Schedule

- 4/5 (Tue) Lecture – 3D rotation (Chap 6~7) / Interpolation (Chap 8)
- 4/6 (Wed) Open Lab
- 4/7 (Thur) Lecture – Projection& Depth (Chap 9,10,11)

- 4/12 (Tue) TA's Special Session for OpenGL (RE: **HW#3 out**!)
- 4/13 <Election Day> No lab session ----------- **HW#2 DUE**
- 4/14 (Thur) Lecture – From Vertex to Pixel (Chap.12)

- 4/19 (Tue) Lecture – Modeling (Chap.22) or Varying variable (Chap 13)
- 4/20~26 (Midterm Week)
  4/26 (Tuesday) 4~7 PM **Midterm Exam** @ E3-1 #1501

- 4/27 (Wed) Open Lab
- 4/28 (Thur) Lecture – Lighting (Chap. 13) ---------- **HW#3 DUE**

Midterm Exam

- Reading: Chapters 1 ~ 12 (13?)

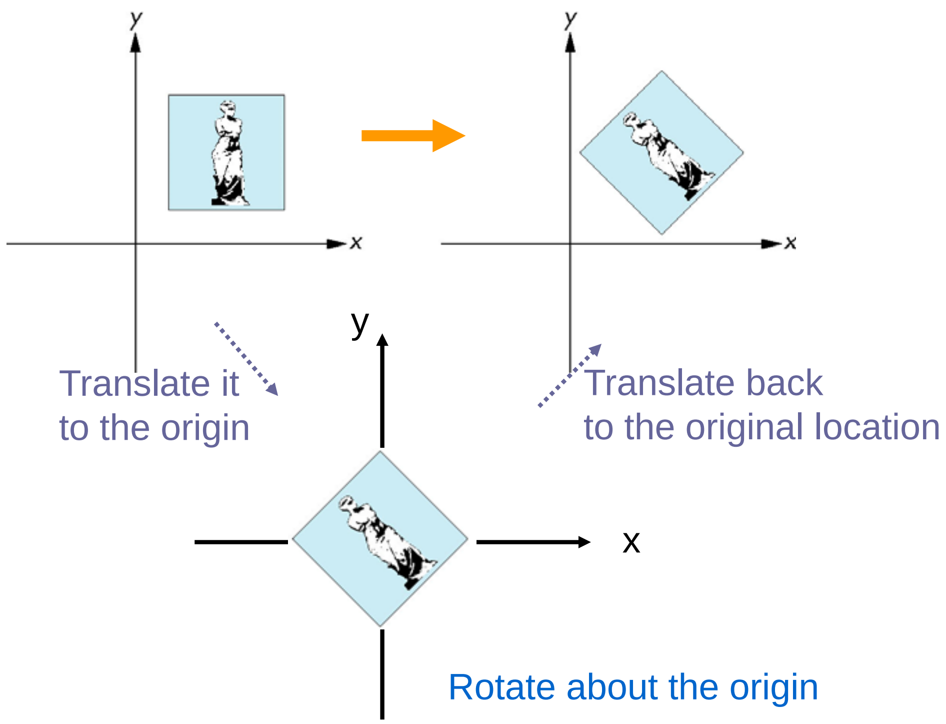
- Topics
  - Rendering Pipeline
  - Geometric Representation
  - Frames
  - Affine Transformation
  - Projection

3D Rotation

Ed Angel Book (Interactive Computer Graphics)
Chapter 3
Geometric Objects and Transformations

Our Textbook (S Gortler's book) :
Chapter 7 & 8
Quaternions & Balls

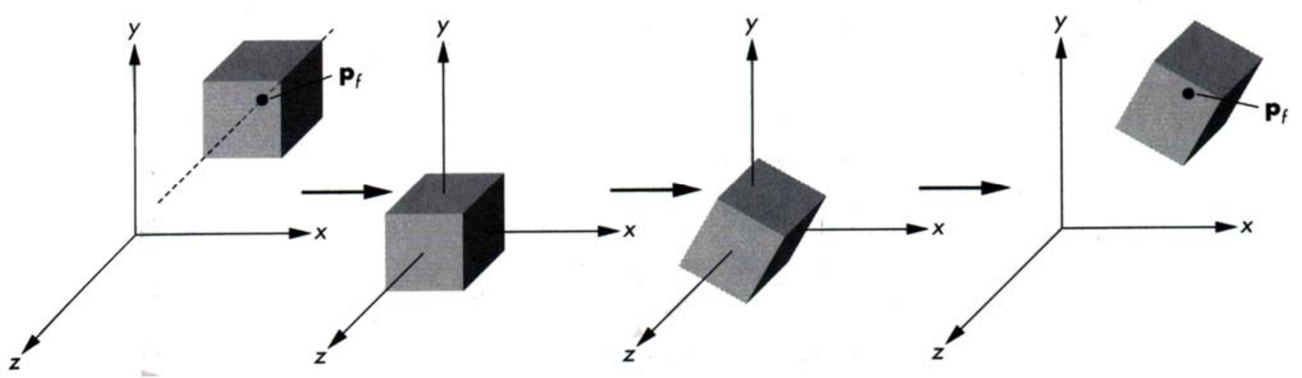
2D Rotation



Rotation about a Fixed Point

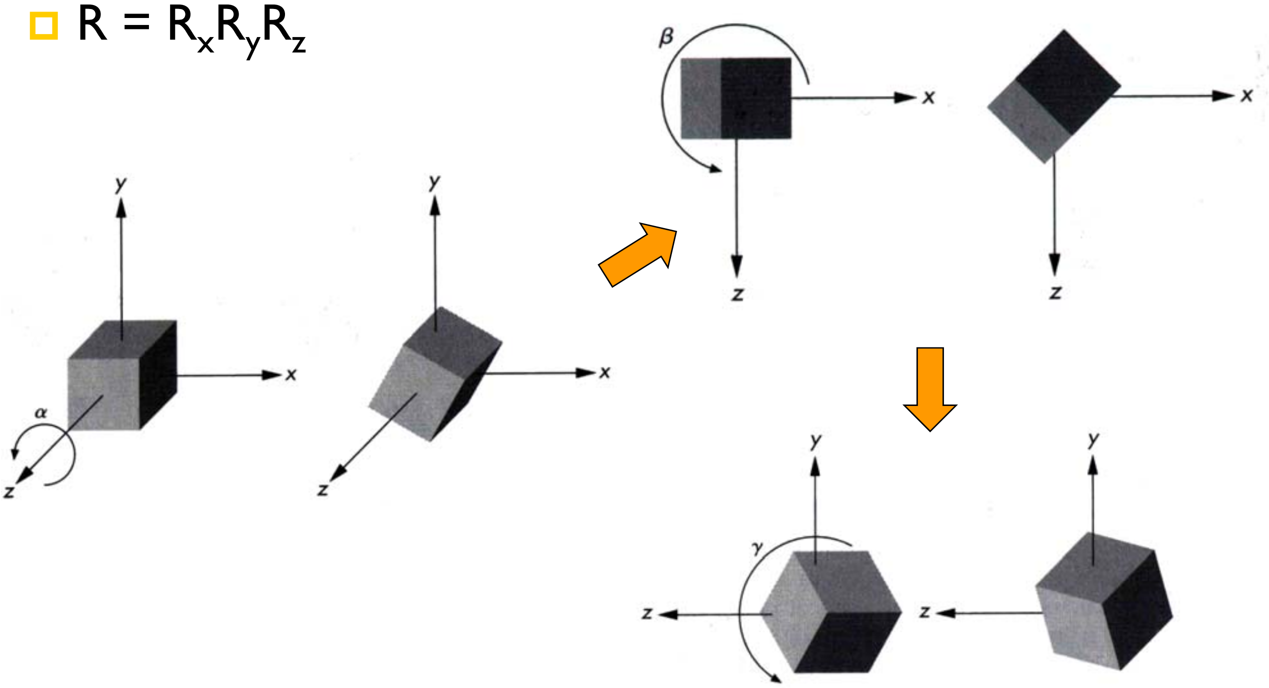
(& about Z-axis)

- $M = T(p_f)\, R_z(\theta)\, T(-p_f)$



General Rotation

- $R = R_x R_y R_z$



xyz-Euler angle rotation

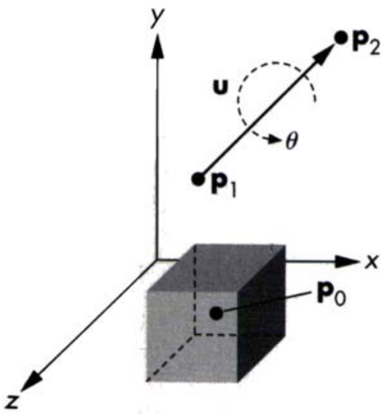
$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z R_y R_x = \begin{bmatrix} \cos\beta\cos\gamma & \cos\gamma\sin\alpha\sin\beta - \cos\alpha\sin\gamma & \cos\alpha\cos\gamma\sin\beta + \sin\alpha\sin\gamma & 0 \\ \cos\beta\sin\gamma & \cos\alpha\cos\gamma + \sin\alpha\sin\beta\sin\gamma & -\cos\gamma\sin\alpha + \cos\alpha\sin\beta\sin\gamma & 0 \\ -\sin\beta & \cos\beta\sin\alpha & \cos\alpha\cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x R_y R_z = \begin{bmatrix} \cos\beta\cos\gamma & -\cos\beta\sin\gamma & \sin\beta & 0 \\ \cos\alpha\sin\gamma + \sin\alpha\sin\beta\cos\gamma & \cos\alpha\cos\gamma - \sin\alpha\sin\beta\sin\gamma & -\sin\alpha\cos\beta & 0 \\ \sin\alpha\sin\gamma - \cos\alpha\sin\beta\cos\gamma & \sin\alpha\cos\gamma + \cos\alpha\sin\beta\sin\gamma & \cos\alpha\cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Axis

- Move P_o to the origin.
- Align u with the z-axis.
- Rotate by θ about the z-axis.
- Undo the alignment.
- Undo the translation.



$$M = T(p_0)R_x(-\theta_x)R_y(-\theta_y)R_z(\theta)R_y(\theta_y)R_x(\theta_x)T(-p_0)$$

Rotation about an Arbitrary Axis

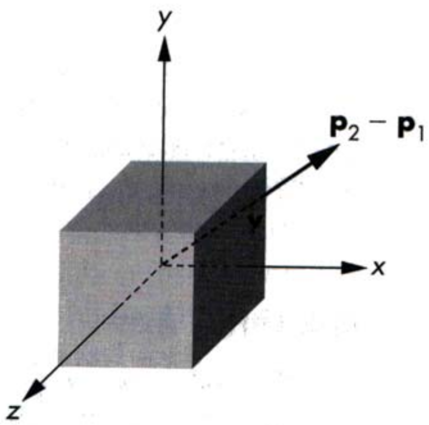
- Normalize u

$$u = P_2 - P_1$$

$$v = \frac{u}{|u|} = \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{bmatrix}$$

- Rotate along the x-axis until v hits the x-z plane.

- Rotate along the y-axis until v hits the z-axis.



$$M = T(p_0)R_x(-\theta_x)R_y(-\theta_y)R_z(\theta)R_y(\theta_y)R_x(\theta_x)T(-p_0)$$

Rotation about an Arbitrary Axis

- Finding θ_x and θ_y

$$v = (\alpha_x, \alpha_y, \alpha_z)$$

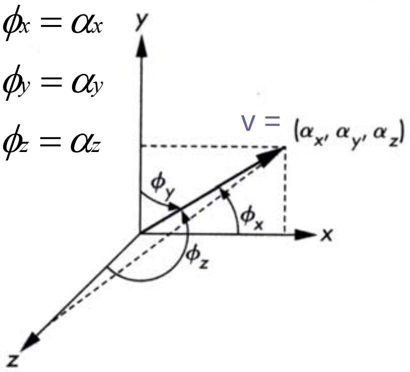
$$\alpha_x^2 + \alpha_y^2 + \alpha_z^2 = 1$$

- Direction cosines

$$\cos\phi_x = \alpha_x$$

$$\cos\phi_y = \alpha_y$$

$$\cos\phi_z = \alpha_z$$

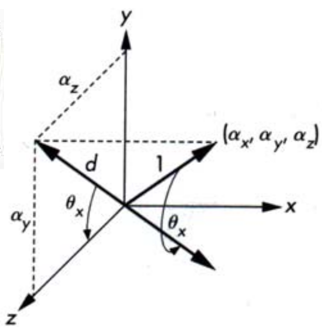


<note> only 2 of the direction angles are independent

- Computation of the x rotation

$$R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \alpha_z/d & -\alpha_y/d & 0 \\ 0 & \alpha_y/d & \alpha_z/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotate the line segment into the xz plane.

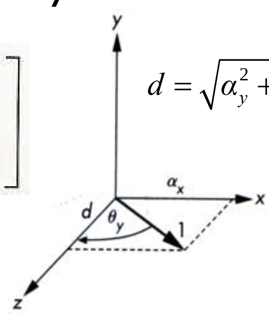


- Computation of the y rotation

$$R_y(\theta_y) = \begin{bmatrix} d & 0 & -\alpha_x & 0 \\ 0 & 1 & 0 & 0 \\ \alpha_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$d = \sqrt{\alpha_y^2 + \alpha_z^2}$$

<note> angle is clockwise



Rotation about an Arbitrary Axis

$$\mathbf{M} = \underline{\mathbf{T}(\mathbf{p}_0)}\,\underline{\mathbf{R}_x(-\theta_x)\mathbf{R}_y(-\theta_y)}\,\underline{\mathbf{R}_z(\theta)}\,\underline{\mathbf{R}_y(\theta_y)\mathbf{R}_x(\theta_x)}\,\underline{\mathbf{T}(-\mathbf{p}_0)}$$

Interfaces to 3D Applications

- How to control the direction of rotation of our scene?
  - Series of mouse clicks –
    left, mid, and right bottoms for x, y, and z axis rotations, respectively
  - Suppose that we want to use one mouse button for orienting an object.
    - One for zooming in and out
    - The other one for translation

Smooth Rotations

- $R(\theta) = R_x(\theta_x)\ R_y(\theta_y)\ R_z\ (\theta_z)$
  - Euler's angles
  - Fixed angles representation
  - Small angle approximation
    - $\sin\theta \approx \theta$, $\cos\theta \approx 1$.
- Rotation interpolation!

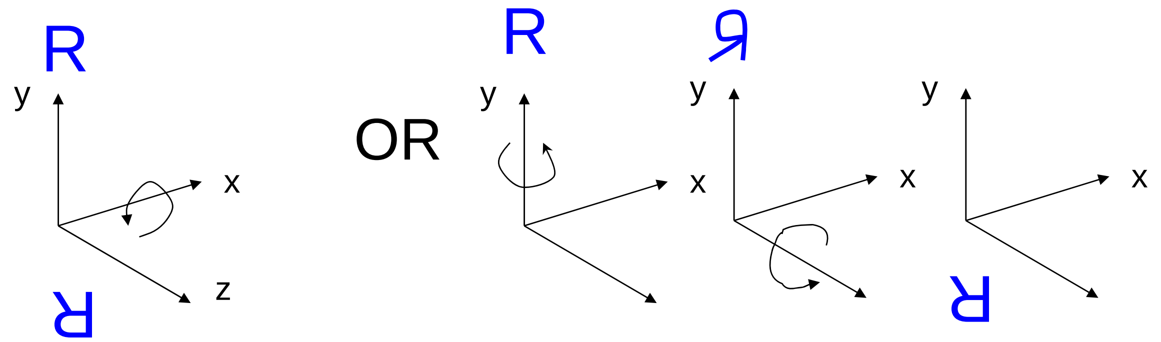
3D Rotations with Euler Angles

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

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

Euler Rotation Problems

- Interpolation between two Euler angles are not unique.
  - Example: (x,y,z) rotation to achieve the following:



Rotate(180, 1,0,0)
(0,0,0) -> (180,0,0)

Rotate(180, 0,1,0) then Rotate(180,0,0,1)
(0,0,0) -> (0,180,180)

- Direct interpolation of transformation matrix values can result in nonsense

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}$$

+90 y-axis rotation

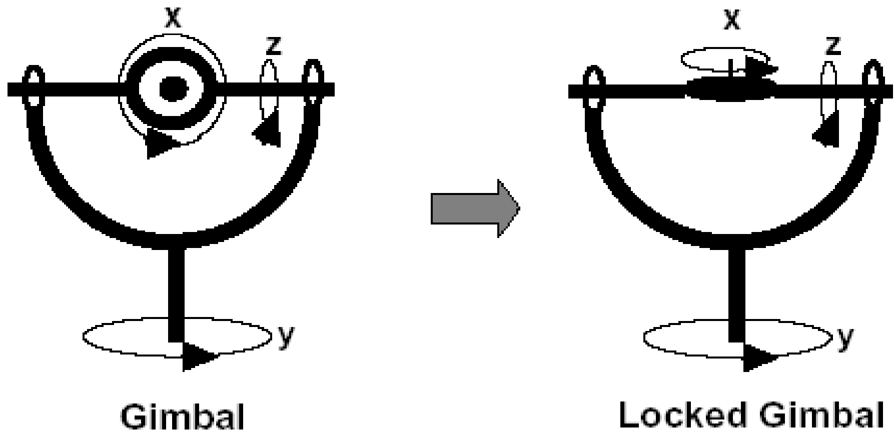
$$\begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

-90 y-axis rotation

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Halfway between orientation representation

- Gimbal Lock problem



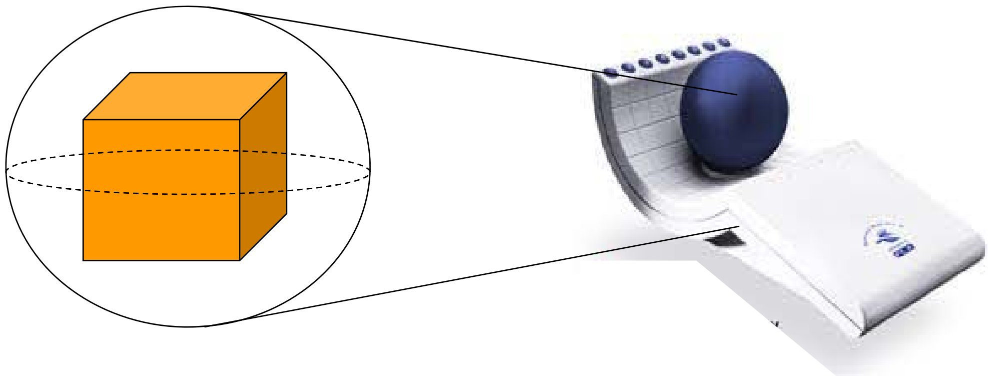
Alternative representation for **Rotations**

- $R(\theta) = R_x(\theta_x)\ R_y(\theta_y)\ R_z\ (\theta_z)$
  - Fixed angles representation

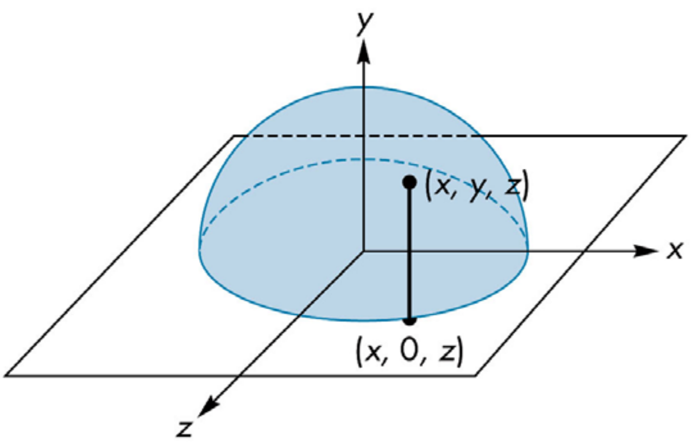
- Trackball
  - **Axis and angle representation**
  - A smooth rotation between the two orientations corresponds to a great circle on the surface of a sphere.

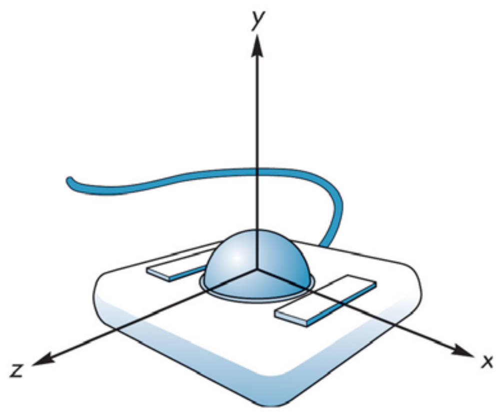
3D Rotations with Trackball

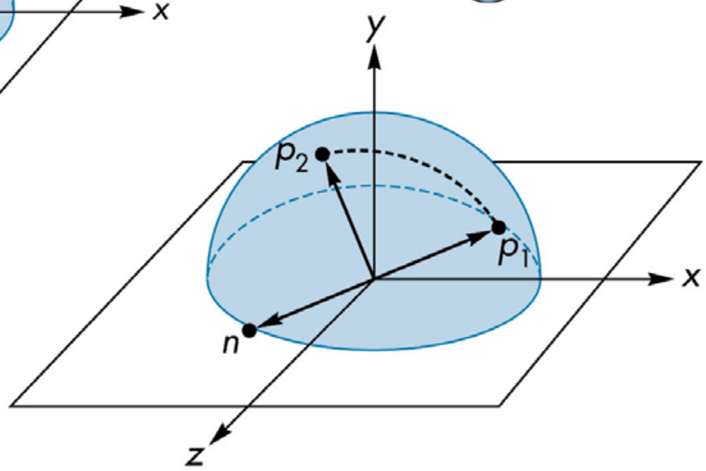
- Imagine the objects are rotated along with a imaginary hemi-sphere
- Allow the user to define 3D *rotation* using mouse click in 2D windows
- Work similarly like the hardware trackball devices



Trackball

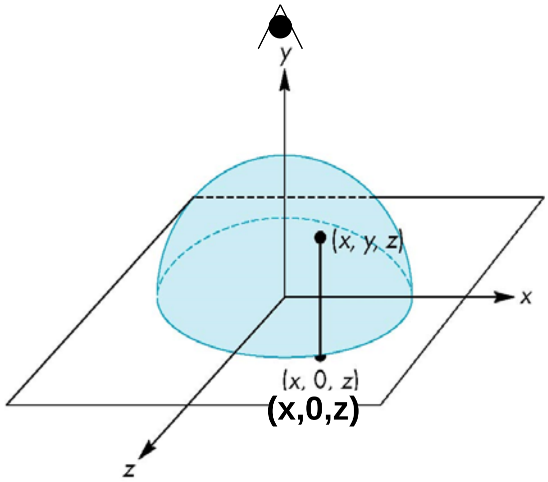


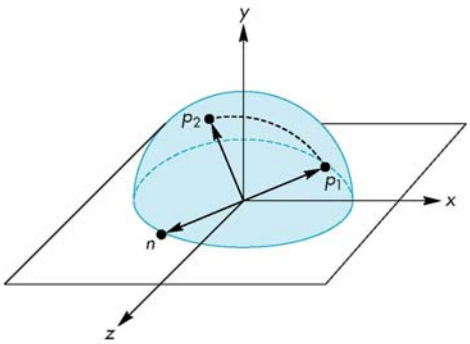


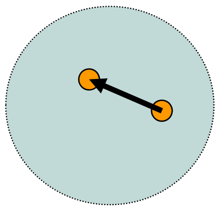


Virtual Trackball

- Superimpose a hemi-sphere onto the viewport
- This hemi-sphere is projected to a circle inscribed to the viewport
- The mouse position is projected orthographically to this hemi-sphere







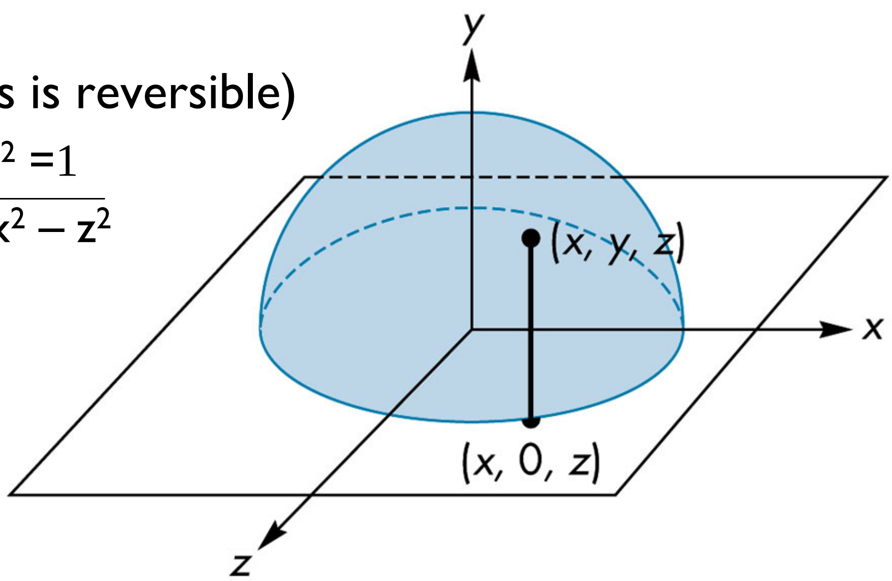
- Keep track the previous mouse position and the current position
- Calculate their projection positions p_1 and p_2 to the virtual hemi-sphere
- We then rotate the sphere from p_1 to p_2 by finding the proper rotation axis and angle
- This rotation (in eye space!) is then applied to the object

Virtual Trackball

- The radius of the ball = 1
- Orthogonal projection to the plane (this process is reversible)

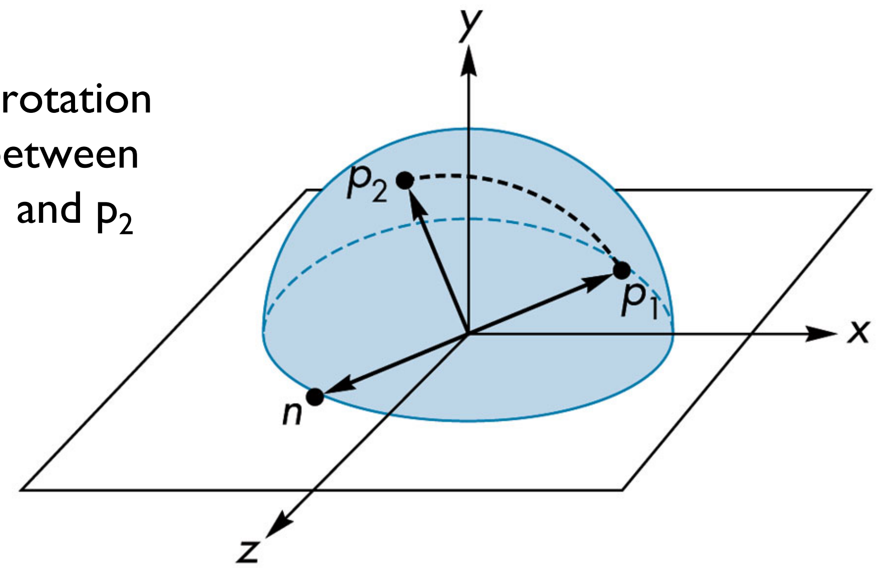
  : $x^2 + y^2 + z^2 = 1$

  : $y = \sqrt{1 - x^2 - z^2}$



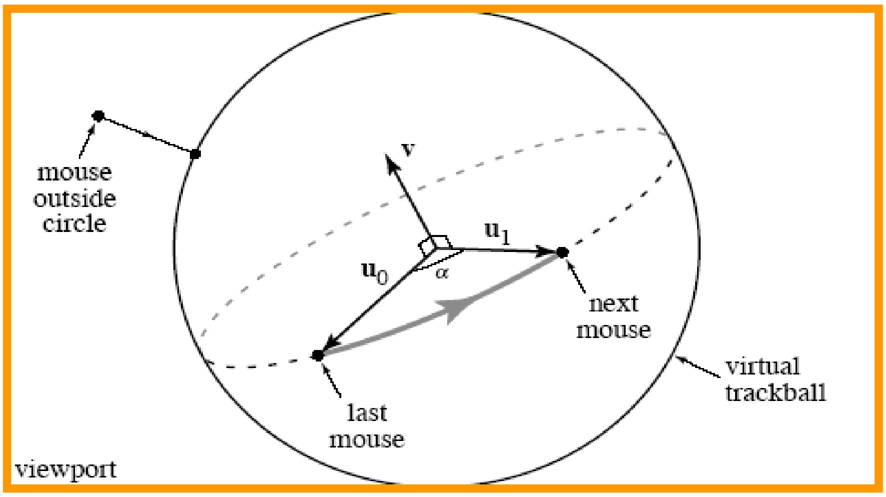
- 2 positions on the hemisphere
  - The motion of the trackball that moves from p_1 to p_2 can be achieved by a rotation about n.
  - $n = p_1 \times p_2$
  - The angle of rotation is the angle between the vector p_1 and p_2
  - $|\sin \theta| = |n|$

$$|\sin \theta| = \frac{|\mathbf{n}|}{|\mathbf{p}_1||\mathbf{p}_2|}$$



  - Small angle
    $\sin \theta \approx \theta$

Virtual Trackball



If a point is outside the circle, project it to the nearest point on the circle (set z to 0 and renormalize (x,y))

Note: normalize viewport y extend to -1 to 1

(arbitrary)rotation axis and an angle

Axis-angle representation

- Unit vector axis $[k_x, k_y, k_z]^t$ and angle θ
- Can be expressed using the matrix

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

where $c \equiv cos\theta$, $s \equiv sin\theta$, and $v \equiv 1 - c$

<p.16 textbook: **Chapter 2**>

Quaternions

- Invented in 1843 as an extension to the complex numbers
- Used by computer graphics since 1985
- Quaternions:
  - Provide an alternative method to specify rotation
  - Allow smooth and continuous rotation

Quaternions

- Extend the concept of rotation in 3 dimensions to rotation in 4 dimensions.
  - This avoids the problem of 'gimbal-lock' and allows for the implementation of smooth and continuous rotation.
- Defined using 4 floating point values (x,y,z,w).
  - These are calculated from the combination of the 3 coordinates of the rotation axis and the rotation angle.
  - Allow the programmer to rotate an object through an arbitrary rotation axis and angle (rather than through a series of successive rotations).

Mathematical Background

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

  $q = [q_w, q_x, q_y, q_z] = q_v + q_w$, where

  q_w is the real part and

  $q_v = iq_x + jq_y + kq_z = (q_x, q_y, q_z)$ is the imaginary part
- $i*i = j*j = k*k = -1;$
- $j*k = -j*k = i;$ $k*l = -i*k = j;$ $i*j = -j*l = k;$
- All the regular vector operations (dot product, cross product, scalar product, addition, etc) applied to the imaginary part q_v

Don't get confused with a homogenous representation!

Basic Operations

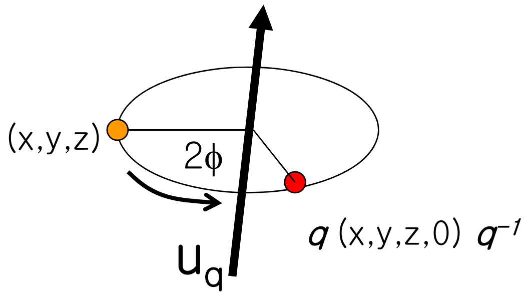
- Multiplication:
  $qr = (q_w r_w - q_v \bullet r_v,\ q_v \times r_v + r_w q_v + q_w r_v)$
- Addition: $q+r = (q_w+r_w\ q_v+r_v,)$
- Conjugate: $q^* = (q_w,\ -q_v)$
- Norm (magnitude) = qq^*
  $= q^*q$
  $= q_v \bullet q_v + q_w^2$
- Identity $i = (1, 0)$
- Inverse $q^{-1} = (1/\ |q|)\ q^*$

Polar Representation

- Remember a 2D unit complex number
  $\cos\theta + i \sin\theta = e^{i\theta}$
- A unit quaternion $\boldsymbol{q}$ may be written as:
  $\boldsymbol{q} = (\cos\phi, \sin\phi\ \mathbf{u}_q)$
  $= \cos\phi + \sin\phi\ \mathbf{u}_q,$
  where $\mathbf{u}_q$ is a unit 3-tuple vector
- So we can have a similar form for quaternions:
  $\boldsymbol{q} = e^{\phi \mathbf{u}_q}$

Quaternion Rotation

- Given a point p = (x,y,z)
  we first convert it to a quaternion
  $p' = ix+jy+kz+0 = (0, p_v)$
- Also given a unit quaternion
  $q = (\cos\phi, \sin\phi u_q)$
- Then, $q\, p'\, q^{-1}$ rotates p around u_q by an angle 2ϕ !!



$q = (\cos\phi, \sin\phi\, u_q)$

Rotation Concatenation

- Concatenation is easy – just multiply all the quaternions q_1, q_2, q_3, …. Together

$$(q_3\,(q_2\,(\,q_1\;\; p'\;\; q_1^{-1}\,)\,q_2^{-1})\,q_3^{-1}\,)$$
$$= (q_3\, q_2\, q_1)\, p'\, (q_1^{-1}\, q_2^{-1}\, q_3^{-1})$$

Quaternions

- There is a corresponding 4x4 matrix
  - For unit quaternions, it simplifies to

$$M^q = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy & 0 \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx & 0 \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where q = (x,y,z,w).

  - Once the quaternion is constructed, no trigonometric functions need to be computed , so the conversion process is efficient in practice.

Quaternions

- 4-vector related to axis and angle, unit magnitude
  - Rotation about axis (n_x,n_y,n_z) by angle θ:
- The rotation is still performed using matrix mathematics.
  - However, instead of multiplying matrices together, quaternions representing the axis of rotation are multiplied together.
  - The final resulting quaternion is then converted to the desired rotation matrix.
  - Since the rotation axis is a unit direction vector, it may be calculated through vector mathematics, or from spherical coordinates (longitude/latitude)

Quaternions

- Given two unit quaternions, *q* and *r*, the concatenation of first applying *q* and then *r* to a quaternion *p* (which can be interpreted as a point p) is:

$$r\,(qpq^*)\,r^* = (rq)\,p\,(q^*r^*)$$
$$= (rq)\,p\,(rq)^* \quad = c\,p\,c^*$$

- Smooth rotation
  - Quaternions are interpolated!
  - This allows for smooth and predictable rotation effects.

Back to our textbook

Motivation

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

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

- Then we add back in the translations

- <note> For your homework#2, you were asked to implement separately Linear part and Translation part!

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 $\vec{\mathbf{o}}_1^t$

What we want

- Want 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:

$$\vec{\mathbf{w}}^t (R_1 R_0^{-1})^0 R_0 = \vec{\mathbf{w}}^t R_0 = \vec{\mathbf{o}}_0$$

$$\vec{\mathbf{w}}^t (R_1 R_0^{-1})^1 R_0 = \vec{\mathbf{w}}^t R_1 = \vec{\mathbf{o}}_1$$

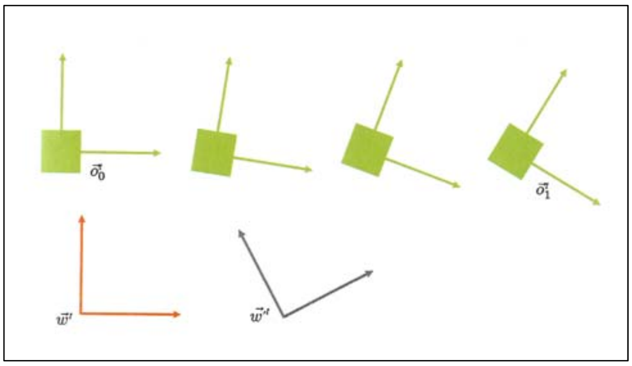
- The transition rotation fixes a unique axis
- This axis depends only on $\vec{\mathbf{o}}_0$ and $\vec{\mathbf{o}}_1$. Not any choice of world frame.
- Up to cycles, this gives us a unique interpolation

Hard part .. Solved!

- 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.

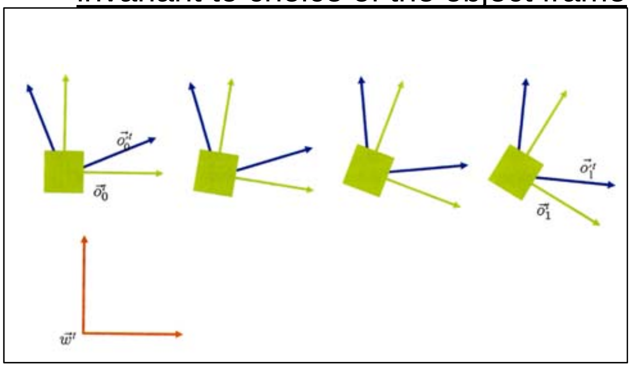
Left and right invariant

- Relation ~
- left invariant: x~y implies zx ~ zy
- right invariant: x~y implies xz ~ yz



invariant to choice of the world frame

invariant to choice of the object frame



Quaternion (rotation)

- We use the quaternion representation for interpolating *rotation* only (not translation).
- We cannot interpolate a rotation matrix *R* in
  $$\vec{\mathbf{o}}^t_\alpha = \vec{\mathbf{w}}^t R_\alpha$$
- Interpolating the three scalar in the XYZ Euler angles is not a good solution for natural movement.
- The quaternion rotation itself allows us to interpolate the rotation angle. $\vec{\mathbf{o}}^t_\alpha = \vec{\mathbf{w}}^t (R_1 R_0^{-1})^\alpha R_0$

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 is presented as

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

  - Oddity: *the division by 2* will be needed to make the operations work out as needed.

Unit Quaternion

- 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.
- Identity rotation example, flip rotation example

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

Operations

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

$$\begin{bmatrix} 0 \\ \hat{\mathbf{c}}_1 \end{bmatrix}\begin{bmatrix} 0 \\ \hat{\mathbf{c}}_2 \end{bmatrix} = \begin{bmatrix} -\hat{\mathbf{c}}_1 \cdot \hat{\mathbf{c}}_2 \\ \hat{\mathbf{c}}_1 \times \hat{\mathbf{c}}_2 \end{bmatrix} \qquad \begin{bmatrix} \hat{\mathbf{k}}_1 \cdot \hat{\mathbf{k}}_2 \\ \hat{\mathbf{k}}_1 \times \hat{\mathbf{k}}_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \hat{\mathbf{k}}_2 \end{bmatrix}\begin{bmatrix} 0 \\ -\hat{\mathbf{k}}_1 \end{bmatrix}$$

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

Rotation with quaternion operations

- 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 $\mathbf{c}' = \begin{bmatrix} \hat{\mathbf{c}}' & 1 \end{bmatrix}^t$
- 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 $\hat{\mathbf{c}}$ to create the non unit norm quaternion $\begin{bmatrix} 0 \\ \hat{\mathbf{c}} \end{bmatrix}$
- 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: $\begin{bmatrix} 0 \\ \hat{\mathbf{c}}' \end{bmatrix}$

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

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

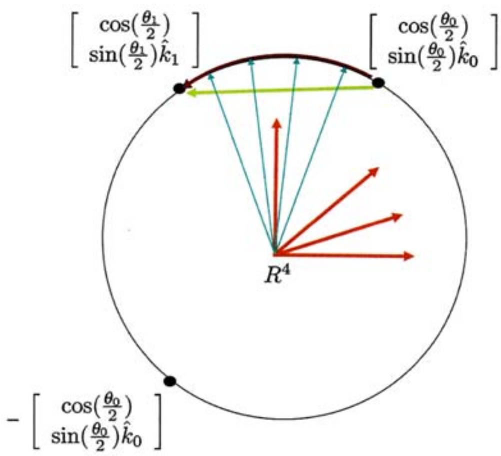
$$R_\alpha := (R_1 R_0^{-1})^{\alpha} R_0$$

Slerping

- This is called *spherical linear interpolation* or just *slerping* since it happens to match moving on a great circle in 4-dimension.

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

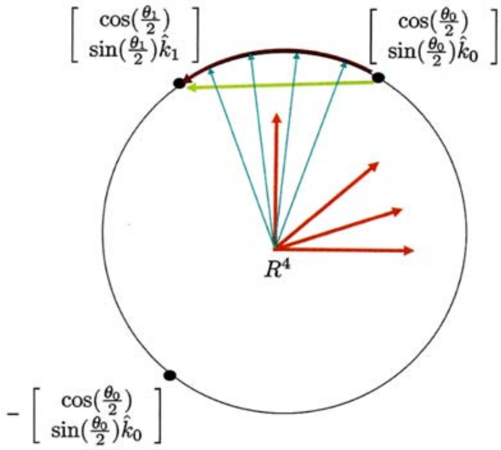
$$\frac{\sin[(1-\alpha)\Omega]}{\sin(\Omega)} \begin{bmatrix} \cos\left(\frac{\theta_0}{2}\right) \\ \sin\left(\frac{\theta_0}{2}\right)\hat{\mathbf{k}}_0 \end{bmatrix} + \frac{\sin[\alpha\Omega]}{\sin(\Omega)} \begin{bmatrix} \cos\left(\frac{\theta_1}{2}\right) \\ \sin\left(\frac{\theta_1}{2}\right)\hat{\mathbf{k}}_1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



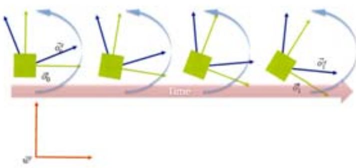
Lerping

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

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



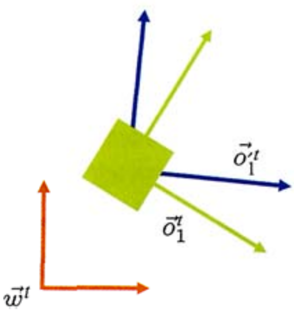
Quaternion (rotation)



- To interpolate between two frames related to world frame by R_0 and R_1

$$\vec{\mathbf{o}}_0^t = \vec{\mathbf{w}}^t R_0 \ when\ \alpha = 0$$

$$\vec{\mathbf{o}}_1^t = \vec{\mathbf{w}}^t R_1 \ when\ \alpha = 1$$



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

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

Putting back the translation

- Let's now build a data structure to represent an RBT
- Recall: RBT data structure $A = TR$

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

RBT Interpolation

- Given two frames $\vec{\mathbf{o}}_0^t = \vec{\mathbf{w}}^t O_0, \vec{\mathbf{o}}_1^t = \vec{\mathbf{w}}^t O_1$
- Given two RBTs
  - We will write it as matrices $O_0 = (O_0)_T (O_0)_R$ and $O_1 = (O_1)_T (O_1)_R$
- Interpolate between them by: linearly interpolating the two translations to get: T_α
- Slerp between the rotation quaternions to obtain the rotation R_α
- Set the interpolation RBT O_α to be $T_\alpha R_\alpha$
- Set $\vec{\mathbf{o}}_\alpha^t = \vec{\mathbf{w}}^t O_\alpha$

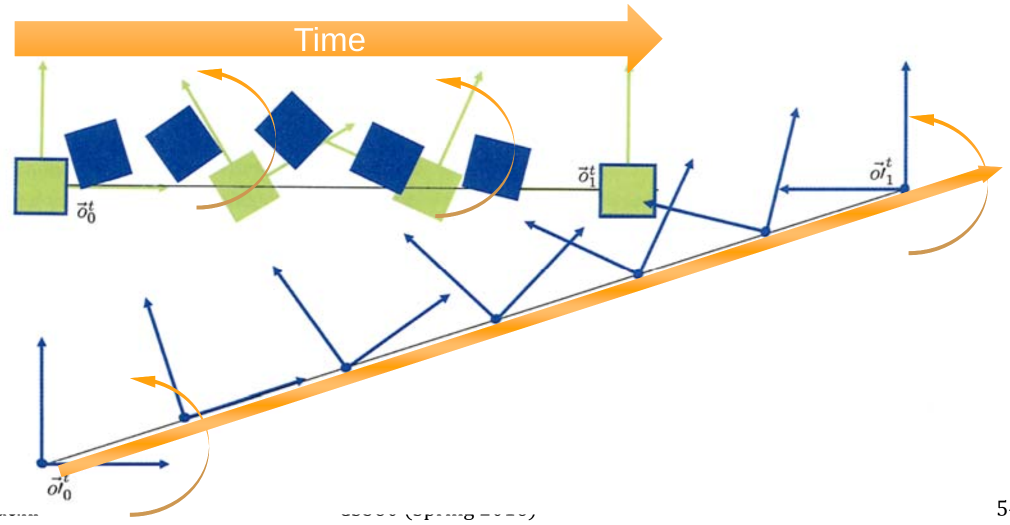
TRACK and ARC

- How should we link mouse motion to object rotation
- Can do better than our current setup
- Want the feeling of pushing a sphere around (trackball)
- Want *path invariance* (arcball)

- Reminders:
  - Affine transform: $A_{affine} = TL$
  - Rigid body transform: $A_{RBT} = TR$

RBT Interpolation Behavior

- Origin of $\vec{\mathbf{o}}^t$ travels in a straight line with constant velocity,
- The vector basis of $\vec{\mathbf{o}}^t$ rotates with constant angular velocity about a fixed axis.
- Physically natural if origin is at center of mass



RBT Interpolation Behavior

- Even though the quaternion rotation is left and right invariant, the quaternion rotation + object translation is left invariant.
- The translation of the origin plays special role.
- If we use different object frames for same geometry, we get different interpolations
  - Not right invariant

Trackball and Arcball

- How should we link mouse motion to object rotation
- Can do better than our current setup
- Want the feeling of pushing a sphere around (trackball)
- Want *path invariance* (arcball)

- ➔ Homework #2