Rotations (and other transformations)

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

\[
R(a, \theta)x = (\cos \theta)x + (\sin \theta)(a \times x) + (1 - \cos \theta)(a \cdot x)a
\]

\[
R(a, \theta) = (\cos \theta)I + (\sin \theta)\hat{a} + (1 - \cos \theta)a\hat{a}^T
\]

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 + \(\cos/\sin/\text{sqrt} \) 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

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

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

\[ i * j = -j * i = k \]
\[ k * i = -i * k = j \]
\[ j * k = -k * j = i \]
### 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**
  \[\|q\| = 1\]
  \[q^{-1} = q^*\]

### Quaternion for Rotation

- **Linear combination of** \(1, i, j, k\)
  \[q = w + xi + yj + zk = (s, v)\]
  \[s = w\]
  \[v = [x \ y \ z]\]

- **Multiplication**
  \[q_1 = (s_1, v_1)\]
  \[q_2 = (s_2, v_2)\]
  \[q_1 * q_2 = (s_1s_2 - v_1.v_2, s_1v_2 + s_2v_1 + v_1 \times v_2)\]

### Rotation Using Quaternion

- **A point in space is a quaternion with 0 scalar**
  \[X = (0, \vec{x})\]
  \[x_{\text{rotated}} = qXq^{-1}, \text{where,} \quad q^{-1} = q^*\]

- **Composing rotations**
  q1 and q2 are two rotations
  First, q1 then q2
  \[x_{\text{rotated}} = q_2 * (q_1Xq_1^{-1}) * q_2^{-1}\]
  \[x_{\text{rotated}} = (q_2 * q_1)X(q_1^{-1} * q_2^{-1})\]
  \[x_{\text{rotated}} = (q_2 * q_1)X(q_2 * q_1)^{-1}\]
Matrix for quaternion

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

Quaternion spline interpolation

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
**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 \(a = (P0-O) \times (P1-O)\)
- Angle \(\theta = k ||P1-P0||\)
- Rotate by \(\theta\) around \(a\)
- \(P_0\) (next frame) = \(P_1\)

Trackball Terms

- \(P_0 = (x_0, y_0, z_0)\)
- \(P_1 = (x_1, y_1, z_1)\)
- \((x-O_x)^2 + (y-O_y)^2 + (z-O_z)^2 = R^2\)
- \(D = P_1-P_0\)
- \(\text{axis}_\text{rotation} = (P1-O) \times (P1-O)\)
- \(\text{angle}_\text{rotation} = k ||D||\)

Trackball Terms

- \(Z-O_z = \sqrt{R^2-(x-O_x)^2 + (y-O_y)^2}\)
- \(O\) is arbitrary
- \(O = (x_{res}/2, y_{res}/2, z_c)\)