# CSc-165

## 16 - Quaternions

## Representing Orientation / Rotation

Angle / Axis

Euler Angles

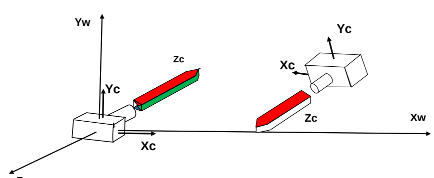
- stored as homogenous 4x4 matrices
- simple to understand, easy to combine
- vulnerable to "gimbal lock"
  *https://www.youtube.com/watch?v=zc8b2Jo7mno*

Quaternions

- stored as a 1x4 vector (more compact)
- complex to understand, but easy to combine
- scalar + 1x3 vector (imaginary component)
- not vulnerable to "gimbal lock"
- easier to "interpolate" for smooth rotations

## Orientation as a *Rotation Vector*

- **Consider the "look-at" vector (Zc)**
  - **Imagine Zc has "orientation"** ***<u>about its direction</u>***
  - **Then,** ***<u>camera orientation change</u>*** **can be considered as** ***<u>"transforming the Zc 'vector'"</u>***




3

## Quaternions

**A four-element object that represents a "<u>3D orientation</u>"** **William Hamilton (1805-1865)**

$$q = (w, x, y, z) = (w, \vec{v}), \quad \text{where } \vec{v} = [x\ y\ z]$$

- $w$ represents "rotation angle"
- $\vec{v}$ represents "rotation axis"

But only if `magnitude(q)` = 1

- `magnitude(q) = |q| = sqrt (w² + x² + y² + z²)`

4

## Orientations as Quaternions

- **Any rotation "r" of amount $\alpha$ about an axis `v = [x y z]` can be represented as a quaternion**

$$q_r = \left(\cos(\alpha/2),\ \ \sin(\alpha/2)\vec{v}\right)$$
$$= \left(\cos(\alpha/2),\ \ [x \sin(\alpha/2),\ \ y \sin(\alpha/2),\ \ z \sin(\alpha/2)]\right)$$

- **Multiplying two (unit) quaternions is the same as concatenation of rotation transformations!**

$q_1 = (w_1, x_1, y_1, z_1)$ *// an orientation*
$q_2 = (w_2, x_2, y_2, z_2)$ *// another orientation*
$q_3 = q_2 * q_1$ *// combined orientation*

5

## Vectors as Quaternions

- **Any vector v = [x y z] can be represented as a quaternion:**

$q_v$ `= (0,[x y z]) = (0, x, y, z)`

- **Any "quaternion vector"** $q_v$ **can be transformed by a "rotation quaternion"** $q_r$

$q_v' = q_r * q_v * \text{conjugate}(q_r)$

6

## Quaternion to Angle/Axis

- Given a quaternion

```
q = ( w, [qx, qy, qz] )
```

- The corresponding angle/axis rotation is

```
angle α = 2 * arccos(w)
 xAxis = qx / sin(α/2)
 yAxis = qy / sin(α/2)
 zAxis = qz / sin(α/2)
```



## Quaternion to Angle/Axis (cont.)

- Note the potential for *divide-by-zero*

```
xAxis = qx / sin(α/2)
yAxis = qy / sin(α/2)
zAxis = qz / sin(α/2)
```

- Occurs when $\alpha = 0$ (or 360)
- Recall that *if $\alpha = 0$*, axis doesn't matter

```
denom = sin(alpha/2);
if (abs(denom) < 0.0001 ) {
    denom = 1 ;
}
xAxis = qx / denom;
yAxis = qy / denom;
zAxis = qx / denom;
```



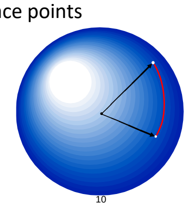
## Orientation Interpolation

- Complicated when using Euler, UVN, or Angle/Axis
  - Many variables → many paths
  - How to choose one?
- Quaternions provide a simple, unique interpolation
  - Two approaches:
    - Linear ("*lerp*")
    - Spherical linear ("*slerp*")



## Quaternion Interpolation

- Orientation quaternions represent radius vectors of a unit sphere
  - Actually, *infinitely many* unit spheres
- *Interpolation* = finding quaternions along the arc between surface points





## Linear Interpolation ("*lerp*")

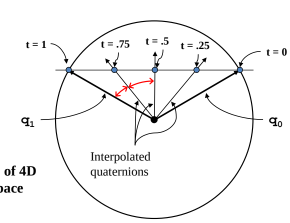
- Given:
  - two quaternions $\mathbf{q_0}$ and $\mathbf{q_1}$
  - a parameter $t$ $(0 \le t \le 1)$,

$$lerp(q_0, q_1, t) = (1-t)q_0 + tq_1$$



## Drawback of *lerp*

- Uniform parametric changes don't produce uniform angle changes




**2D representation of 4D unit quaternion space**

## Spherical Linear Interpolation ("*slerp*")

- Given:
  - two quaternions $\mathbf{q}_0$ and $\mathbf{q}_1$
  - a parameter $t$ $(0 \le t \le 1)$
  - angle $\theta$ between $\mathbf{q}_0$ and $\mathbf{q}_1$ = *arccos*($q_0 \bullet q_1$)

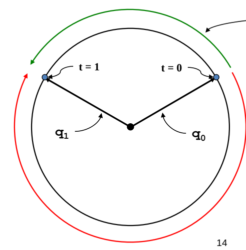
$$slerp(q_0, q_1, t) = \frac{q_0 \sin((1-t)\theta) + q_1 \sin(t\theta)}{\sin(\theta)} \qquad [1]$$

[1] Watt, Alan, 3D Computer Graphics, 3rd ed., p. 490-491



## A Problem with *Slerp*

- There are always *two arcs around the sphere…*




*Slerp* will take this path when $(q_0 \bullet q_1) > 0$

*Slerp* will take this path when $(q_0 \bullet q_1) < 0$

Solution:

```
if ((q0•q1)<0)
{
  q1 = -q1;
}
```



## Another Problem with *Slerp*

- Doesn't work well for very small angles
  - What happens in the limit – i.e. with the smallest possible angle ?
- Solution:

```
if ( abs(θ) < epsilon )
  use lerp()
else
  use slerp()
```