CS380: Introduction to Computer Graphics
Quaternions
Chapter 7

Min H. Kim
KAIST School of Computing

Hello World 3D

SUMMARY

Modelview matrix

- Modelview matrix (*MVM*) $E^{-1}O$
  - Describes the orientation and position of the view E^{-1} and the orientation and position of the object O with respect to the eye frame $\vec{\mathbf{e}}^t$

    $\tilde{p} = \vec{\mathbf{o}}^t \mathbf{c} = \vec{\mathbf{w}}^t O\mathbf{c} = \vec{\mathbf{e}}^t E^{-1} O\mathbf{c}$
  - The vertex shader will take these vertex data and perform the multiplication $E^{-1}O\mathbf{c}$, producing the eye coordinates used in rendering

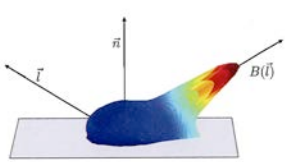
- normalMatrix() produces the inverse transpose of the linear factor to get uniform *NMVM*

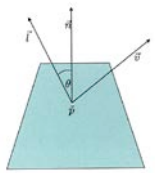
Material appearance

- Bidirectional reflectance distribution function (BRDF)
- Simple reflectance model = diffuse + specular
- The appearance change by the view vector $\vec{v}$ is managed by OpenGL

$$L_{diffuse} = R_{diffuse} \frac{\cos(\theta)}{d^2}$$

$\cos(\theta) = \vec{n} \cdot \vec{l}$





Motion callback

- Move the object

```
Matrix4 M = makeXRotation(deltay) * makeYRotation(deltax);
Matrix4 A = makeMixedFrame(objRbt, eyeRbt);
objRbt = doMtoOwrtA(M, objRbt, A);
```

- Move the eye

```
objRbt = doMtoOwrtA(inv(M), eyeRbt, A);
```

- Perform ego motion

```
objRbt = doMtoOwrtA(inv(M), eyeRbt, eyeRbt);
```

- NB For the eye motions, we invert the M so that the mouse movements produce the image movements in more desired directions.

Assignment

- **HW3 Deadline: 2015.4.2 (Thur.) 23:55**
- Go to the course website and download the files: hw2d.pdf, hw2d.zip in Week1
- Solve three tasks (described in the pdf) and submit your codes with some screenshots to the TA by email.
- **Yeong Beum Lee (yblee@vclab.kaist.ac.kr)**
- **Next lecture, we will have quiz!**

Chapter 7

QUATERNIONS

Quaternions

- The *complex numbers* could be interpreted as points in a plane, analogous to points in three-dimensional space.

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

– Not commutative:

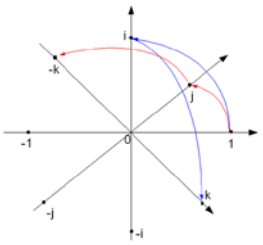
$$ij = k$$

$$ji = -k$$

$$ji = -ji$$

Quaternion multiplication

×	1	*i*	*j*	*k*
1	1	*i*	*j*	*k*
i	*i*	−1	*k*	−*j*
j	*j*	−*k*	−1	*i*
k	*k*	*j*	−*i*	−1



Graphical representation of quaternion units product as 90°-rotation in 4D-space
© wikipedia

Motivation

- For animation, we will want to interpolate between frames in a natural way.
- We will study quaternions as alternative to rotation matrices:

$$R = \begin{bmatrix} r & 0 \\ 0 & 1 \end{bmatrix}$$

- Later we will add back in the translations

Interpolation of rotation

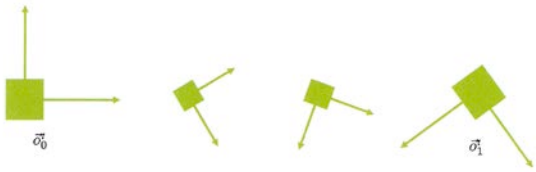
- Desired object frame rotation for "time=0" : $\vec{\mathbf{o}}_0^t = \vec{\mathbf{w}}^t R_0$
- Desired object frame rotation for "time=1" : $\vec{\mathbf{o}}_1^t = \vec{\mathbf{w}}^t R_1$
- We wish to find a sequence of frames $\vec{\mathbf{o}}_\alpha^t$ for $\alpha \in [0...1]$, that naturally rotates from $\vec{\mathbf{o}}_0^t$ to $\vec{\mathbf{o}}_1^t$

Invariance

- Invariance is a property of a class of mathematical objects that remains unchanged when transformations of a certain type are applied to the objects.
- An object flying through space with no forces acting on it has its center of mass follow a straight line.
- Its orientation spins along a fixed axis.
- This kind of orientation satisfies both left and right invariance.

Bad ideas 1

- Linear interpolation of matrices $R_\alpha := (1-\alpha)R_0 + (\alpha)R_1$ and then set $\vec{\mathbf{o}}_\alpha^t = \vec{\mathbf{w}}^t R_\alpha$
- Each basis vector simply moves along a straight line
- In this case, the intermediate R_α are not rotation matrices



Bad idea 2

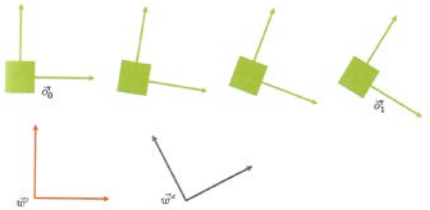
- Factor both R_0 and R_1 into 3, so-called XYZ Euler angles

$$\begin{bmatrix} k_x^2 v + c & k_x k_y v - k_z s & k_x k_z v + k_y s \\ k_y k_x v + k_z s & k_y^2 v + c & k_y k_z v - k_x s \\ k_z k_x v - k_y s & k_z k_y v + k_x s & k_z^2 v + c \end{bmatrix}$$

- These three scalar values could each be linearly interpolated using α and used to generate intermediate rotations

Bad idea 2

- Not natural,
- Not invariant to choice of the world frame
- This is called left invariance



- We need an intrinsic geometric operation (describable independent of coordinates)

What we want

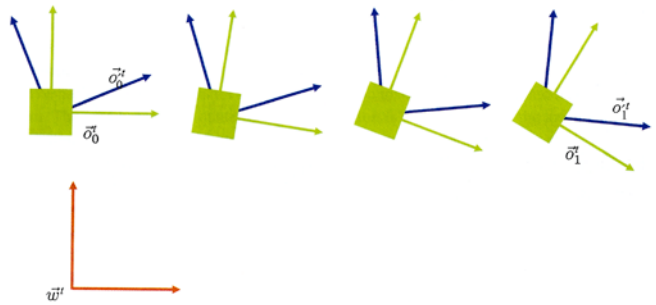
- Like to first create a single transition matrix $R_1 R_0^{-1}$
- This matrix can, as any rotation matrix, be thought of as a rotation of some θ degrees about some axis $[k_x, k_y, k_z]^t$
- Suppose we had a power operator: $(R_1 R_0^{-1})^\alpha$
  - which gave us a rotation about $[k_x, k_y, k_z]^t$ by $\alpha\theta$ degrees instead.
- Then we could set $R_\alpha := (R_1 R_0^{-1})^\alpha R_0$ and set

$$\vec{\mathbf{o}}_\alpha^t = \vec{\mathbf{w}}^t R_\alpha$$
$$\vec{\mathbf{o}}_\alpha^t = \vec{\mathbf{w}}^t (R_1 R_0^{-1})^\alpha R_0$$

Result

- This is a sequence of frames obtained by more and more rotation about a single axis
  - Read right to left
- Correct start and finish:
  $\vec{\mathbf{w}}^t (R_1 R_0^{-1})^0 R_0 = \vec{\mathbf{w}}^t R_0 = \vec{\mathbf{o}}_0$
  $\vec{\mathbf{w}}^t (R_1 R_0^{-1})^1 R_0 = \vec{\mathbf{w}}^t R_1 = \vec{\mathbf{o}}_1$
- The transition rotation fixes a unique axis
- This axis depends only on $\vec{\mathbf{o}}_0$ and $\vec{\mathbf{o}}_1$. Not any choice of world frame.
- Up to cycles (which we can uniquify soon). This gives us a unique interpolation

Result

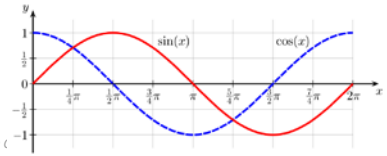


Hard part

- Hard part: factor $R_1 R_0^{-1}$ into its axis/angle form
- Main quaternion idea: is to keep track of the axis and angle at all times, but in a way that allows our manipulations.
- This will allow us to do this interpolation
- It also could help in general with avoiding numerical drift away from RBTs.

Cycles

- $R_1 R_0^{-1}$ matrix can, be thought of as a rotation of some $\theta + n2\pi$ degrees for any integer n .
- Not relevant for linear transformation on vectors, but is relevant for interpolation
- The natural choice is to choose n such that $|\theta + n2\pi|$ is minimal.
- This might result in a negative rotational angle (the other way).



The representation

- A quaternion is 4 tuple with operations
- Written: $\begin{bmatrix} \omega \\ \hat{\mathbf{c}} \end{bmatrix}$
  where ω is a scalar and $\hat{\mathbf{c}}$ is a coordinate 3-vector.
- A rotation of θ degree about a unit length axis $\hat{\mathbf{k}}$ is presented as
- Oddity: the division by 2 will be needed to make the operations work out as needed.

$$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}$$

Antipodes of quaternion (diametrically opposite to it)

- Note that a rotation of $-\theta$ degrees about the axis $-\hat{\mathbf{k}}$ gives us the same quaternion.
- A rotation of $\theta + 4\pi$ degrees about an axis $\hat{\mathbf{k}}$ also gives us the same quaternion
- A rotation of $\theta + 2\pi$ degrees about an axis $\hat{\mathbf{k}}$, which in fact is the same rotation, gives us the negated quaternion
- So antipodes represent the same rotation transformation
  - But heads up regarding cycles and power

Unit norm quats. == rotations

- Squared norm is sum of 4 squares. $\begin{bmatrix} \omega \\ \hat{\mathbf{c}} \end{bmatrix}$
- Any quaternion of the form

$$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}$$

  has a unit form
- Conversely, any such unit norm quaternion can be interpreted (along with its negation) as a unique rotation matrix.

Unit norm quats

- Identity rotation example

$$\begin{bmatrix} 1 \\ \hat{0} \end{bmatrix}, \begin{bmatrix} -1 \\ \hat{0} \end{bmatrix}$$

- Flip rotation example

$$\begin{bmatrix} 0 \\ \hat{k} \end{bmatrix}, \begin{bmatrix} 0 \\ -\hat{k} \end{bmatrix}$$

Operations

- Quat * quat multiply

$$\begin{bmatrix} \omega_1 \\ \hat{\mathbf{c}}_1 \end{bmatrix}\begin{bmatrix} \omega_2 \\ \hat{\mathbf{c}}_2 \end{bmatrix} = \begin{bmatrix} (\omega_1\omega_2 - \hat{\mathbf{c}}_1 \cdot \hat{\mathbf{c}}_2) \\ (\omega_1\hat{\mathbf{c}}_2 + \omega_2\hat{\mathbf{c}}_1 + \hat{\mathbf{c}}_1 \times \hat{\mathbf{c}}_2) \end{bmatrix}$$

- Where $\cdot$ and $\times$ are the dot and cross product on 3 dimensional coordinate vectors.
- Correctly models (rotation matrix) * (rotation matrix) multiplications
- Unit quaternion multiplicative inverse

$$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}^{-1} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ -\sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}$$

Quaternion vector multiply setup

- Start with 4-coordinate vector $\mathbf{c} = \begin{bmatrix} \hat{\mathbf{c}} & 1 \end{bmatrix}^t$
- Left multiply it by a 4 by 4 rotation matrix R to get: $\mathbf{c}' = R\mathbf{c}$
- With result of from $\mathbf{c}' = \begin{bmatrix} \hat{\mathbf{c}}' & 1 \end{bmatrix}^t$

Quaternion vector multiply setup

- Let *R* be represented with the unit norm quaternion: $\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}$
- Use Cvec3 $\hat{\mathbf{c}}$ to create the non unit norm quaternion $\begin{bmatrix} 0 \\ \hat{\mathbf{c}} \end{bmatrix}$

Quat. vector multiply setup

- Perform the following triple quaternion multiplication: $\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix} \begin{bmatrix} 0 \\ \hat{\mathbf{c}} \end{bmatrix} \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}^{-1}$
- Result is of form: $\begin{bmatrix} 0 \\ \hat{\mathbf{c}}' \end{bmatrix}$
- We will write this in code as: quat * cvec = cvec
- Moral: quaternions encode the axis, and let us do ops.

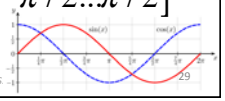
Power

- First extract the unit axis $\hat{\mathbf{k}}$ by normalizing the three last entries of the quaternion.
- Next, extract θ using the atan2 function in C++.
- atan(a,b) returns a unique $\theta \in [-\pi..\pi]$ such that $\sin(\theta) = a$ and $\cos(\theta) = b$
- So we get a unique value $\theta/2 \in [-\pi..\pi]$ and thus a unique $\theta \in [-2\pi...2\pi]$
- Define $\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}^{\alpha} = \begin{bmatrix} \cos\left(\frac{\alpha\theta}{2}\right) \\ \sin\left(\frac{\alpha\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}$

As α goes from 0 to 1, we get a series of rotations with angles going between 0 and θ

Power antipodes and cycle

- But what if the transition quaternion $\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}$
- presents a θ of more than 180(π) degrees :
- In particular, if $\cos\left(\frac{\theta}{2}\right) < 0$ then $|\theta| \in [\pi ... 2\pi]$
  - So $\alpha\theta$ would go more than 180 degrees which we don't want during interpolation
- In this case, suppose we had swapped to the antipode before calling power
- Then $\cos\left(\frac{\theta}{2}\right) > 0$, we get $\theta/2 \in [-\pi/2 .. \pi/2]$
  - And thus $\theta \in [-\pi .. \pi]$



Power antipodes and cycle

- So when we interpolate, before calling the power operator, we first check the sign of the first coordinate, and conditionally negate the quaternion.
- We call this the conditional negation operator cn

To interpolate

- To interpolate between two frames related to world frame by R_0 and R_1
- And suppose that these two matrices corresponds to the two quaternions:

$$\begin{bmatrix} \cos\left(\frac{\theta_0}{2}\right) \\ \sin\left(\frac{\theta_0}{2}\right)\hat{\mathbf{k}}_0 \end{bmatrix}, \begin{bmatrix} \cos\left(\frac{\theta_1}{2}\right) \\ \sin\left(\frac{\theta_1}{2}\right)\hat{\mathbf{k}}_1 \end{bmatrix}$$

- We output

$$\left(cn\left(\begin{bmatrix} \cos\left(\frac{\theta_1}{2}\right) \\ \sin\left(\frac{\theta_1}{2}\right)\hat{\mathbf{k}}_1 \end{bmatrix} \begin{bmatrix} \cos\left(\frac{\theta_0}{2}\right) \\ \sin\left(\frac{\theta_0}{2}\right)\hat{\mathbf{k}}_0 \end{bmatrix}^{-1} \right)^{\alpha} \begin{bmatrix} \cos\left(\frac{\theta_0}{2}\right) \\ \sin\left(\frac{\theta_0}{2}\right)\hat{\mathbf{k}}_0 \end{bmatrix}$$

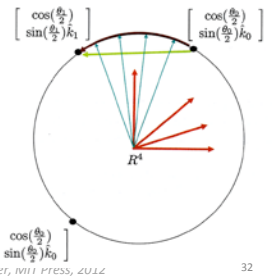
$$R_\alpha := (R_1 R_0^{-1})^\alpha R_0$$

Slerping

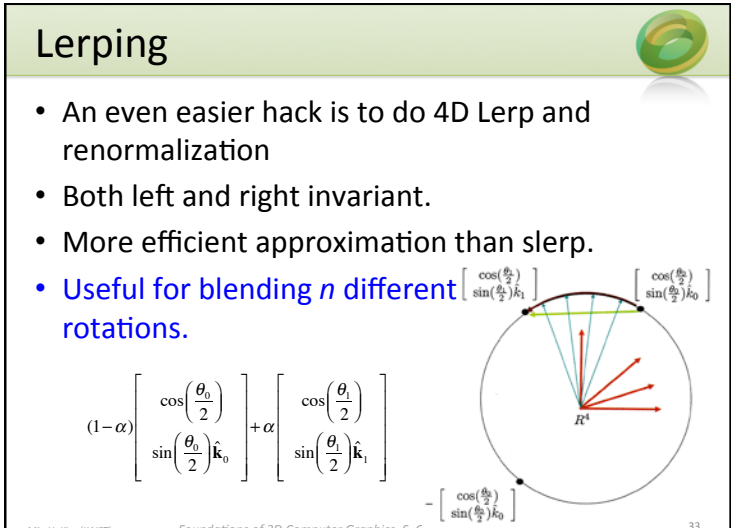
- This is called *spherical linear interpolation* or just *slerping* since it happens to match moving on a great circle in $\mathbb{R}^4$

$$\frac{\sin[(1-\alpha)\Omega]}{\sin(\Omega)}\vec{v}_0 + \frac{\sin[\alpha\Omega]}{\sin(\Omega)}\vec{v}_1$$

$$\frac{\sin[(1-\alpha)\Omega]}{\sin(\Omega)} \begin{bmatrix} \cos\left(\frac{\theta_0}{2}\right) \\ \sin\left(\frac{\theta_0}{2}\right)\hat{\mathbf{k}}_0 \end{bmatrix} + \frac{\sin[\alpha\Omega]}{\sin(\Omega)} \begin{bmatrix} \cos\left(\frac{\theta_1}{2}\right) \\ \sin\left(\frac{\theta_1}{2}\right)\hat{\mathbf{k}}_1 \end{bmatrix}$$



Lerping



- An even easier hack is to do 4D Lerp and renormalization
- Both left and right invariant.
- More efficient approximation than slerp.
- Useful for blending *n* different rotations.

$$(1-\alpha)\begin{bmatrix}\cos\left(\frac{\theta_0}{2}\right)\\ \sin\left(\frac{\theta_0}{2}\right)\hat{\mathbf{k}}_0\end{bmatrix}+\alpha\begin{bmatrix}\cos\left(\frac{\theta_1}{2}\right)\\ \sin\left(\frac{\theta_1}{2}\right)\hat{\mathbf{k}}_1\end{bmatrix}$$

Code

- Quaternion is four-tuple of real numbers.
- We provide qq multiplication (q1 * q2)
- inv(q)
- (q * c)
  - Copy over the 0/1 fourth coordinate
- MakeRotation functions
- Later on (asst 5) you will code the power operator: pow(q, alpha).
  - Using slerp(q0, q1, alpha)
  - Remember the conditional negation