Chapter 4:
Transformations- Transforming Normals, Hierarchies and OpenGL, Assignment 2
Transformations in OpenGL
The Rendering Pipeline

- Geometric Content
- Model/View Transform.
- Lighting
- Perspective Transform.
- Clipping
- Scan Conversion
- Texturing
- Depth Test
- Blending
- Framebuffer

Geometry Processing

Rasterization

Fragment Processing
Modeling Transformation

- Purpose:
  - Map geometry from local object coordinate system into a global world coordinate system
  - Same as placing objects
  - Hardware support for arbitrary affine transformations
Viewing Transformation

- Purpose:
  - Map geometry from *world coordinate system* into *camera coordinate system*
    - Camera coordinate system is *right-handed*, viewing direction is *negative z-axis*
  - Same as placing camera
- Transformations:
  - Usually only *rigid body transformations*
    - Rotations and translations
  - Objects have same size and shape in camera and world coordinates
Model/View Transformation

- Combine modeling and viewing transform
  - Combine into single matrix

- Saves computation time
  - if many points are to be transformed

- Possible because viewing transformation directly follows modeling transformation without intermediate operations
Transformations in OpenGL

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glBegin(GL_LINE_LOOP);
  glVertex2f(0, 0);
  glVertex2f(2, 0);
  glVertex2f(2, 2);
  glVertex2f(1, 3);
  glVertex2f(0, 2);
glEnd();
```

DrawHouse()
Transformations in OpenGL

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}_w = \begin{bmatrix}
  2 & 0 & 0 & 3 \\
  0 & 2 & 0 & 1 \\
  0 & 0 & 2 & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}_w
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}_{obj}
\]

GLfloat T[16] = {2,0,0,0,0,2,0,0,0,0,2,0,3,1,0,1};

glMatrixMode(GL_MODELVIEW);

Matrixf(T);

DrawHouse();
Transformations in OpenGL

- An easier way to do the same thing....

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glTranslatef(3,1,0);
glScalef(2,2,2);

DrawHouse();
```
Composing Transformations

suppose we want

\[ P_A = \text{Rot}(z,-90) P_h \]
\[ P_w = \text{Trans}(2,3,0) P_A \]
\[ P_w = \text{Trans}(2,3,0) \text{Rot}(z, -90) P_h \]
Composing Transformations

\[ P_w = Trans(2,3,0) \cdot Rot(z,-90) \cdot P_n \]

- R-to-L: interpret operations wrt fixed coords
  - moving object
- L-to-R: interpret operations wrt local coords
  - changing coordinate system
- OpenGL (L-to-R, local coords)

\[
M_{MV} = Trans(2,3,0) \cdot M_{MV} \\
M_{MV} = Rot(z,-90) \cdot M_{MV}
\]

updates current transformation matrix by postmultiplying
Post Multiplication

- Composite transformation = matrix product

- Rather than multiply each point sequentially with 3 matrices, first multiply the matrices, then multiply each point with only one (composite) matrix

- Much faster for large # of points!
- Same reason to use homogeneous coordinates
Interpreting Composite OpenGL Transformations

- Example from earlier lectures:
  - Rotation around arbitrary center
  - In OpenGL:

  ```c
  // initialization of matrix
  glMatrixMode( GL_MODELVIEW );
  glLoadIdentity();

  glTranslatef( 4.3 );
  glRotatef( 30.0, 0.0, 0.0, 1.0 );
  glTranslatef( -4.0, -3.0 );
  glBegin( GL_TRIANGLES );
  // specify object geometry...
  ```

Top-to-bottom: transf. of coordinate frame
Bottom-to-top: transf. of object
Matrix Operations in OpenGL

- 2 Matrices:
  - Model/view matrix $M$
  - Projective matrix $P$

- Example:

  ```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity(); // M=Id
glRotatef( angle, x, y, z ); // M= R(\alpha)*Id
glTranslatef( x, y, z ); // M= T(x,y,z)*R(\alpha)*Id
glMatrixMode( GL_PROJECTION );
glRotatef( ... ); // P= ... 
```
Transformation Hierarchies
Transformation Hierarchies

- Scenes have multiple coordinate systems
  - Often strongly related
    - Parts of the body
    - Object on top of each other
      - Next to each other...
  - Independent definition is bug prone
- Solution: Transformation Hierarchies
Transformation Hierarchy Examples

```
glTranslatef(x, y, 0);
glRotatef(θ1, 0, 0, 1);
DrawBody();
glTranslatef(2.5, 5.5, 0);
glRotatef(θ2, 0, 0, 1);
DrawUArm();
glTranslatef(0, -3.5, 0);
glRotatef(θ3, 0, 0, 1);
DrawLArm();
```
Matrix Stacks

D = C \text{ scale}(2,2,2) \text{ trans}(1,0,0)

gl\text{PushMatrix}()
gl\text{PopMatrix}()

\begin{array}{ccc}
C & C & C \\
B & B & B \\
A & A & A \\
\end{array}

\begin{array}{ccc}
D & C \\
B & B \\
A & A \\
\end{array}

\text{DrawSquare}()
gl\text{PushMatrix}()
gl\text{Scale3f}(2,2,2)

\text{gl\text{Translate3f}(1,0,0)}

\text{DrawSquare}()
gl\text{PopMatrix}()
glTranslatef(x,y,0);
glRotatef(θ₁,0,0,1);
DrawBody();
glPushMatrix();
glTranslatef(2.5,5.5,0);
glRotatef(θ₂,0,0,1);
DrawUArm();
glTranslatef(0,-3.5,0);
glRotatef(θ₃,0,0,1);
DrawLArm();
glPopMatrix();
glPushMatrix();
glTranslatef(0,7,0);
DrawHead();
glPopMatrix();
... (draw other arm)
Transformation Hierarchies

world

<table>
<thead>
<tr>
<th>torso</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUleg</td>
</tr>
<tr>
<td>RUleg</td>
</tr>
<tr>
<td>LUarm</td>
</tr>
<tr>
<td>RUarm</td>
</tr>
<tr>
<td>head</td>
</tr>
<tr>
<td>LLeg</td>
</tr>
<tr>
<td>RLeg</td>
</tr>
<tr>
<td>LLarm</td>
</tr>
<tr>
<td>RLarm</td>
</tr>
<tr>
<td>Lhand</td>
</tr>
<tr>
<td>Rhand</td>
</tr>
</tbody>
</table>

rot(z, $\theta$) trans(0.30, 0, 0)
Matrix Stacks

- Advantages
  - No need to compute inverse matrices all the time
  - Modularize changes to pipeline state
  - Avoids incremental changes to coordinate systems
    - Accumulation of numerical errors
- Practical issues
  - In graphics hardware, depth of matrix stacks is limited
    - Typically 16 for model/view and ~4 for projective matrix
glLoadIdentity();
glTranslatef(4,1,0);
glPushMatrix();
glRotatef(45,0,0,1);
glTranslatef(0,2,0);
glScalef(2,1,1);
glTranslatef(1,0,0);
glPopMatrix();
Hierarchical Modeling

- Advantages
  - Define object once, instantiate multiple copies
  - Transformation parameters often good control knobs
  - Maintain structural constraints if well-designed
- Limitations
  - Expressivity: not always the best controls
  - Can’t do closed kinematic chains
    - Keep hand on hip
Transforming Normals
Computing Normals

- polygon:
  \[ N = \frac{(P_2 - P_1) \times (P_3 - P_1)}{\left\| (P_2 - P_1) \times (P_3 - P_1) \right\|} \]

- assume vertices ordered CCW when viewed from visible side of polygon

- normal for a vertex
  - used for lighting
  - supplied by model (i.e., sphere), or computed from neighboring polygons
Transforming Normals

- When transforming triangle(s) can we use the same transformation to transform the normal & avoid re-computation?
- What is a normal?
  - **Vector**
    - Orthogonal (perpendicular) to plane/surface
- Do standard transformations preserve orthogonality?
  - Or angles in general?
Planes and Normals

- Plane - all points where \( N \cdot P = 0 \)

\[
P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \quad N = \begin{bmatrix} A \\ B \\ C \\ 0 \end{bmatrix}
\]

- Implicit form

\[
Plane = A \cdot x + B \cdot y + C \cdot z + D
\]
Finding Correct Normal Transform

- transform a plane

\[
\begin{align*}
P & \rightarrow P' = MP \\
N & \rightarrow N' = QN
\end{align*}
\]

Given \( M \), find \( Q \)

\( N^T P' = 0 \) \quad stay perpendicular

\( (QN)^T (MP) = 0 \) \quad substitute from above

\( N^T Q^T MP = 0 \)

\( Q^T M = I \)

\( Q = (M^{-1})^T \) \quad Normal transformed by transpose of the inverse of the modeling transformation
Transformation properties
<table>
<thead>
<tr>
<th>Transformation</th>
<th>Straight lines</th>
<th>Parallel lines</th>
<th>Distance</th>
<th>Angles</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform scaling</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>non-uniform scaling</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rotation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>translation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shear</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>