Clipping

• analytically calculating the portions of primitives within the viewport

Cohen-Sutherland Line Clipping

• outcodes
  – 4 flags encoding position of a point relative to top, bottom, left, and right boundary
  – $OC(p1) = 0010$
  – $OC(p2) = 0000$
  – $OC(p3) = 1001$

Clipping Lines To Viewport

• combining trivial accepts/rejects
  – trivially accept lines with both endpoints inside all edges of the viewport
  – trivially reject lines with both endpoints outside the same edge of the viewport
  – otherwise, reduce to trivial cases by splitting into two segments

Polygon Clipping

• not just clipping all boundary lines
  – may have to introduce new line segments
Sutherland-Hodgeman Clipping

• basic idea:
  – consider each edge of the viewport individually
  – clip the polygon against the edge equation
  – after doing all edges, the polygon is fully clipped

Sutherland-Hodgeman Clipping

• basic idea:
  – consider each edge of the viewport individually
  – clip the polygon against the edge equation
  – after doing all edges, the polygon is fully clipped

Sutherland-Hodgeman Clipping

• basic idea:
  – consider each edge of the viewport individually
  – clip the polygon against the edge equation
  – after doing all edges, the polygon is fully clipped

Sutherland-Hodgeman Clipping

• edge from \( s \) to \( p \) takes one of four cases:
  (blue line can be a line or a plane)

University of British Columbia
CPSC 414 Computer Graphics

Rotations and Quaternions

Camera Movement Hints

• change viewing transformation
  – don’t try doing this with perspective xform!
• methods
  – gluLookAt
  – direct camera control using rotate/translate
• camera motion opposite of object motion
  – rotate world by \( a = \) orbit camera by \(-a\)
Parameterizing Rotations

- Straightforward in 2D
  - A scalar, \( \theta \), represents rotation in plane
- More complicated in 3D
  - Three scalars are required to define orientation
  - Note that three scalars are also required to define position
  - Objects free to translate and tumble in 3D have 6 degrees of freedom (DOF)

Representing 3 Rotational DOFs

- 3x3 Matrix (9 DOFs)
  - Rows of matrix define orthogonal axes
- Euler Angles (3 DOFs)
  - \( \text{Rot } x + \text{Rot } y + \text{Rot } z \)
- Axis-angle (4 DOFs)
  - Axis of rotation + Rotation amount
- Quaternion (4 DOFs)
  - 4 dimensional complex numbers

3x3 Rotation Matrix

- 9 DOFs must reduce to 3
- Rows must be unit length (-3 DOFs)
- Rows must be orthogonal (-3 DOFs)
- Drifting matrices is very bad
  - Numerical errors results when trying to gradually rotate matrix by adding derivatives
  - Resulting matrix may scale / shear
  - Gram-Schmidt algorithm will re-orthogonalize
- Difficult to interpolate between matrices
  - How would you do it?

Rotation Matrix

- general rotation can be represented by a single 3x3 matrix
  - length preserving (isometric)
  - reflection preserving
  - orthonormal
- problem:
  - property: rows and columns are orthonormal (unit length and perpendicular to each other)
  - linear interpolation doesn’t maintain this property

Rotation Matrices Not Interpolatable

- interpolate linearly from +90 to -90 in y
  \[
  \begin{pmatrix}
  0 & 0 & 1 \\
  0 & 1 & 0 \\
  -1 & 0 & 0
  \end{pmatrix}
  \]
- halfway through component interpolation
  \[
  \begin{pmatrix}
  0 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 0
  \end{pmatrix}
  \]
- problem 1: not a rotation matrix anymore!
  - not orthonormal, x flattened out

Euler Angles

- \((\theta_x, \theta_y, \theta_z) = R_x R_y R_z\)
  - Rotate \( \theta_x \) degrees about x-axis
  - Rotate \( \theta_y \) degrees about y-axis
  - Rotate \( \theta_z \) degrees about z-axis
- Axis order is not defined
  - \((y, z, x), (x, z, y), (z, y, x)\) all legal
  - Pick one
**Euler Angle Interpolation**

- **solution 1:** can interpolate angles individually
- **problem 2:** interpolation between two Euler angles is not unique
  - ex: (x, y, z) rotation
    - (0, 0, 0) to (180, 0, 0) vs. (0, 0, 0) to (0, 180, 180)
    - interpolation about different axes are not independent
    - Cartesian coordinates are independent of one another, but Euler angles are not

---

**Interpolation**

- **Problem 3:** Gimbal Lock
  - term derived from mechanical problem that arises in gimbal mechanism that supports a compass or a gyro
  - gimbal: hardware implementation of Euler angles (used for mounting gyroscopes and globes)

---

**Gimbal Lock**

- Occurs when two axes are aligned
- Second and third rotations have effect of transforming earlier rotations
  - If Rot y = 90 degrees, Rot z == -Rot x

[demo]
Locked Gimbal

- Hardware implementation of Euler angles (used for mounting gyroscopes and globes)

Axis-angle Rotation

- A counter-clockwise (right-handed) rotation $\theta$ about the axis specified by the unit vector $n=(n_x, n_y, n_z)$

Axis-angle Notation

- Define an axis of rotation $(x, y, z)$ and a rotation about that axis, $\theta$: $R(\theta, n)$

- 4 degrees of freedom specify 3 rotational degrees of freedom because axis of rotation is constrained to be a unit vector

Angular displacement

- $(\theta, n)$ defines an angular displacement of $\theta$ about an axis $n$

$$ v_{par} = (n \cdot v) n $$
$$ R[v_{par}] = v_{par} \cos \theta + (n \times v_{par}) \sin \theta $$
$$ R[v_{par}] = v_{par} \cos \theta + (n \times v_{par}) \sin \theta $$

Axis-angle Rotation

- Step 1
  - Compute $v_{par}$, an extended version of the rotation axis $n$
  $$ v_{par} = (n \cdot v) n $$

- Given
  - $r$ – Vector in space to rotate
  - $n$ – Unit-length axis in space about which to rotate
  - $\theta$ – The amount about $n$ to rotate

- Solve
  - $r'$ – The rotated vector
Axis-angle Rotation

• Compute \( r_{\text{perp}} \)

\[
 r_{\text{perp}} = r - r_{\text{par}} = r - (n \cdot r)n
\]

Angular Displacement

• \( R(\theta, n) \) is the rotation matrix to apply to a vector \( v \), then,

\[
 R[v] = v\cos\theta + n(n.v)(1-\cos\theta) + (n\times v)\sin\theta
\]

− It guarantees a simple steady rotation between any two key orientations
− It defines moves that are independent of the choice of the coordinate system

Axis-angle Notation

• solutions
  − any orientation can be represented by a 4-tuple: angle, vector(x,y,z)
  − can interpolate the angle and axis separately
  − no gimbal lock problems!

• problems
  − no easy way to determine how to concatenate many axis-angle rotations that result in final desired axis-angle rotation
  − so can’t efficiently compose rotation, must convert to matrices first!

Quaternions

• extend the concept of rotation in 3D to 4D
• avoids the problem of "gimbal-lock" and allows for the implementation of smooth and continuous rotation
• in effect, they may be considered to add an additional rotation angle to spherical coordinates i.e. longitude, latitude and rotation angles
• a quaternion is defined using four floating point values \( |x \ y \ z \ w| \). These are calculated from the combination of the three coordinates of the rotation axis and the rotation angle.

Quaternions Definition

• Extension of complex numbers
• 4-tuple of real numbers
  − \( s,x,y,z \) or \( [s,v] \)
  − \( s \) is a scalar
  − \( v \) is a vector
• Same information as axis/angle but in a different form
• Can be viewed as an original orientation or a rotation to apply to an object
**Quaternion**

- **Extension of complex numbers:** $a + ib$
  - Remember $i^2 = -1$
- **Quaternion:**
  - $Q = a + bi + cj + dk$
  - Where $i^2 = j^2 = k^2 = -1$ and $ij = k$ and $ji = -k$
  - Represented as: $q = (s, v) = s + v_i + v_j + v_k$
- Invented by Sir William Hamilton (1843)
  - Carved equation into Dublin bridge when discovered after decade of work

**Quaternion**

- A quaternion is a 4-D unit vector $q = [x \ y \ z \ w]$
  - It lies on the unit hypersphere $x^2 + y^2 + z^2 + w^2 = 1$
- For rotation about (unit) axis $v$ by angle $\theta$
  - Vector part = $(\sin(\theta/2), v)$
  - Scalar part = $(\cos(\theta/2), w)$
  - $(\sin(\theta/2) n_x, \sin(\theta/2) n_y, \sin(\theta/2) n_z, \cos(\theta/2))$
  - Only a unit quaternion encodes a rotation
    - Must normalize!

**Quaternion Example**

- Rotation matrix corresponding to a quaternion:
  - $[x \ y \ z \ w]$
    - $[-2x^2 - 2z^2 \ 2xy + 2wz \ 2xz - 2yw \ 2yz + 2wx \ 2xy - 2wz \ -2x^2 - 2y^2 \ 2xz + 2yw \ 2yz - 2wx \ 2yz + 2wx \ -2x^2 - 2y^2 \ 2xz - 2yw \ 2yz + 2wx \ 2xy + 2wz \ -2x^2 - 2z^2 \ 2xz + 2yw \ -2x^2 - 2y^2]$  

  - Quaternion Multiplication
  - $q_1 * q_2 = [v_1, w_1] * [v_2, w_2] = [w_1v_2 + w_2v_1 + (v_1 \times v_2), w_1w_2 - v_1 \cdot v_2]$
  - Quaternion * quaternion = quaternion
  - This satisfies requirements for mathematical group
  - Rotating object twice according to two different quaternions is equivalent to one rotation according to product of two quaternions

**Quaternion Interpolation**

- Biggest advantage of quaternions
  - Cannot linearly interpolate (lerp) between two quaternions because it would speed up in middle
  - Instead, spherical linear interpolation, (slerp)
    - Ensure vectors remain on the hypersphere
    - Step through using constant angles

**SLERP**

- Quaternion is a point on the 4-D unit sphere
  - Interpolating rotations requires a unit quaternion at each step
    - Another point on the 4-D unit sphere
  - Move with constant angular velocity along the great circle between two points
  - Any rotation is defined by 2 quaternions, so pick the shortest SLERP
  - To interpolate more than two points, solve a non-linear variational constrained optimization
    - Ken Shoemake in SIGGRAPH ’85 (www.acm.org/dl)
Quaternion Libraries

- Gamasutra
  - Code, explanatory article
  - Registration required

  http://www.gamasutra.com/features/19980703/quaternions_01.htm

Evaluating Quaternions

- Advantages:
  - Flexible.
  - No parametrization singularities (gimbal lock)
  - Smooth consistent interpolation of orientations.
  - Simple and efficient composition of rotations.

- Disadvantages:
  - Each orientation is represented by two quaternions.
  - Complex!

Summary

- 3x3 matrices
  - Drifting, can't interpolate
- Euler angles
  - Gimbal lock
- Axis-angle
  - Can't concatenate or interpolate
- Quaternions
  - Solve all problems, but complex

Project Strategy Suggestion

- Debug basics with simple euler angles
  - With single drag, does view change the right way?

- Then can add quaternions