

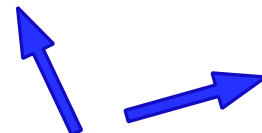
CS770/870 Spring 2017

Quaternions

- Primary resources used in preparing these notes:
 1. van Osten, *3D Game Engine Programming: Understanding Quaternions*,
<https://www.3dgep.com/understanding-quaternions>
 2. Barbic, *Quaternions and Rotations*,
<http://run.usc.edu/cs520-s12/quaternions/quaternions-cs520.pdf>
 3. OpenGL Tutorials: Tutorial 17: Rotations,
<http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/>
 4. Ibanez, *Tutorial on Quaternions: Part I*,
<https://itk.org/CourseWare/Training/QuaternionsI.pdf>
 5. Dam et al. *Quaternions, Interpolation and Animation*,
<http://web.mit.edu/2.998/www/QuaternionReport1.pdf>

Rotation Specification Problem

- Objects in space have *states* that are changed by operations
 - *location* changed by translation
 - *size* changed by scale
 - *orientation* changed by rotation
- Given start/end *location* and *size*, trivial to determine parameters for translation and scale
- What about *orientation*?
 - In 3D it's especially hard



Remember this!

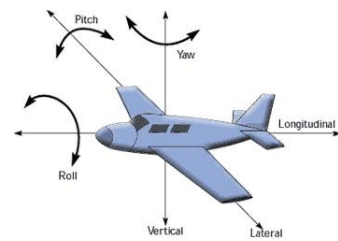
Rotation about Arbitrary Axis

- Rotate α degrees about an axis defined by a point (c_x, c_y, c_z) and a *direction*, $d=(d_x, d_y, d_z)$
 - Translate (c_x, c_y, c_z) to origin: $T=T(-c_x, -c_y, -c_z)$
 - Rotate about x so that (d_x, d_y, d_z) lies in xy plane
 - Rotate about z so that d' (transformed d) lies along y axis
 - Rotate α degrees about y axis: $R_y(\alpha)$
 - Apply inverse of z rotation
 - Apply inverse of x rotation
 - Apply inverse of translation: $T^{-1}=T(c_x, c_y, c_z)$

Composite transformation: $R = T^{-1}R_x^{-1}R_z^{-1}R_y(\alpha)R_zR_xT$

Rotation Specification Options

- Matrices (as in previous slide)
 - comprehensive, but tedious
- Euler angles (rotations about x, y, z axes)
 - *pitch* (x), *yaw* (y), and *roll* (z) angles
 - Compose in what order?
 - unique: xyz, xzy, yxz, yzx, zxy, zyx
 - reuse: xyx, xzx, yxy, yzy, zxz, zyz
 - can also use fixed axes, or rotated axes
 - that's 24 different versions!
- Quaternions



Quaternions Made Usable

- Discovered by Irish mathematician, Sir William Rowan Hamilton in 1843.
- Let rotationAxis = $[r_x, r_y, r_z]$, α = angle
- quaternion = $[x, y, z, w]$, where

$$x = r_x * \sin(\alpha / 2)$$

$$y = r_y * \sin(\alpha / 2)$$

$$z = r_z * \sin(\alpha / 2)$$

$$w = \cos(\alpha / 2)$$

A little Quaternion algebra

- Lots of useful operators and operations

Let $q = [v, w]$, where $v = [x, y, z]$

$$\text{length}(q) = |q| = \sqrt{w^2 + v \cdot v'}$$

$$q + q' = [v + v', w + w']$$

$$q * q' = qq' = [v \times v' + ww' + w'v, ww' - v \cdot v']$$

where $\times$ is vector cross product

Same as rotation composition (matrix multiplication)!

$qq' \neq q'q$ quaternion multiplication is **not** commutative

but rotations aren't either!

A few JOML Quaternion Methods

- Matrix* methods

```
set(Quaternionf q); // set mat to q rotation  
rotate(Quaternionf q); // multiply by q rot.
```

- Quaternion* methods

```
rotationTo(Vector3f from, Vector3f to);  
    // Quaternion that rotates 'from' -> 'to'  
rotate( ... ); // rotate this Quaternion by args
```

Quaternion Advantages

- Simple composition
- Obvious geometric interpretation (axis/angle)
 - simple to define high level rotations: map one direction to another
- Simple interpolation methods
- Compact representation (vs matrix)
- No *gimbal lock* (suffered by Euler angles)

Motivation for Quaternions

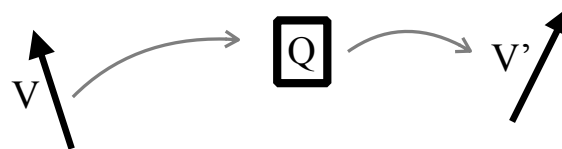
- Consider the relationship between a point, P and a vector, V:



- Can “apply” V to P to get a “new” P, P’



- Can apply a Quaternion to a Vector to get a “new” Vector:



- Vector has length and orientation; a Quaternion must represent a *relative* length and a *relative* orientation.
- A **Versor** is a quaternion of length 1; it is a pure rotation.

Overview of Quaternion Definition

- Motivated by complex numbers

$$c = a + bi \text{ where } i = \sqrt{-1}$$

$$q = xi + yj + zk + w \text{ where } i=j=k=\sqrt{-1}$$

- Complex numbers can be modeled in 2D space
x-axis is real part; y-axis is imaginary part
- Quaternions can be modeled in 4D
xyz-axes is vector part; w-axis is angle part
- Restriction to unit length quaternions reduces (somewhat) the 4D complexity

Complex Number Reminders

- Complex number: $c = a + bi$ where $i^2 = -1$

$$c_1 + c_2 = (a_1 + a_2) + (b_1 + b_2)i$$

$$\begin{aligned} c_1 * c_2 &= c_1 c_2 = (a_1 + b_1 i)(a_2 + b_2 i) = a_1 a_2 + a_1 b_2 i + a_2 b_1 i - a_1 b_2 \\ &= (a_1 a_2 - b_1 b_2) + (a_1 a_2 + b_1 b_2)i \end{aligned}$$

$$c^2 = (a^2 - b^2) + 2abi$$

$$\text{Conjugate}(c) = c^* = a - bi$$

$$\text{norm}(c) = \text{length}(c) = |c| = \sqrt{c c^*} = \sqrt{a^2 + b^2}$$

$$\begin{aligned} c_1 / c_2 &= (c_1 c_2^*) / (c_2 c_2^*) i = (a_1 a_2 + b_1 b_2) / (a_1^2 + a_2^2) \\ &\quad + (b_1 a_2 - a_1 b_2) / (a_1^2 + a_2^2) \end{aligned}$$

and some more

Quaternion Basics

$$q = xi + yj + zk + w \text{ where } i=j=k = \sqrt{-1}$$

$$i^2 = j^2 = k^2 = ijk = -1$$

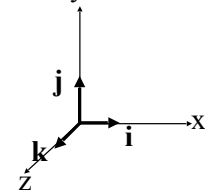
$$qq' = [v \times v' + wv' + w'v, ww' - v \cdot v']$$

Express i and j as *pure* quaternions ($w=0$)

$$ij = [i \times j, -i \cdot j], i \cdot j = 0 \text{ and } i \times j = k$$

Similarly, $jk=i$ and $ki=j$

and $ji=-k$, $kj=-i$ and $ik=-j$ based on cross product



Quaternion Operations

- Addition/subtract: $q+q' = [\mathbf{v}+\mathbf{v}', w+w']$, $q-q' = [\mathbf{v}-\mathbf{v}', w-w']$
- Product: $qq' = [\mathbf{v} \times \mathbf{v}' + w\mathbf{v}' + w'\mathbf{v}, ww' - \mathbf{v} \cdot \mathbf{v}']$
 $qq' = [(wx' + w'x + yz' - y'z)\mathbf{i}, 0] + [(wy' + w'y + zx' - z'x)\mathbf{j}, 0]$
 $+ [(wz' + w'z + xy' - x'y)\mathbf{k}, 0] + [0, 0, 0, (ww' - xx' - yy' - zz')]$
- Multiply by scalar: $sq = [s\mathbf{v}, sw]$
- *real* quaternion: $r = [(0, 0, 0), w]$ # no vector component
- *pure* quaternion: $r = [\mathbf{v}, 0]$ # 0 angle
- Conjugate, $q^* = [\mathbf{v}, -w]$, so $qq^* = [|\mathbf{v}|^2 + w^2]$ and $|q| = \sqrt{qq^*}$
- Normalized quaternion = $q/|q|$
- Inverse: $q^{-1} = q^*/|q|^2$
- Dot: $q \bullet q' = xx' + yy' + zz' + ww' = \cos \alpha$, angle between q and q'
if $|q| = |q'| = 1$