Camera Models and Quaternions

Adam Mally
CIS 5600
University of Pennsylvania

Review of Camera Matrices

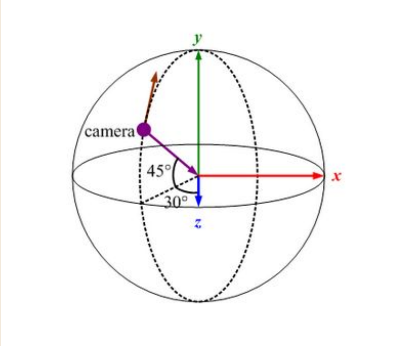
- Two main components to a camera matrix:
    - Projection matrix
        - Perspective projection makes objects farther from the camera appear smaller
        - Projects from the viewing frustum to <-1,1>
    - View matrix
        - Transforms the world by the inverse of the camera’s position and rotation

Spherical Camera Rotation Models

- How can we compute the “eye position” needed to compute the view matrix?
    - Polar coordinate rotation
    - Euler angle rotation
    - Quaternion-based rotation

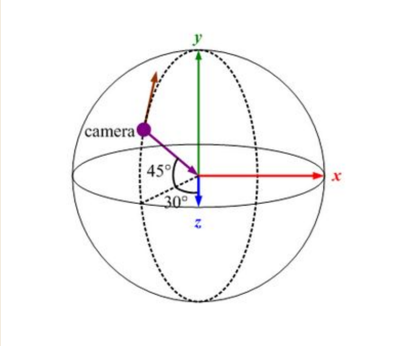
Polar coordinates

- Camera on a sphere
  - Use polar coordinates (ϕ, θ, r)
  - Rotate vertically/horizontally by changing ϕ and θ
  - Zoom in/out by changing the radius r
  - Pan up/down/left/right by moving the center of the sphere



Polar coordinates: problem!

- What happens when we rotate so we're looking down the "north pole" of the sphere?
    - Rotating about the Y axis (i.e. the north pole of the sphere) twists our view but does not change the position of the camera!

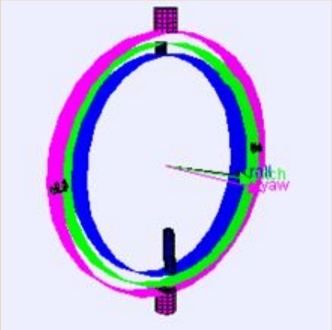


Euler angles

- Use three rotation angles instead of two
  - Polar coordinates: (ϕ, θ, r)
  - Euler angles: (ϕ, θ, ψ, r)
  - ϕ, θ, ψ are applied one at a time and represent the pitch, roll, and yaw rotations (X, Y, and Z axes)

Euler angles: problem!

- When two of the three angles line up, you still lose a degree of freedom
  - Known as **gimbal lock**



Quaternions

- Quaternions are an extension of complex numbers
- $q = [q_w \; q_x \; q_y \; q_z]$
- $q = w + xi + yj + zk$
  - i, j, and k are complex numbers where
    - $i^2 = j^2 = k^2 = -1$
    - $ij = k$
    - $jk = i$
    - $ki = j$
    - $ji = -k$
    - $kj = -j$

Quaternion Operations

- Remember: $q = w + xi + yj + zk$
    - We only store the real components of a quaternion
- Length of a quaternion: $\sqrt{(w^2 + x^2 + y^2 + z^2)}$
    - Can normalize a quaternion by dividing it by its length, as you would a vec4
- Dot product
    - $\text{dot}(q1, q2) = q1_w q2_w + q1_x q2_x + q1_y q2_y + q1_z q2_z$
    - $\text{dot}(q1, q2) = |q1||q2|\cos(\theta)$
        - The angle between *q1* and *q2* is half the angle needed to rotate between their orientations in 3D space

Quaternion Multiplication

- Multiplication:
  - $q1 = [w1, x1, y1, z1], q2 = [w2, x2, y2, z2]$
  - $q3 = q1 * q2$
  - $q3_w = w1w2 - x1x2 - y1y2 - z1z2$
  - $q3_x = w1x2 + x1w2 + y1z2 - z1y2$
  - $q3_y = w1y2 - x1z2 + y1w2 + z1x2$
  - $q3_z = w1z2 + x1y2 - y1x2 + z1w2$
- Note that it is anti-commutative
  - $q1q2 = -q2q1$

Quaternions as rotations

- We can use normalized quaternions to represent rotations
  - Let (vx, vy, vz) be a unit vector
  - $q = cos(\theta/2) + sin(\theta/2)v_x i + sin(\theta/2)v_y j + sin(\theta/2)v_z k$
  - $q = [cos(\theta/2), sin(\theta/2)v_{x,} sin(\theta/2)v_{y,} sin(\theta/2)v_z]$
  - *q* represents the rotation around axis *v* by *θ*
- Compose rotations by multiplying: $q_{total} = q_{new} * q_{original}$
- Like matrix multiplication, this is not commutative!
- As you’ll see soon, -*q* represents the same orientation as *q*

Quaternion to Matrix

$$\begin{pmatrix} w^2+x^2-y^2-z^2 & 2xy-2wz & 2xz+2wy & 0 \\ 2xy+2wz & w^2-x^2+y^2-z^2 & 2yz-2wx & 0 \\ 2xz-2wy & 2yz+2wx & w^2-x^2-y^2+z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$q = [w, x, y, z]$

$w = \cos(\theta/2)$, $\langle x,y,z \rangle = \sin(\theta/2) * \langle v_x\ v_y\ v_z \rangle$

Visualizing quaternion rotation

- You’re standing on the surface of a sphere
- From your perspective, you can move forward/backward and left/right
- You can also turn to change what “forward” and “right” represent
- You have three degrees of freedom of movement
  - No *perception* of things like fixed poles or longitude/latitude, therefore there are no singularities
- From your point of view, you might as well be moving on an infinite 2D plane, except that if you go too far in a particular direction you end up back where you started

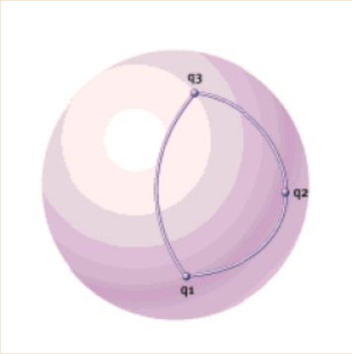
Hyperspheres

[Animation of a hypersphere]

- Now you’re moving on the “surface” of a hypersphere
- You now have three orthogonal directions in which to move
  - Forward/backward, left/right, up/down
- If you move too far in any one direction, you loop back to where you began
  - Less intuitive in the up/down direction, but that’s four dimensional shapes for you!

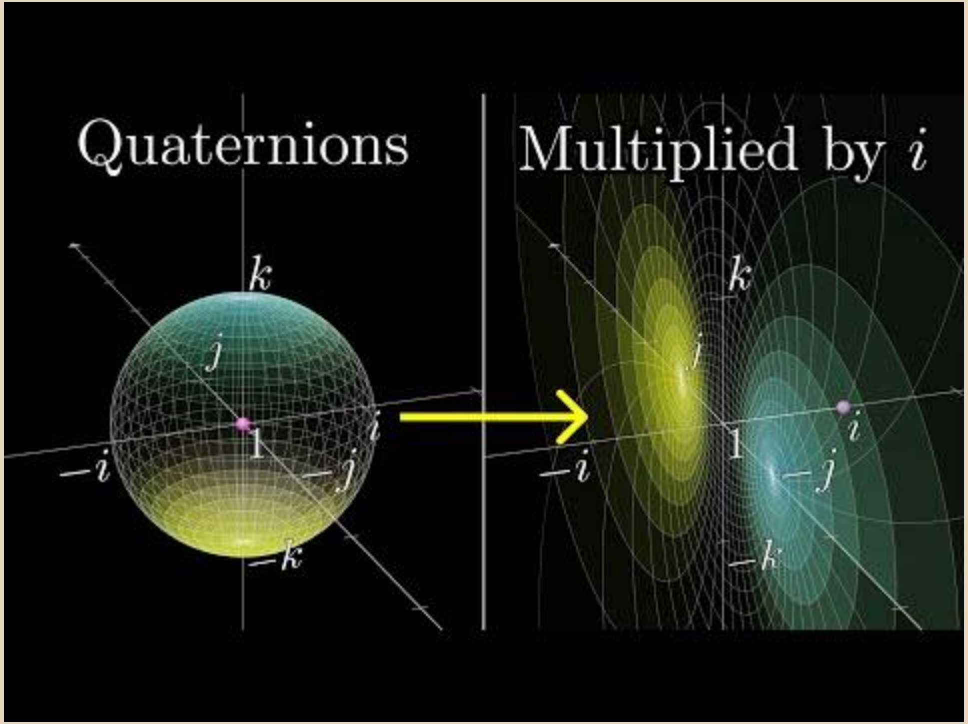
Hyperspheres

- Use our location on that hypersphere to represent an orientation
- Distances along the hypersphere correspond to angles in 3D space
- Moving in some arbitrary direction corresponds to rotating about some arbitrary axis



Useful video on quaternions

Take some time to watch this outside of class



Hypersphere distance to rotation

- A distance of x on a hypersphere corresponds to a rotation angle of $2x$ radians
    - e.g. moving along a 180 degree arc is equivalent to rotating by 360 degrees
- This is why $-q$ and q represent the same rotation
    - Also makes sense because $-q$ and q define the same axis, just in opposite directions
- [Demonstration]

Arcball rotation

- The demo you just saw was an implementation of Ken Shoemake’s Arcball Rotation paper
- It maps mouse cursor movement to a quaternion rotation, treating the length and width of the view screen as full rotations

Quaternion Interpolation

- Given two orientations, we want to interpolate between the two for animation purposes
- Can't just linearly interpolate
  - $(1-t)*q1 + t*q2$ doesn't give us the right numbers
- Use Spherical Linear Interpolation (SLERP)

$$\mathrm{slerp}(t, q_1, q_2) = \frac{\sin((1-t)\theta)}{\sin(\theta)} q_1 + \frac{\sin(t\theta)}{\sin\theta} q_2$$

- $t = [0,1]$ and $\theta = cos^{-1}(dot(q1, q2))$