 0.000 0.000
 0.000 0.000 -1.000 0.000
 0.707 -0.707 0.000 0.000
 0.000 0.000 0.000 1.000

Original point = 1.000 1.000 0.000
Transformed point = -1.414 0.001 0.000

0 0.000 45.000 degrees about ( 0.000 0.000 1.000 )
1 0.111 53.754 degrees about ( 0.080 -0.194 0.978 )
2 0.222 64.007 degrees about ( 0.136 -0.328 0.935 )
3 0.333 75.145 degrees about ( 0.175 -0.423 0.889 )
4 0.444 86.824 degrees about ( 0.204 -0.493 0.846 )
5 0.556 98.848 degrees about ( 0.226 -0.546 0.807 )
6 0.667 111.101 degrees about ( 0.244 -0.588 0.771 )
7 0.778 123.508 degrees about ( 0.258 -0.623 0.739 )
8 0.889 136.021 degrees about ( 0.270 -0.652 0.708 )
9 1.000 148.606 degrees about ( 0.281 -0.678 0.679 )

```



mjb - February 21, 2018

A Really Good (i.e., Complete) Reference

19

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



mjb - February 21, 2018