Cornell University CS 569: Interactive Computer Graphics

Rotations (and other transformations)

Lecture 4

Rotation as rotation matrix

- Storage
  - 9 floats
  - orthogonal and unit length columns and rows
  - inverse is transpose
- Apply to vector
  - matrix–vector multiply (15 flops)
- Compose rotations
  - matrix–matrix multiply (45 flops)
- Pro
  - simple; efficient to apply; easy to compose
- Con
  - 3x redundant; slow to construct and compose; interpolation ill-behaved

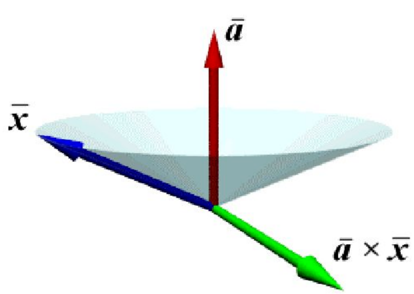
Rotation as Euler angles

- Storage
  3 floats
- Apply to vector
  three rotations
- Compose rotations
  ouch!
- Pro
  simple; compact; efficient to apply
- Con
  gimbal lock; hard to construct; hard to compose; interpolation very ill-behaved

Rodrigues' rotation formula

$$R(\mathbf{a}, \theta)\mathbf{x} = (\cos\theta)\mathbf{x} + (\sin\theta)(\mathbf{a} \times \mathbf{x}) + (1 - \cos\theta)(\mathbf{a} \cdot \mathbf{x})\mathbf{a}$$

$$R(\mathbf{a}, \theta) = (\cos\theta)I + (\sin\theta)\tilde{\mathbf{a}} + (1 - \cos\theta)\mathbf{a}\mathbf{a}^T$$



[Leonard McMillan]

Rotation as axis & angle

- Storage
  unit vector axis + angle (4 floats), or
  axis scaled by angle (3 floats)
- Apply to vector
  Rodrigues' formula (32 flops + cos/sin/sqrt for setup)
- Compose rotations
  ouch!
- Pro
  simple; reasonably efficient to apply; construction simple
- Con
  composition not obvious

Quaternions for Rotation

- A quaternion is an extension of complex numbers
- Review: complex numbers

$$z = a + bi$$
$$z' = a - bi$$
$$\| z \| = \sqrt{z * z'} = \sqrt{a^2 + b^2}$$

[Kavita Bala]

ONB in quaternions

- Each of i, j and k are square root of –1

$$i * i = -1, \quad j * j = -1, \quad k * k = -1$$

- Cross-multiplication is like cross product

$$i * j = -j * i = k$$
$$k * i = -i * k = j$$
$$j * k = -k * j = i$$

[Kavita Bala]

ONB in quaternions

- A quaternion is an extension of complex numbers: 4D space

$$q = w + xi + yj + zk$$
$$q' = w - xi - yj - zk$$
$$\| q \| = \sqrt{w^2 + x^2 + y^2 + z^2}$$

[Kavita Bala]

Quaternion Properties

- Associative

$$(q_1 * q_2) * q_3 = q_1 * (q_2 * q_3)$$

- Not commutative

$$q_1 * q_2 \neq q_2 * q_1$$

- Unit quaternion

$$\| q \| = 1$$

$$q^{-1} = q'$$

[Kavita Bala]

Quaternion for Rotation

- Linear combination of 1, *i*, *j*, *k*

$$q = w + xi + yj + zk = (s, v)$$

$$s = w$$

$$v = [x \quad y \quad z]$$

- Multiplication

$$q_1 = (s_1, v_1)$$

$$q_2 = (s_2, v_2)$$

$$q_1 * q_2 = (s_1 s_2 - v_1 . v_2, s_1 v_2 + s_2 v_1 + v_1 \times v_2)$$

[Kavita Bala]

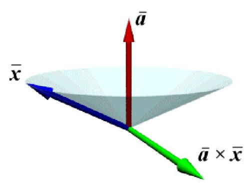
Quaternion for Rotation

- Rotate about axis a by angle θ

$$q = (s, v)$$

$$s = \cos\frac{\theta}{2}$$

$$v = \vec{a}\sin\frac{\theta}{2}$$



[Leonard McMillan]

[Kavita Bala]

Rotation Using Quaternion

- A point in space is a quaternion with 0 scalar

$$X = (0, \vec{x})$$

$$x_{rotated} = qXq^{-1}, where, \quad q^{-1} = q'$$

- Composing rotations

  q1 and q2 are two rotations

  First, q1 then q2

$$x_{rotated} = q_2 * (q_1 X q_1^{-1}) * q_2^{-1}$$

$$x_{rotated} = (q_2 * q_1) X (q_1^{-1} * q_2^{-1})$$

$$x_{rotated} = (q_2 * q_1) X (q_2 * q_1)^{-1}$$

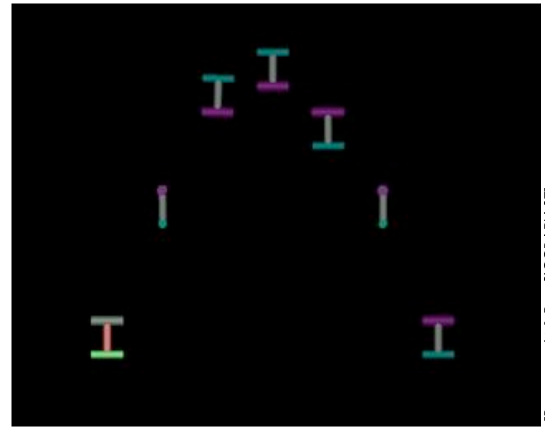
[Kavita Bala]

Matrix for quaternion

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

[Kavita Bala]

Quaternion spline interpolation



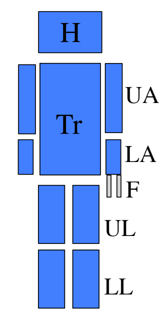
[Ramamoorthi & Barr SIGGRAPH 97]

Rotation as quaternion

- Storage
  - coeffs of 1, i, j, k (4 floats)
  - unit vector
- Apply to vector
  - quaternion rotation formula
- Compose rotations
  - quaternion multiplication
- Pro
  - reasonably efficient to apply; construction simple; well-behaved interpolation
- Con
  - difficult to understand at first

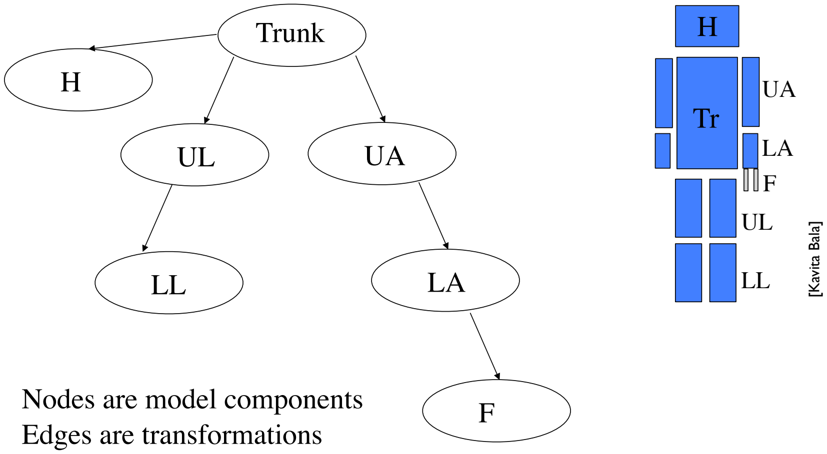
Hierarchical Transforms

- Articulated body
- Every object has local frame of reference
  - Example local coordinate system at center of box



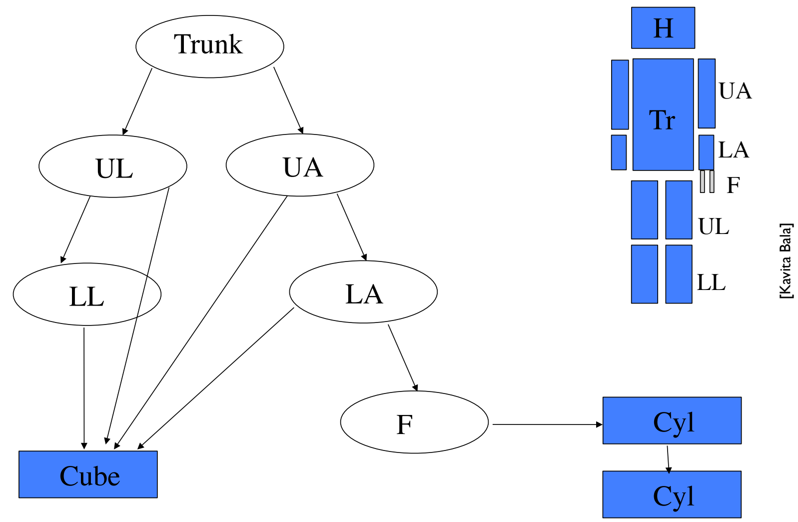
[Kavita Bala]

Tree of Transforms



Nodes are model components
Edges are transformations

DAG/Instancing

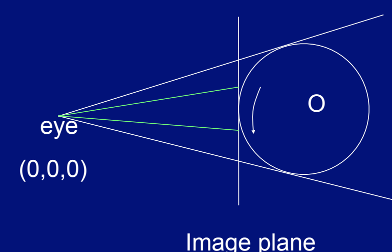


Trackball

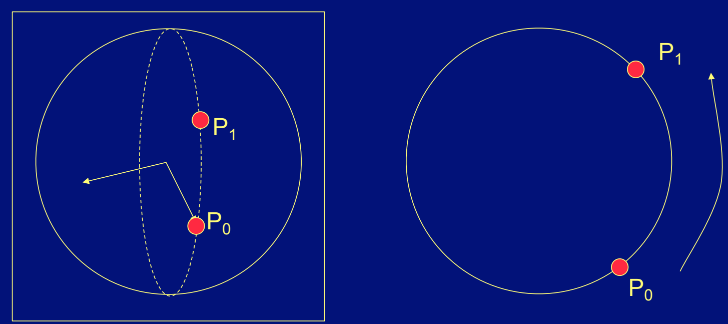
- Pan/Zoom/Orbit are not enough
- Want to inspect an object
- Want to rotate about some axis and angle
- But only have 2 degrees of freedom

Trackball

- There is a ball in front of image plane
- Grab the ball to move camera



How does it work?



Algorithm

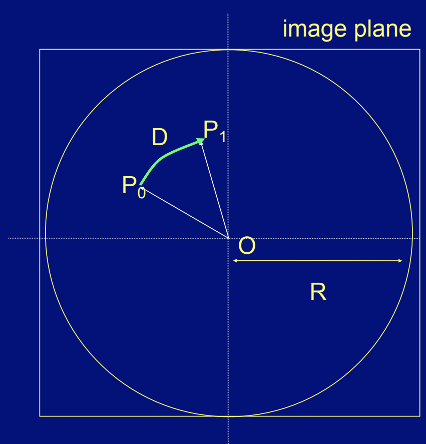
- Assume circle in screen

$$(x-O_x)^2 + (y-O_y)^2 + (z-O_z)^2 = R^2$$

- Assume mouse moves from P0 to P1
- Get 3D points P0 and P1 from equation
- Axis a = (P0-O) x (P1-O)

- Angle theta = k ||P1-P0||
- Rotate by theta around a
- P0 (next frame) = P1

Trackball Terms



$P_0 = (x_0, y_0, z_0)$

$P_1 = (x_1, y_1, z_1)$

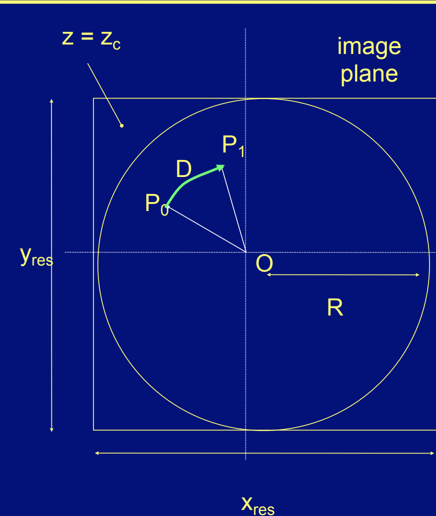
$(x-O_x)^2 + (y-O_y)^2 + (z-O_z)^2 = R^2$

$D = P_1 - P_0$

$axis_{rotation} = (P_1-O)\times(P_0-O)$

$angle_{rotation} = k\,|D|$

Trackball Terms



$(x-O_x)^2 + (y-O_y)^2 + (z-O_z)^2 = R^2$

$Z-O_z = \mathrm{sqrt}(R^2-(x-O_x)^2 + (y-O_y)^2)$

O is arbitrary

$O = (x_{res}/2, y_{res}/2, z_c)$