Quaternion

CSE 781

Han-Wei Shen

Winter 2010

3D Rotations with Euler Angles

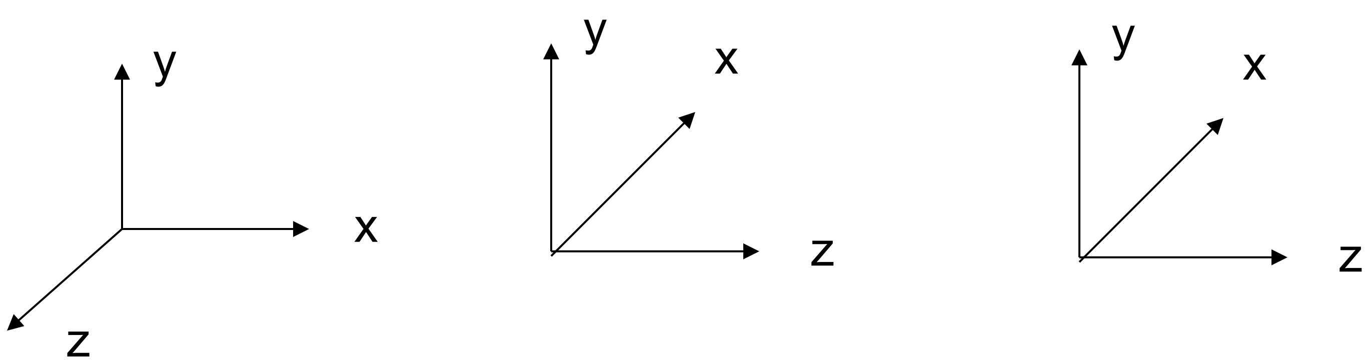
- A simple but non-intuitive method – specify separate x, y, z axis rotation angles based on the mouse's horizontal, vertical, and diagonal movements

OpenGL - glRotatef(θ, 0,0,1) glRotatef(θ, 0,1,0) glRotatef(θ, 1,0,0)

$$\begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad \begin{vmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Euler Rotation Problems

- Gimbal Lock – lose one degree of freedom
- Problem happens when the axes of rotation line up on top of each other. For example:

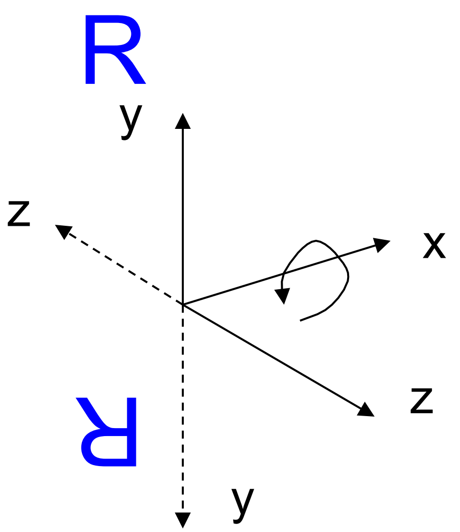


Initially | Rotate(90, 0,1,0) | Rotate(??, 0,0,1)

This is same as x rotation !!

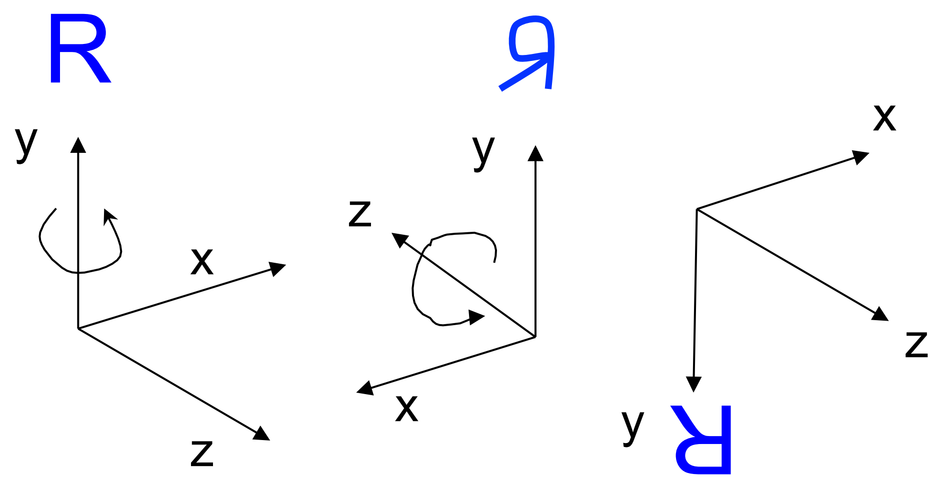
Euler Rotation Problems

- Rotations with Euler angles to change from one orientation to another are not unique. Example: (x,y,z) rotation to achieve the following:



Rotate(180, 1,0,0)
Euler angles: (0,0,0) -> (180,0,0)

OR



Rotate(180, 0,1,0) then Rotate(180,0,0,1)
Euler angles: (0,0,0) -> (0,180,180)

Quaternion

- Invented in 1843 as an extension to the complex numbers
- Used by computer graphics since 1985
- Quaternion:
  - Provide an alternative method to specify rotation
  - Can avoid the gimbal lock problem
  - Allow unique, smooth and continuous rotation interpolations

Mathematical Background

- A quaternion is a 4-tuple of real number, which can be seen as a vector and a scalar

  $Q = [q_x, q_y, q_z, q_w] = \mathbf{q}_v + q_w$, where

  q_w is the real part and

  $\mathbf{q}_v = iq_x + jq_y + kq_z = (q_x, q_y, q_z)$ is the imaginary part

- i*i = j*j = k*k = -1;
- j*k= -k*j= i; k*i=-i*k=j; i*j=-j*i= k;
- All the regular vector operations (dot product, cross product, scalar product, addition, etc) can be applied to the imaginary part $\mathbf{q}_v$

Basic Operations

- Multiplication: $QR = (\mathbf{q}_v \times \mathbf{r}_v + r_w\mathbf{q}_v + q_w\mathbf{r}_v,$
  $q_w r_w - \mathbf{q}_v.\mathbf{r}_v)$
- Addition: $Q+R = (\mathbf{q}_v+\mathbf{r}_v, q_w+r_w)$
- Conjugate: $Q^* = (-q_v, q_w)$
- Norm (magnitude) = $QQ^* = Q^*Q = q_x{}^*q_x+q_y{}^*q_y+q_z{}^*q_z+q_w{}^*q_w$
- Identity $i = (\mathbf{0},1)$
- Inverse $Q_{-1} = (1/ \text{Norm}(Q))\, Q^*$
- Some more rules can be found in the reference book (real time rendering) pp46

Imaginary

real

Polar Representation

- Remember a 2D unit complex number

  $\cos\theta + i \sin\theta = e^{i\theta}$

- A unit quaternion Q may be written as:

  $Q = (\sin\phi\ \mathbf{u_q}\ ,\ \cos\phi) = \cos\phi + \sin\phi\ \mathbf{u_q}$, where $\mathbf{u_q}$ is a *unit* 3-tuple vector

- We can also write this unit quaternion as:

  $Q = e^{\mathbf{u_q}\phi}$

Quaternion Rotation

- A rotation can be represented by a unit quaternion $Q = (\sin\phi\, \mathbf{u_q}, \cos\phi)$
  - Given a point p = (x,y,z) -> we first convert it to a quaternion $p' = ix + jy + kz + 0 = (p_v, 0)$
  - Then, $Qp'Q^{-1}$ is in fact a rotation of p around $\mathbf{u_q}$ by an angle 2ϕ !!

Rotation Concatenation

- Concatenation is easy – just multiply all the quaternions Q_1, Q_2, Q_3, …. Together

$$(Q_3\ (Q_2\ (\ Q_1\ \ P'\ \ Q1^{-1}\ )\ Q2^{-1}\ )\ Q3^{-1}\ )\ =$$

$$(Q3*Q2*Q1)\ P'\ (Q1^{-1}*Q2^{-1}*Q3^{-1})$$

- There is a one-to-one mapping between a quaternion rotation and 4x4 rotation matrix.

Quaternion to Rotation Matrix

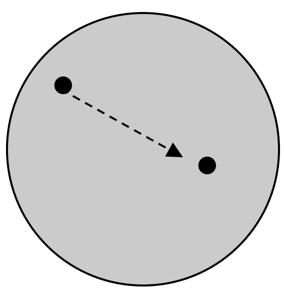
- Given a quaternion w + xi +yj + kz, it can be translated to the rotation matrix R:

$$R = \begin{bmatrix} 1-2y^2-2z^2 & 2xy+2wz & 2xz-2wy \\ 2xy-2wz & 1-2x^2-2z^2 & 2yz+2wx \\ 2xz+2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix}$$

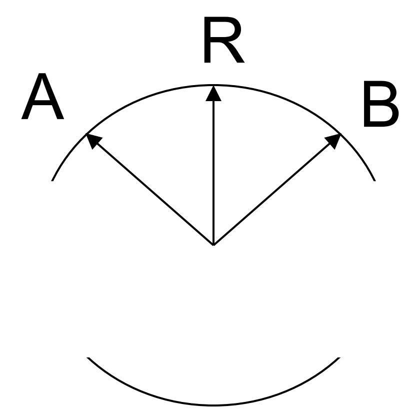
- Also you can convert a matrix to quaternion (see the reference book for detail)

Interpolation of Rotation

- Should avoid sudden change of orientation and also should maintain a constant angular speed
- Each rotation can be represented as a point on the surface of a 4D unit sphere
  - Need to perform smooth interpolation along this 4D sphere



How to interpolate A and B to get R?



Interpolation Rotation

- Spherical Linear Interpolation (slerp):

  Given two unit quaternion (i.e., two rotations), we can create a smooth interpolation using *slerp*:

  slerp(Q1, Q2, t) =

$$\frac{\sin(\phi(1-t))}{\sin\phi} Q1 + \frac{\sin(\phi t)}{\sin\phi} Q2$$

  where 0<=t<=1

- To compute ϕ, we can use this property:

$$\cos\phi = Q1_x Q2_x + Q1_y Q2_y + Q1_z Q2_z + Q1_w Q2_w$$