Chapter 4: Transformations - Transforming Normals, Hierarchies and OpenGL, Assignment 2

Transforming Normals

- 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

\[
Plane = A \cdot x + B \cdot y + C \cdot z + D
\]

Computing Normals

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

Finding Correct Normal Transform

- transform a plane

\[
P \quad N \quad P' = MP \quad N' = QN
\]

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

- stay perpendicular
  - substitute from above

\[
(N^T P') = 0 \\
(QN)^T (MP) = 0 \\
N^T Q^T MP = 0 \\
Q^T M = I
\]

Normal transformed by transpose of the inverse of the modeling transformation

Copyright 2012, A. Sheffer, UBC
Transformations in OpenGL

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

The Rendering Pipeline

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

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

Modeling Transformation

Purpose:
- Map geometry from local object coordinate system into a global world coordinate system
- Same as placing objects

Transformations:
- Arbitrary affine transformations are possible
- More complex transformations may be desirable
  - Freeform deformations
  - Not available in hardware

Copyright 2012, A. Sheffer, UBC
Transformations in OpenGL

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

GLfloat T[16] = { 2,0,0,0, 0,2,0,0, 0,0,2,0, 3,1,0,1};
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
DrawHouse();

Transformations in OpenGL

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

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(3,1,0);
DrawHouse();

Transformations in OpenGL

Matrix Operations in OpenGL

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

Example:

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(x, y, z);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

Composing Transformations

\( R_\text{e} = \text{Trans}(2,3,0) \cdot \text{Rot}(z,-90) \cdot R_\text{i} \)

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

\[
M_{\text{MVP}} = \text{Trans}(2,3,0) \cdot M_{\text{MVP}}
\]

Post Multiplication

- Composite transformation is now just the product of a few matrices
- 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

Copyright 2012, A. Sheffer, UBC
Interpreting Composite OpenGL Transformations

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

  ```
  // initialization of matrix
  glMatrixMode( GL_MODELVIEW );
  glLoadIdentity();
  glTranslatef( 4, 3 );
  glRotatef( 30, 0.0, 0.0, 1.0 );
  glTranslatef( -4, -3 );
  glBegin( GL_TRIANGLES );
  // specify object geometry...
  ```

Top-to-bottom:
transf. of coordinate frame

Bottom-to-top:
transf. of object

Transformation Hierarchies

- Scene may have a hierarchy of coordinate systems
- Multiple objects, multiple joint links, ...
- Stores matrix at each level with incremental transform from parent's coordinate system

Check out: Brown Applets

http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/scenegraphs.html

Matrix Stacks

- Avoiding unnecessary computation when incremental processing makes no sense
- Using inverse to return to origin costs...

Copyright 2012, A. Sheffer, UBC
Matrix Stacks

- D = C scale(2,2,2) trans(1,0,0)

Matrix Stacks

- glPushMatrix()
- glPopMatrix()

Modularization

- Drawing a scaled square
- Push/pop ensures no coord system change

void drawBlock(float k) {
  glPushMatrix();
  glScalef(k,k,k);
  glBegin(GL_LINE_LOOP);
  glVertex3f(0,0,0);
  glVertex3f(1,0,0);
  glVertex3f(1,1,0);
  glVertex3f(0,1,0);
  glEnd();
  glPopMatrix();
}

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

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

Copyright 2012, A. Sheffer, UBC
Assignment 2

Out this week, due **4pm Fri Oct 12, 2012**
- Start very soon!
- Build and animate a robot made out of cubes and 4x4 matrices
  - think cartoon, not beauty
- Template code - program shell, Makefile

Advice

- **Draw one section at a time**
  - Ensure you're constructing hierarchy correctly
  - Use body as scene graph root
  - Continue with attached parts
  - Finish all required parts before...
    - Adding extra links or DOFs
    - Going for extra credit
  - Visual debugging
    - Draw the current coord system

Advanced transformations example

- Deformation Transfer [Sumner'05]
  - Use transformation gradients (transformation without translation) as per-triangle encoding of motion

Copyright 2012, A. Sheffer, UBC