OpenGL Matrix Stacks

• OpenGL has modelview, projection, and texture matrix stacks
• allows for convenient management of hierarchical transformations
• stacks initialized with identity matrix

glMatrixMode
glPushMatrix
glPopMatrix
glLoadIdentity
glMatrixMode
glPushMatrix
glPopMatrix
glLoadIdentity
OpenGL Transformations

• Multiply the current matrix by the associated transformation matrix and
• replace the current matrix by the resulting product

  glTranslate
  glRotate
  glScale

  glOrtho
  glFrustum
OpenGL Drawing

```cpp
glBegin(GL_TRIANGLES);
 glVertex2f(0.25, 0.25);
 glVertex2f(0.75, 0.25);
 glVertex2f(0.75, 0.75);
 glEnd();
```
general rotations
Rotation

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \theta & -\sin \theta & 0 \\
0 & \sin \theta & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

**X axis**

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} =
\begin{bmatrix}
\cos \theta & 0 & \sin \theta & 0 \\
0 & 1 & 0 & 0 \\
-\sin \theta & 0 & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

**Y axis**

\[
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} =
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 & 0 \\
\sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

**Z axis**

The rows and columns are orthonormal
Rotation about an arbitrary axis

Rotating about an axis by theta degrees

• Rotate about x to bring axis to xz plane
• Rotate about y to align axis with z-axis
• Rotate theta degrees about z
• Unrotate about y, unrotate about x

\[ M = R_x^{-1} R_y^{-1} R_z(\theta) R_y R_x \]

• Can you determine the values of Rx and Ry?
Composite Transformations

- Rotating about a fixed point
- **basic** rotation alone will rotate about origin but we want:
Composite Transformations

- Rotating about a fixed point
- Move fixed point \((p_x, p_y, p_z)\) to origin
- Rotate by desired amount
- Move fixed point back to original position

\[
M = T(p_x, p_y, p_z) R_z(\theta) T(-p_x, -p_y, -p_z)
\]
euler angles
Euler Angles

3 Euler angles can be used to specify an arbitrary orientation

Three Euler angles can be used to specify arbitrary object orientation through a sequence of three rotations around coordinate axes embedded into the object.

Shirley and Marschner
Three Euler angles can be used to specify an arbitrary orientation through a sequence of three rotations around coordinate axes embedded into the object.
Euler Angles

- A general rotation is a combination of three elementary rotations:
  - around the x-axis (roll)
  - around the y-axis (pitch)
  - around the z-axis (yaw)
Gimbal and Euler Angles

Z-X’-Z”

Wikimedia Commons
**intrinsic**
- rotations about the object fixed axes
  - first rotate **outer** gimbal, then **middle** gimbal, finally **inner** gimbal

**extrinsic**
- rotations about the reference axes
  - first rotate **inner** gimbal, then **middle** gimbal, finally **outer** gimbal
Euler Angles and Rotation Matrices

\[
x\text{-roll}(\theta_1) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta_1 & -\sin \theta_1 & 0 \\ 0 & \sin \theta_1 & \cos \theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad y\text{-pitch}(\theta_2) = \begin{pmatrix} \cos \theta_2 & 0 & \sin \theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta_2 & 0 & \cos \theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\]

\[
z\text{-yaw}(\theta_3) = \begin{pmatrix} \cos \theta_3 & -\sin \theta_3 & 0 & 0 \\ \sin \theta_3 & \cos \theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\]

\[
R(\theta_1, \theta_2, \theta_3) = \begin{pmatrix} c_2 c_3 & -c_2 s_3 & s_2 & 0 \\ s_1 s_2 c_3 + c_1 s_3 & -s_1 s_2 s_3 + c_1 c_3 & -s_1 c_2 & 0 \\ -c_1 s_2 c_3 + s_1 s_3 & c_1 s_2 s_3 + s_1 c_3 & c_1 c_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\]
Interpolating Rotations

- naive element-by-element interpolation of rotation matrices does not work! E.g.,

\[ \frac{1}{2} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} + \frac{1}{2} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \]

90 CW + 90 CCW

- We can interpolate Euler angles, but there is a problem known as **Gimbal Lock**
http://www.youtube.com/watch?v=zc8b2Jo7mno
quaternions
Here as he walked by on the 16th of October 1843 Sir William Rowan Hamilton in a flash of genius discovered the fundamental formula for quaternion multiplication

\[ i^2 = j^2 = k^2 = ijk = -1 \]

cut on a stone on the bridge
Quaternions

- axis/angle representation
- interpolates smoothly
- easy to compose

<whiteboard>
Quaternion Interpolation

linear: treat quaternions as 4-vectors, note non-uniform speed
spherical linear: constant speed
Matrix form

\[ q = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} \]

\[ R(q) = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy + 2wz & 2xz - 2wy & 0 \\ 2xy - 2wz & 1 - 2x^2 - 2z^2 & 2yz + 2wx & 0 \\ 2xz + 2wy & 2yz - 2wx & 1 - 2x^2 - 2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
Rotations in Reality

- It’s easiest to express rotations in Euler angles or Axis/angle

- We can convert to/from any of these representations

- Choose the best representation for the task
  - input: Euler angles
  - interpolation: quaternions
  - composing rotations: quaternions, orientation matrix