CS 4731: Computer Graphics
Lecture 9: Introduction to 3D Modeling

3D Modeling
- Overview of OpenGL modeling (Hill 5.6)
- Modeling: create 3D model of scene/objects
- OpenGL commands
  - Coordinate systems (left hand, right hand, OpenGL-way)
  - Basic shapes (cone, cylinder, etc)
  - Transformations/Matrices
  - Lighting/Materials
  - Synthetic camera basics
  - View volume
  - Projection
- GLUT models (wireframe/solid)
- Scene Description Language (SDL): 3D file format

Coordinate Systems
- Recall:

Right hand coordinate system
- Not used in this class and
- Not in OpenGL

3D Modeling: GLUT Models
- Two main categories:
  - Wireframe Models
  - Solid Models
- Basic Shapes
  - Cylinder: glutWireCylinder(), glutSolidCylinder()
  - Cone: glutWireCone(), glutSolidCone()
  - Sphere: glutWireSphere(), glutSolidSphere()
  - Cube: glutWireCube(), glutSolidCube()
- More advanced shapes:
  - Newell Teapot: (symbolic)
  - Dodecahedron, Torus
GLUT Models: glutwireTeapot( )

- The famous Utah Teapot has become an unofficial computer graphics mascot

```cpp
glutWireTeapot(0.5) -
Create a teapot with size 0.5, and position its center at (0,0,0)
Also: glutSolidTeapot()"
```

Again, you need to apply transformations to position it at the right spot.

3D Modeling: GLUT Models

- Without GLUT models:
  - Use generating functions
  - More work!
  - Example: Look in examples bounce, gears, etc.
- What does it look like?
  - Generates a list of points and polygons for simple shapes
  - Spheres/Cubes/Sphere

Cylinder Algorithm

```cpp
gBegin(GL_QUADS)
    For each A = Angles{
        glVertex3f(R*cos(A), R*sin(A), 0);
        glVertex3f(R*cos(A+DA), R*sin(A+DA), 0)
        glVertex3f(R*cos(A+DA), R*sin(A+DA), H)
        glVertex3f(R*cos(A), R*sin(A), H)
    }
// Make Polygon of Top/Bottom of cylinder
```

3D Transforms

- Scale:
  - glScaled(sx, sy, sz) - scale object by (sx, sy, sz)
- Translate:
  - glTranslated(dx, dy, dz) - translate object by (dx, dy, dz)
- Rotate:
  - glRotated(angle, ux, uy, uz) - rotate by angle about an axis passing through origin and (ux, uy, uz)
OpenGL Matrices

- Modelview matrix:
- Combination of modeling matrix $M$ and Camera transforms $V$

OpenGL Matrices/Pipeline

- OpenGL uses 3 matrices:
  - Modelview matrix:
  - Projection matrix:
  - Viewport matrix:
- Modelview matrix:
  - combination of modeling matrix $M$ and Camera transforms $V$

Projection matrix:
- Scales and shifts each vertex in a particular way.
- View volume lies inside cube of $-1$ to $1$
- Reverses sense of $z$: increasing $z = $ increasing depth
- Effectively squishes view volume down to cube centered at $1$
- Clipping then eliminates portions outside view volume

Viewport matrix:
- Maps surviving portion of block (cube) into a 3D viewport
- Retains a measure of the depth of a point

Lighting and Object Materials

- Light components:
  - Diffuse, ambient, specular
  - OpenGL: glLightfv( ), glLightf( )
- Materials:
  - OpenGL: glMaterialf( ), glMaterialf( )
Synthetic Camera

- Define:
  - Eye position
  - LookAt point
  - Up vector (if spinning: confusing)
- Programmer knows scene, chooses:
  - eye
  - lookAt
- Up direction usually set to (0,1,0)
- OpenGL:
  - gluLookAt(eye.x, eye.y, eye.z, look.x, look.y, look.z, up.x, up.y, up.z)

View Volume

- Side walls determined by window borders
- Other walls determined by programmer-defined
  - Near plane
  - Far plane
- Convert 3D models to 2D:
  - Project points/vertices inside view volume unto view window using parallel lines along z-axis

Projection

- Different types of projections:
  - Parallel
  - Perspective
  - Parallel is simple
  - Will use for this intro, expand later
Hierarchical Transforms Using OpenGL

- Object dependency
- Graphical scene: many small objects
- Attributes (position, orientation, etc.) depend on each other

Attributes (position, orientation, etc.) depend on each other.

Object dependency description using tree structure

Root node

Object position and orientation can be affected by its parent, grand-parent, grand-grand-parent nodes.

Leaf node

Hierarchical representation is known as Scene Graph.

Transformations

- Two ways to specify transformations:
  - (1) Absolute transformation: each part of the object is transformed independently relative to the origin
  - (2) Relative transformation: Specify the transformation for each object relative to its parent

Translate the base by (5, 0, 0);
Translate the lower arm by (5, 0, 0);
Translate the upper arm by (5, 0, 0);

Relative Transformation

A better (and easier) way:

(2) Relative transformation: Specify the transformation for each object relative to its parent

Step 1: Translate base and its descendants by (5, 0, 0);
Relative Transformation

Step 2: Rotate the lower arm and all its descendants relative to the base's local y axis by -90 degree

Hierarchical Transforms Using OpenGL

- Translate base and all its descendants by (5,0,0)
- Rotate the lower arm and its descendants by -90 degree about the local y

SDL

SDL data structure format
Hierarchical Models

- SDL makes hierarchical modeling easy
- Without openGL: a little tougher
- Two important calls:
  - `glPushMatrix()`: load transform matrix with following matrices
  - `glPopMatrix()`: restore transform matrix to what it was before

SDL

- Easy interface to use
- 3 steps:
  - Step One:
    - #include "sdl.h"
    - Add sdl.cpp to your makefile/workspace
  - Step Two:
    - Instantiate a Scene Object
      - Example: `Scene scn;`
  - Step Three:
    - `scn.read("your scene file.dat");` // reads your scene
    - `scn.makeLightsOpenGL();` // builds lighting data structure
    - `scn.drawSceneOpenGL();` // draws scene using OpenGl

Example: Table without SDL

```c
void hw02::tableLeg(minigl &mgl, double thick, double len){
mgl.mglPushMatrix();
mgl.mglTranslated(0, len/2, 0);
mgl.mglScaled(thick, len, thick);
mgl.mglutSolidCube(1.0);
mgl.mglPopMatrix();
}
```

```c
void hw02::table(minigl &mgl, double topWid, double topThick, double legThick, double legLen){
// draw the table - a top and four legs
mgl.mglPushMatrix();
mgl.mglTranslated(0, legLen, 0);
// note how table uses tableLeg-
mgl.mglPushMatrix();
mgl.mglTranslated(0, legLen, 0);
```

```c
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(-2*dist, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(-2*dist, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, -2*dist);
mgl.mglPushMatrix();
mgl.mglTranslated(0, 0, 2*dist);
```

Example: Table without SDL

```c
void hw02::tableLeg(minigl &mgl, double thick, double len){
mgl.mglPushMatrix();
mgl.mglTranslated(0, len/2, 0);
mgl.mglScaled(thick, len, thick);
mgl.mglutSolidCube(1.0);
mgl.mglPopMatrix();
}
```
Example: Table without SDL

```cpp
// translate and then call
mgl.mglTranslated(0.4, 0, 0.4);
table(mgl, 0.6, 0.02, 0.02, 0.3); // draw the table
```

Example: Table with SDL

```cpp
def leg{push translate 0 .15 0 scale .01 .15 .01 cube pop}

def table{
push translate 0 .3 0 scale .3 .01 .3 cube pop
push translate .275 0 .275 use leg
push translate 0 0 -.55 use leg
push translate -.55 0 .55 use leg
push translate 0 0 -.55 use leg.pop
} }
push translate 0.4 0 0.4 use table pop
```

Examples

- Hill contains useful examples on:
  - Drawing fireframe models (example 5.6.2)
  - Drawing solid models and shading (example 5.6.3)
  - Using SDL in a program (example 5.6.4)
- Homework 3:
  - Will involve studying these examples
  - Work with SDL files in miniGL
  - Start to build your own 3D model

References

- Hill, 5.6, appendices 3,5
- Angel, Interactive Computer Graphics using OpenGL