Modeler Help Session

Help Session Overview

- Checking out, building, and using the sample solution
- Part 1: Surface of Revolution
- Part 2: Hierarchical Modeling
- Part 3: Blinn-Phong Shader
- Part 4: Custom Shader(s)

Checking Out Your Code

- Use Gitlab, just like for Impressionist
- Detailed instructions are in the project description

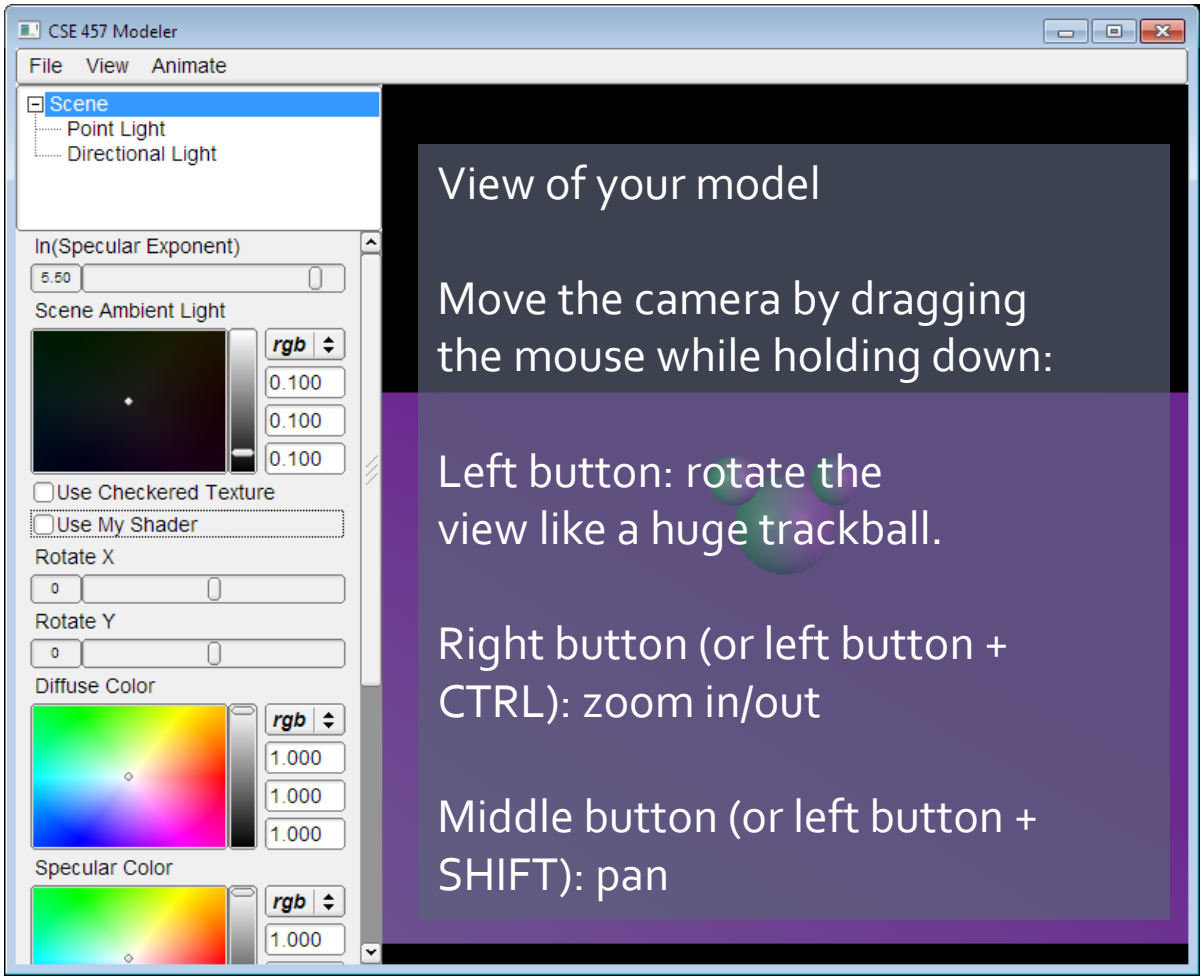
Building in Visual Studio

- Go to your project folder
- Double-click the .vcxproj file
- Configuration menu next to green arrow
  - Debug – lets you set breakpoints
  - Release – for turn-in
- Pick **Debug**, then click the green arrow next to it to build and run your project (Hotkey: F5)
- Let us know if the skeleton doesn't build!

Introducing Modeler

Control Groups

List of Controls

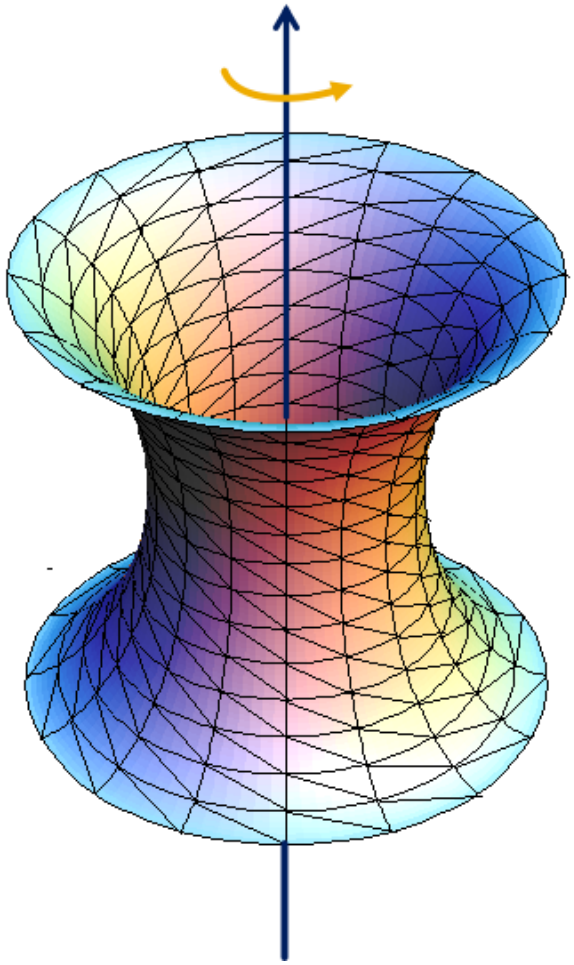


Dividing Up The Work

- Partner A: Modeling
  - Part 1: Surface of revolution
  - Part 2: Hierarchical Modeling
- Partner B: Shading
  - Part 3: Blinn-Phong Shader
  - Part 4: Custom Shader(s)
- NOTE: this division of labor is just a suggestion!

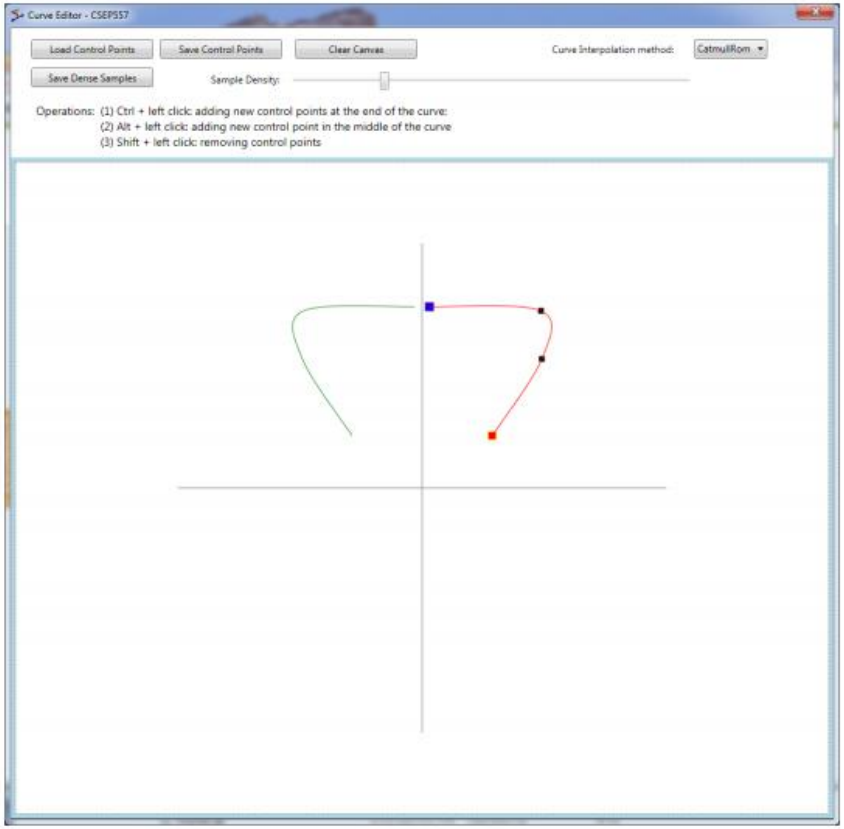
Part 1: Surface of Revolution

- You will write OpenGL code to draw a surface by rotating a curve.
- Each vertex must have an appropriate:
  - Texture coordinate pair
  - Vertex normal
  - Position
- Replace code for drawRevolution() in modelerdraw.cpp
  - The **divisions** variable determines number of slices
  - Current skeleton implementation is a naïve and incorrect one, but illustrates how curve files can be changed into geometry
- Load new curve with File->"Load Revolution Curve File"



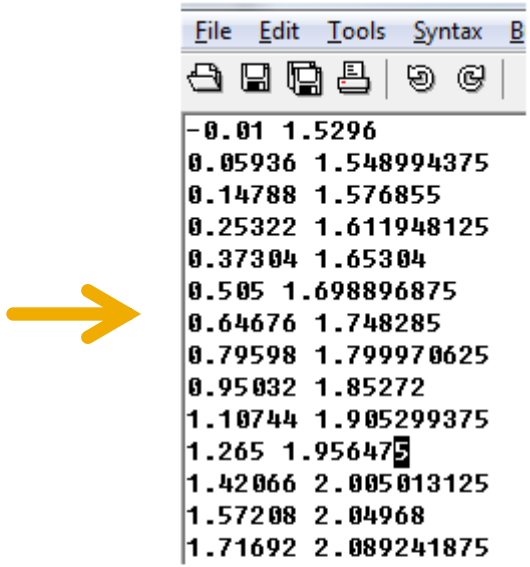
How to start

- Drawing a curve
  - Using the curve editor tool
  - Start by left click with ctrl key on
  - Save dense point samples into .apts file
  - Load point samples in modeler

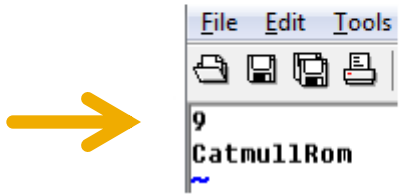


Curve file format

- A curve file is basically a .txt file with a list of x,y coordinates for control points
- .apts
  - Densely sampled points on a curve

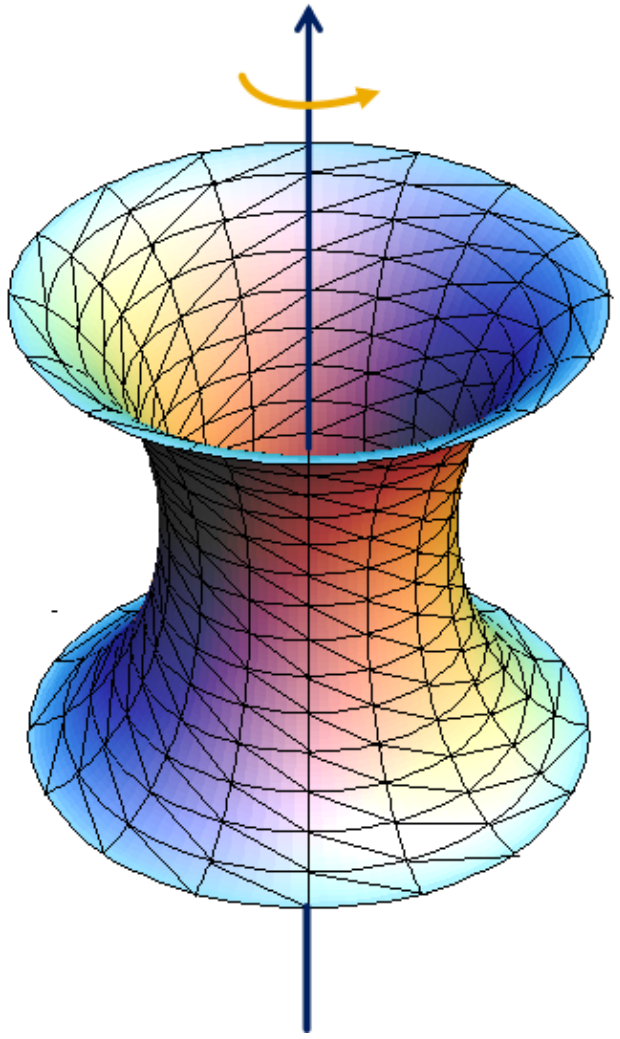


- .cfg: curve configuration file
  - Row 1: sample density
  - Row 2: curve interpolation method



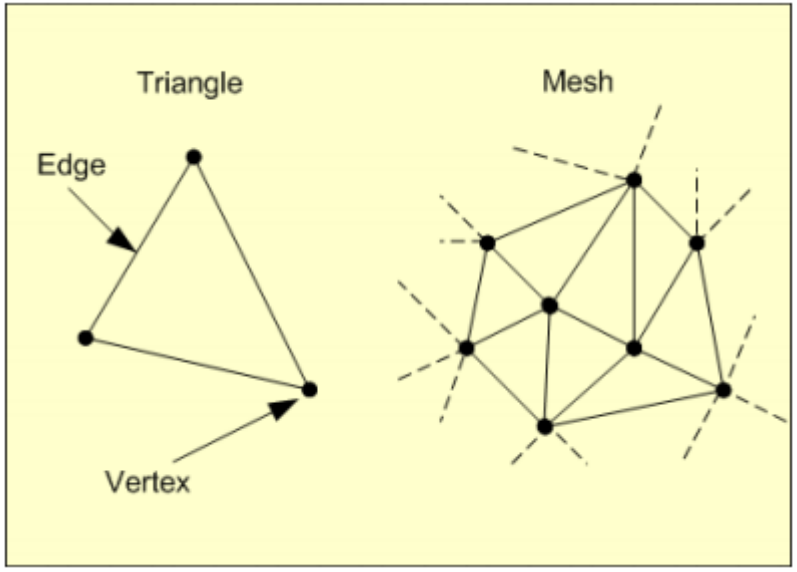
Slicing it into Triangle Strips

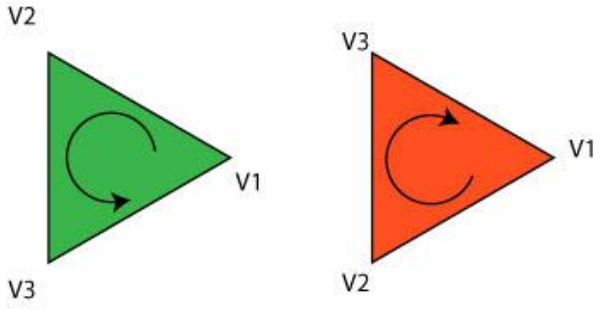
- Divide the surface into "bands" by longitude
- Compute vertex positions and normals
  - Using sin(), cos() in C++ code
  - See lecture notes for normal computation
- Connect the dots with OpenGL triangles



Connecting dots in a modern way

- Use glDrawElements with GL_TRIANGLES (required!)
- The order of vertices matters
  - Right-hand rule





The triangle on the left has a CCW winding order so it will be visible on the screen. The triangle on the right has a CW winding order so you will not see it render on the screen.

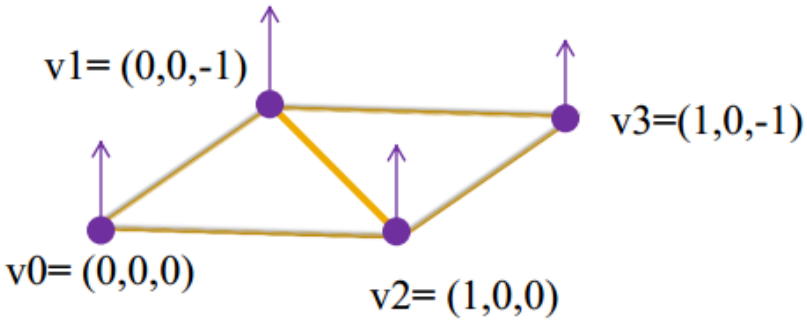
Connecting dots

- It's okay to use glBegin(), glEnd() for testing shapes, but don't use them in the final submitted code
- Don't use GL_QUAD_STRIP or GL_TRIANGLE_STRIP in the final submission, either.

- In the submitted code, you need to build a triangle mesh and send it to OpenGL
  - Using glDrawElements with GL_TRIANGLES

An Example

This is an overly simplified example of drawing a plane using glDrawElements. The plane consists of two connecting triangles and the normal vectors of all vertices are pointing up.

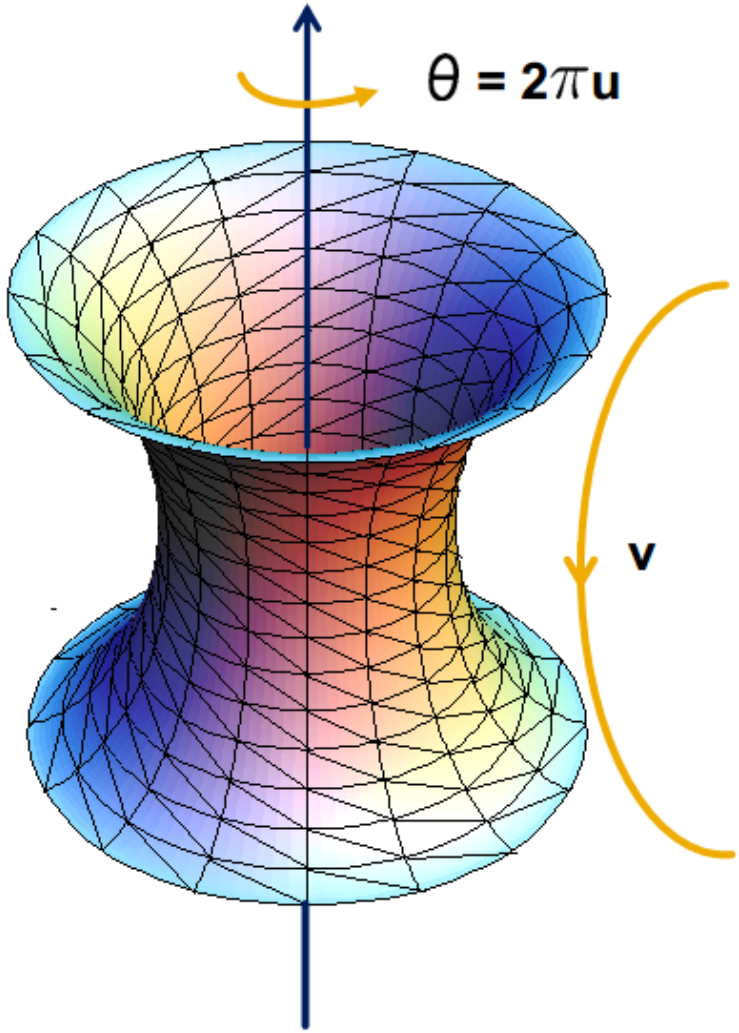



```
// preparing the data for the vertices' positions
GLfloat vertices[12] = { 0,0,0, 0,0,-1, 1,0,0, 1,0,-1 };
// normal directions
GLfloat normals[12] = {0,1,0, 0,1,0, 0,1,0, 0,1,0};
// texture coordinates
GLfloat texture_uv[8] = {0,0, 0,1, 1,0, 1,1};
// vertex indices in order to form triangles
// (order of the vertices follows the right hand rule)
const int indices_length = 6;
GLuint indices[indices_length] = { 1,0,2, 1,2,3 };
```

```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);
glNormalPointer(GL_FLOAT,0,normals);
glTexCoordPointer(2,GL_FLOAT,0,texture_uv);
glDrawElements(GL_TRIANGLES, indices_length
,GL_UNSIGNED_INT, indices);
glDisableClientState(GL_TEXTURE_COORD_ARRAY);
glDisableClientState(GL_NORMAL_ARRAY);
glDisableClientState(GL_VERTEX_ARRAY);
```

Texture Mapping

- See lecture slides for texture mapping
  - Basic idea: use longitude and arc length (curve distance) as texture coordinates
- Each vertex must have an appropriate:
  - Vertex normal
  - Position
  - Texture Coordinate Pair
    - u,v ∈ [0,1]



Part 2: Hierarchical Modeling

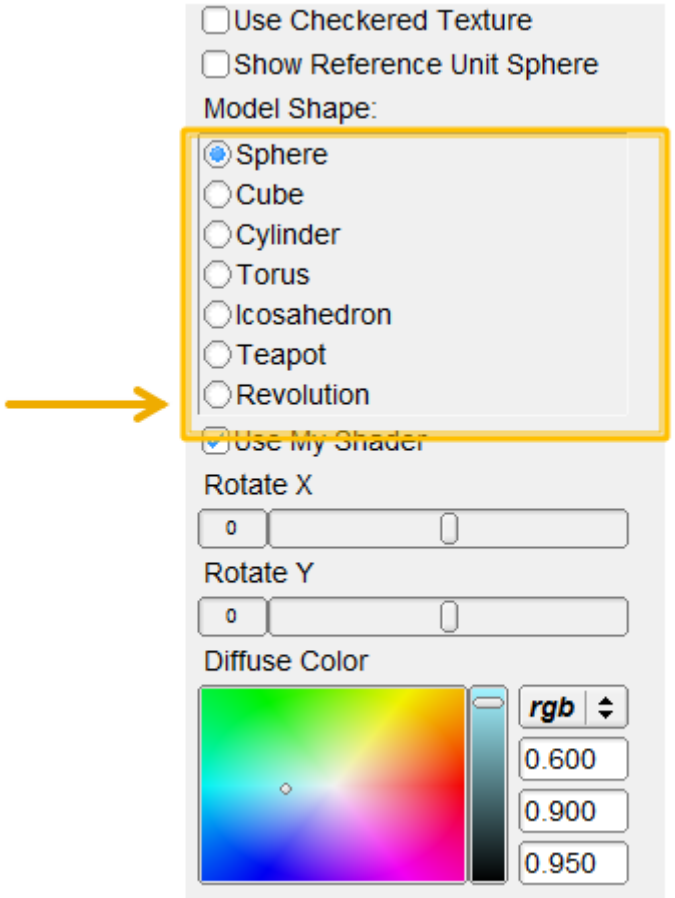
- You must make a **character** with:
  - 2 levels of branching
  - Something drawn at each level
  - Meaningful controls
    - Otherwise, you will be overwhelmed when you animate it!
- You will need to:
  - Extend the Model class
  - Override the draw() method
  - Add properties that Modeler users can control
  - Give an instance of your class to ModelerUserInterface in the main() function

Building a scene of your own

- In sample.cpp, the Scene class extends Model
  - draw() method draws the green floor, sphere, cylinder, etc.
  - Add and replace with drawing commands of your own
- You can use these draw commands as OpenGL references
  - Modelerdraw.cpp
    - drawBox
    - drawCylinder
    - drawRevolution

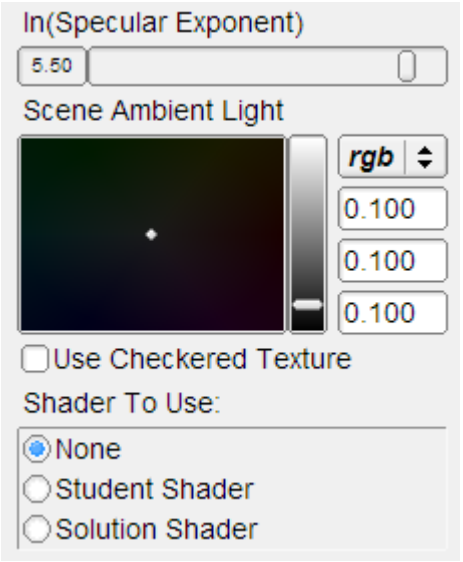
Add a radio button for your scene

- Add a new radio button for your scene at the end of the list



Add Properties to Control It

- Kinds of properties (in properties.h):
  - BooleanProperty = checkbox
  - RangeProperty = slider
  - RGBProperty = color
  - ChoiceProperty = radio buttons
- Need to add it to:
  1. Class definition
  2. Constructor
  3. Property list
- See sample.cpp for example



OpenGL Is A State Machine

- glEnable()/glDisable() changes state
- Once you change something, it stays that way until you change it to something new
- OpenGL's state includes:
  - Current color
  - Transformation matrices
  - Drawing modes
  - Light sources

OpenGL's Transformation Matrix

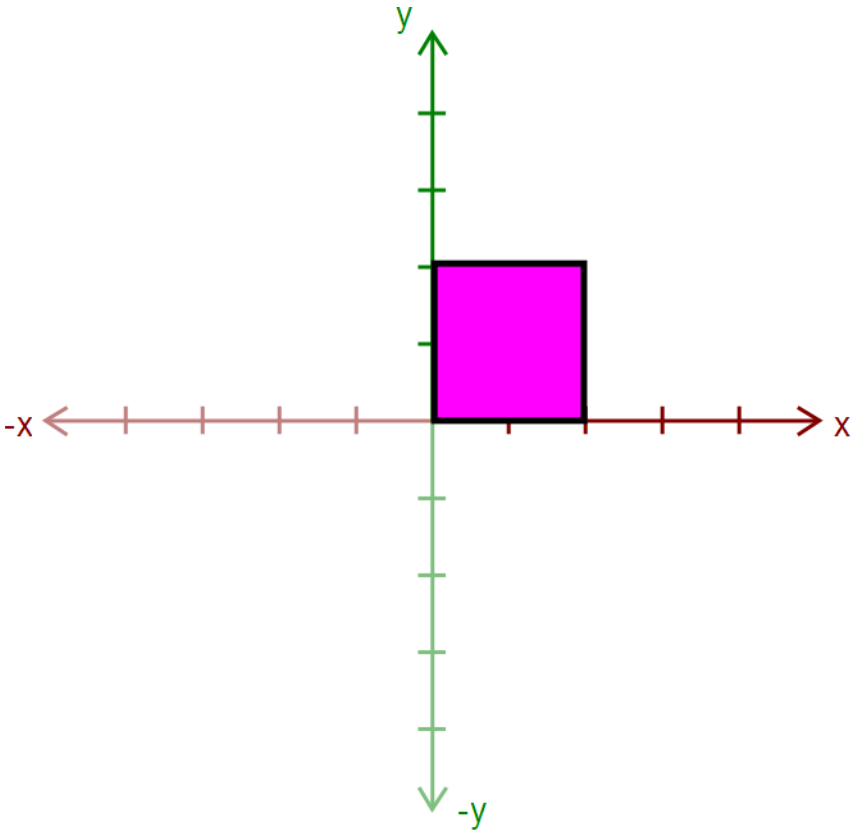
- Just two of them: projection and modelview. We'll modify modelview.
- Matrix applied to all vertices and normals
- These functions multiply transformations: glRotated(), glTranslated(), glScaled()
- Applies transformations in REVERSE order from the order in which they are called.
- Transformations are cumulative. Since they're all "squashed" into one matrix, you can't "undo" a transformation.

Transformations: Going "Back"

- How do we get back to an earlier transformation matrix?
- We can "remember" it
  - OpenGL maintains a stack of matrices.
  - To store the current matrix, call glPushMatrix().
  - To restore the last matrix you stored, call glPopMatrix().

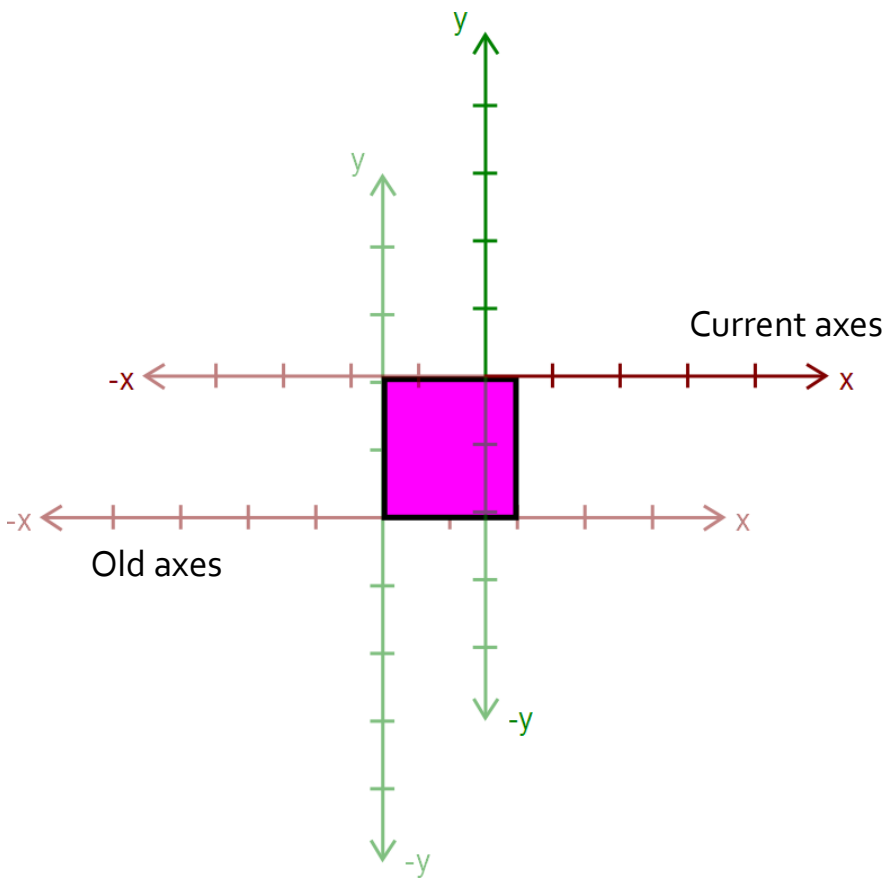
Hierarchical Modeling in OpenGL

- Draw the body
- Use glPushMatrix() to remember the current matrix.
- Imagine that a matrix corresponds to a set of coordinate axes:
  - By changing your matrix, you can move, rotate, and scale the axes OpenGL uses.



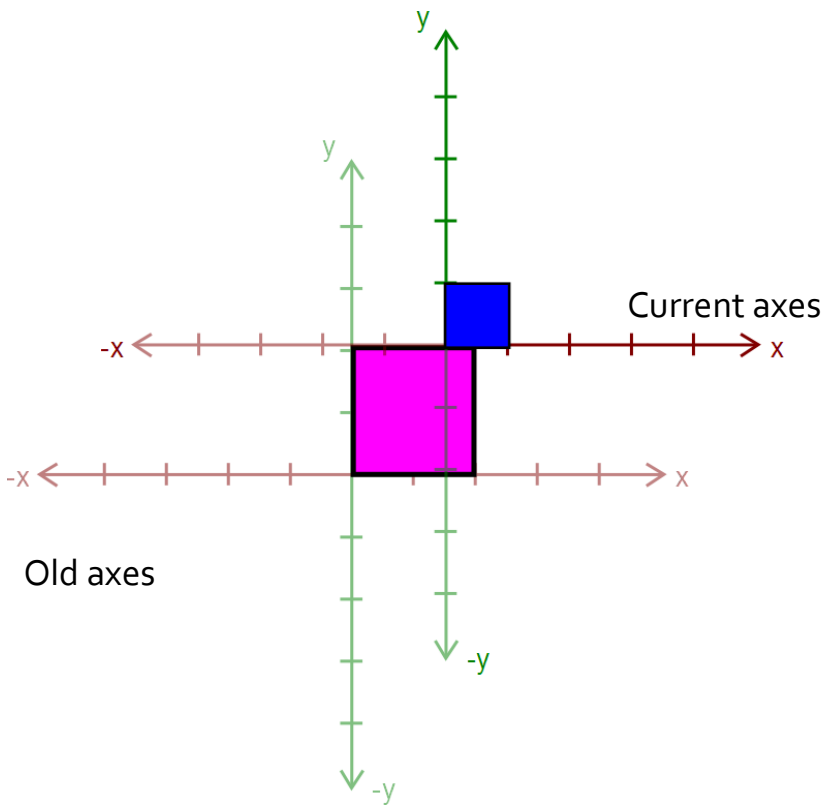
Hierarchical Modeling in OpenGL

- Apply a transform:
  - glRotated()
  - glTranslated()
  - glScaled()
- Here, we apply glTranslated(1.5,2,0)
  - All points translated 1.5 units left and 2 units up
  - It's as if we moved our coordinate axes!



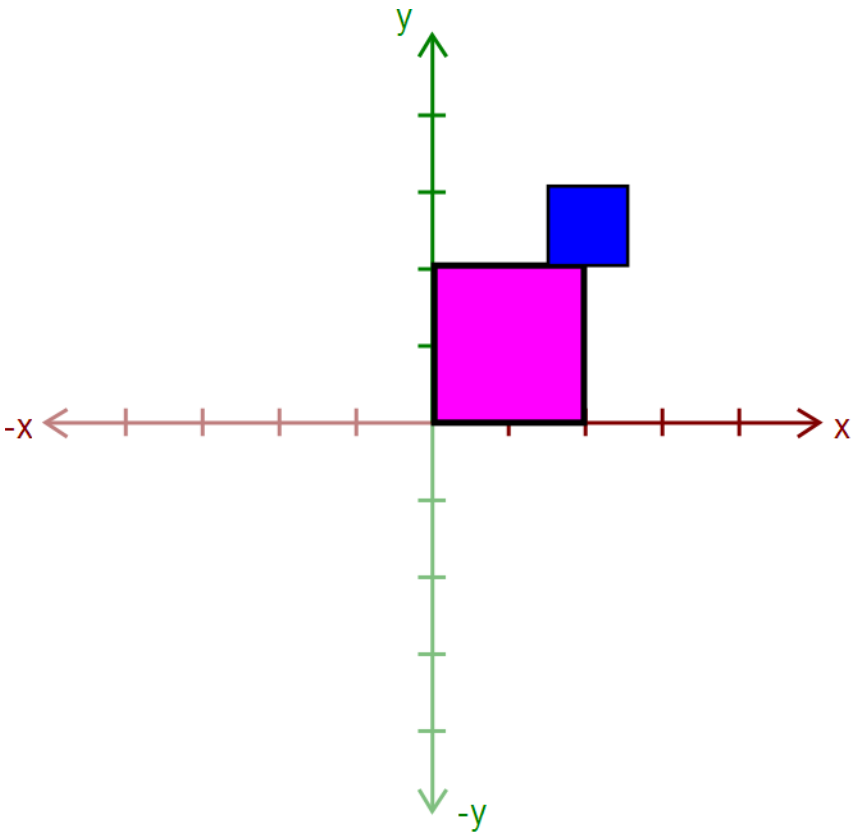
Hierarchical Modeling in OpenGL

- Draw an ear.
  - This ear thinks it was drawn at the origin.
- Transformations let us transform objects without changing their geometry!
  - We didn't have to edit that ear's drawing commands to transform it



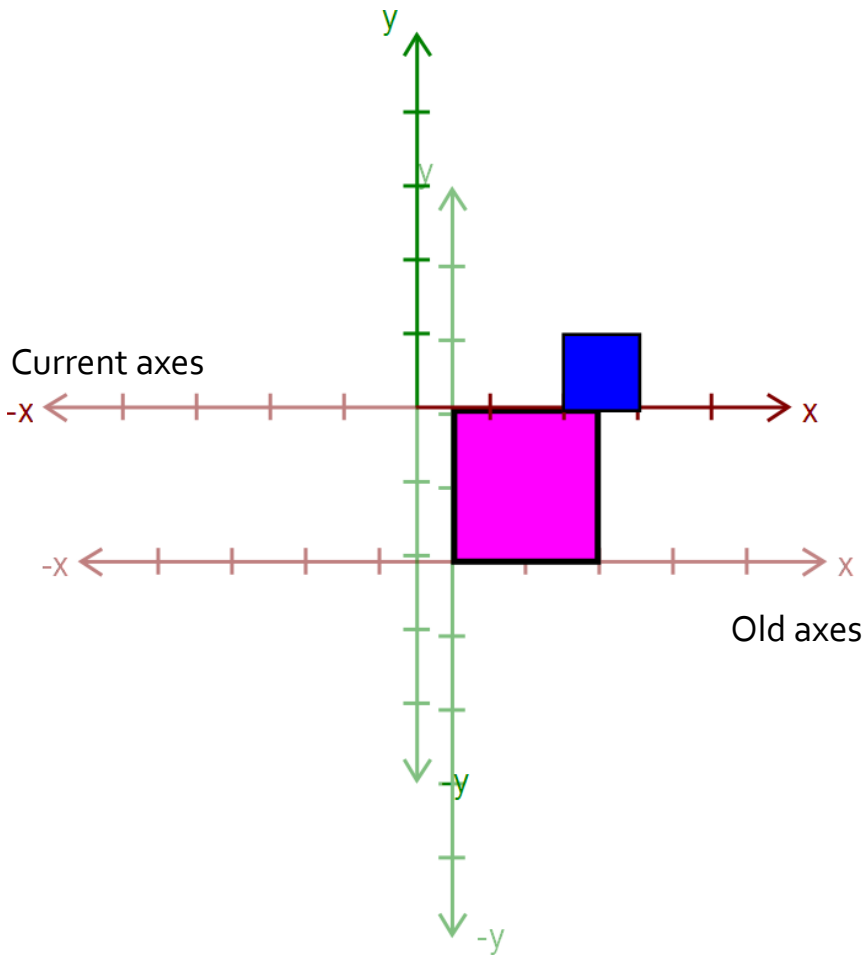
Hierarchical Modeling in OpenGL

- Call glPopMatrix() to return to the body's coordinate axes.
- To draw the other ear, call glPushMatrix() again…



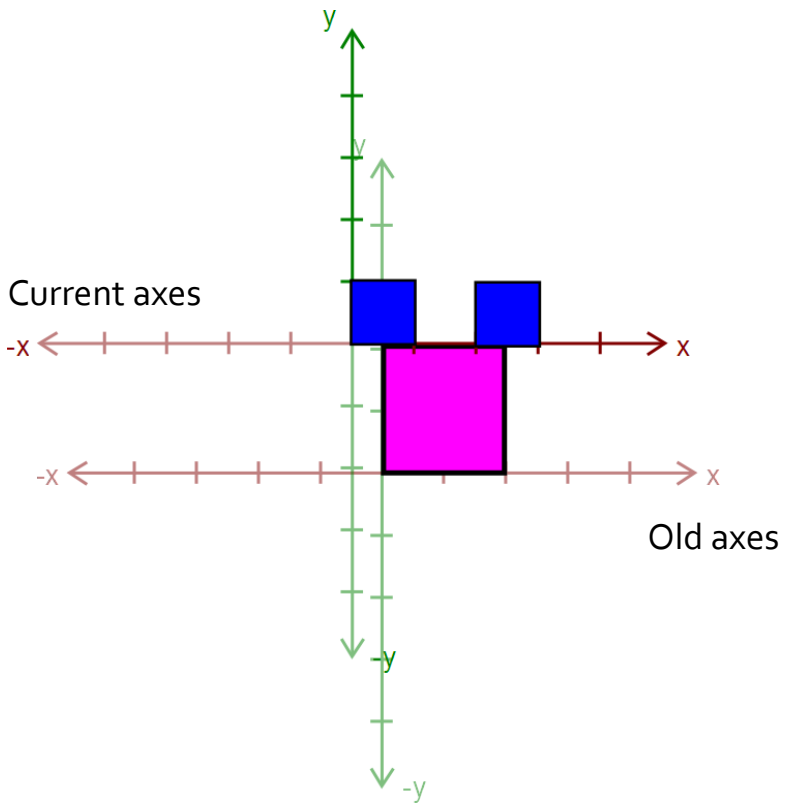
Hierarchical Modeling in OpenGL

- Apply another transform…
  - Where will the ear be drawn now?



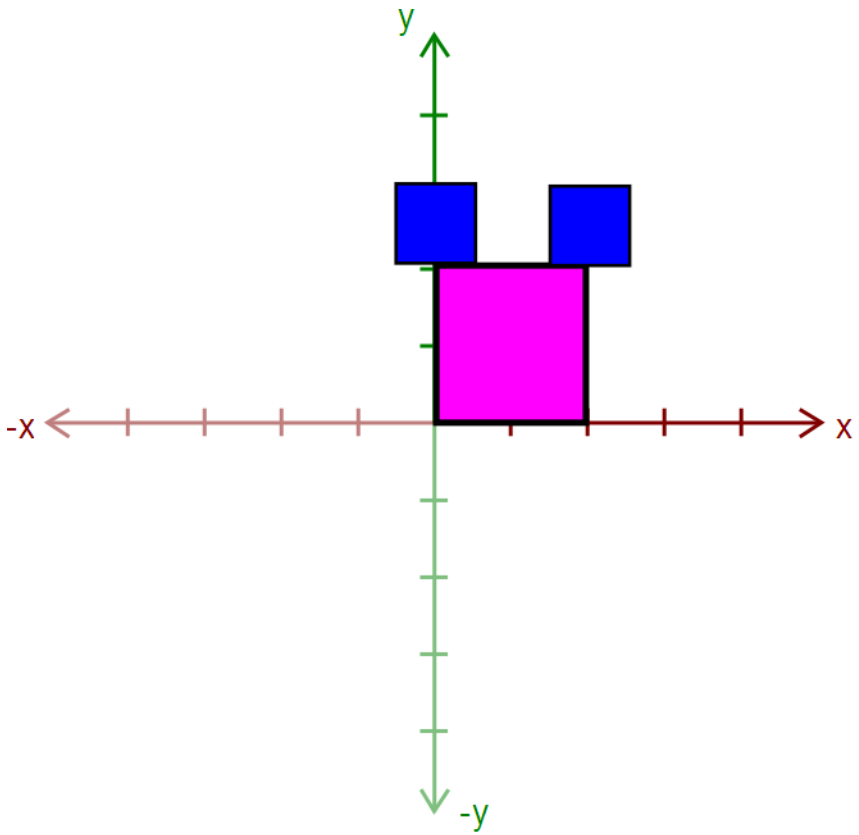
Hierarchical Modeling in OpenGL

- Draw the other ear



Hierarchical Modeling in OpenGL

- Then, call glPopMatrix() to return to the body's "axes"
  - Technically, you don't need to if that second ear is the last thing you draw.
  - But what if you wanted to add something else to the body?

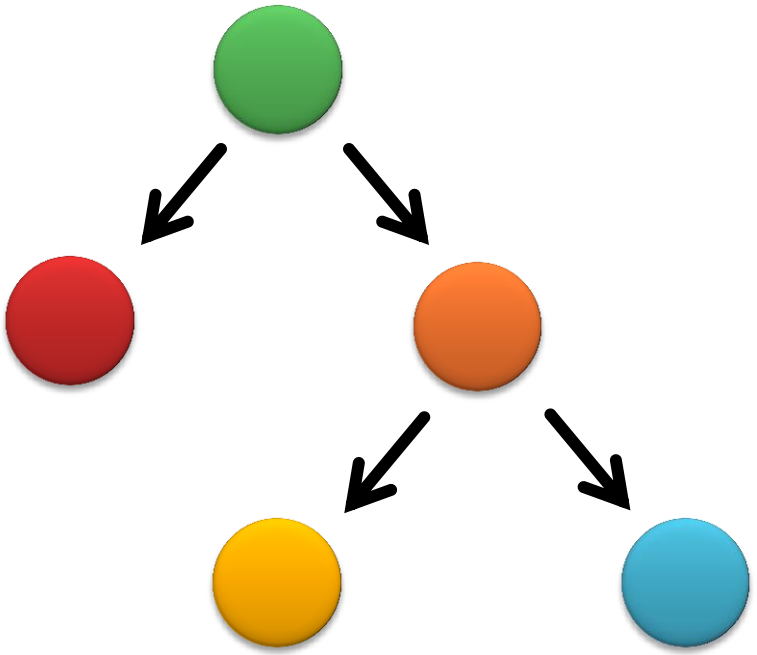


Rule: A Pop For Every Push

- Make sure there's a glPopMatrix() for every glPushMatrix()!
  - You can divide your draw() function into a series of nested methods, each with a push at the beginning and a pop at the end.

Levels of Branching

- Your scene must have two levels of branching like in this diagram.
  - Circles are objects
  - Arrows are transformations
- Call glPushMatrix() after drawing green, so you can draw orange after drawing red
  - Do the same for orange
- You must draw something at each level.



Multiple-Joint Slider

- Needs to control multiple aspects of your model.
  - Example: Rotate multiple joints at once
- Don't get too complicated!
  - Wait for Animator in four weeks!

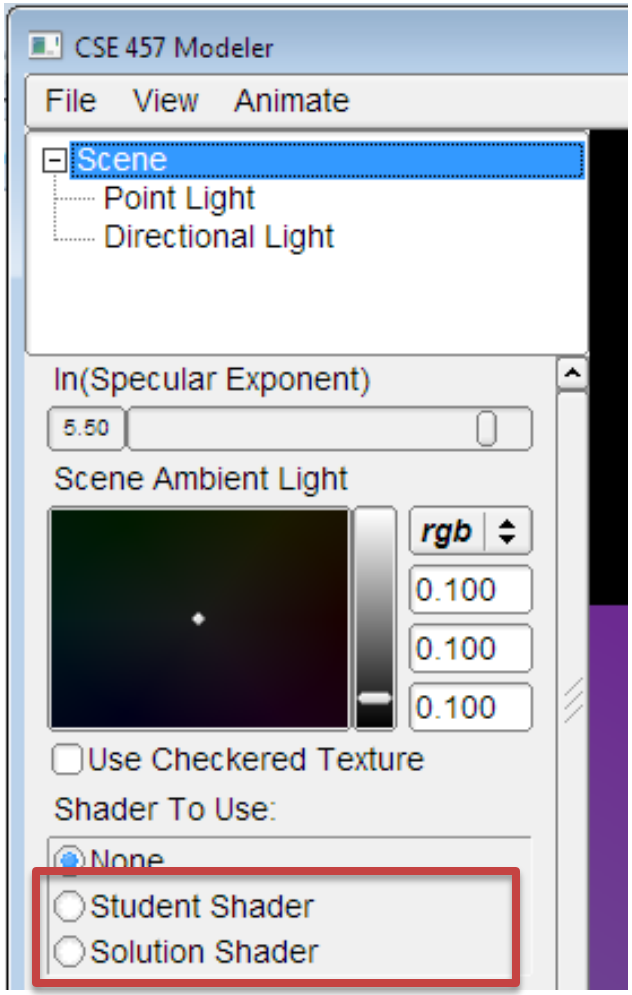
Part 3. Blinn-Phong Shader

- We provide a directional light shader in OpenGL Shading Language (GLSL)
- You must extend it to support point lights.

- Files to edit:
  - shader.frag – your fragment shader
  - shader.vert – your vertex shader

Compare with the Sample Solution

- modeler_solution.exe in your project folder
  - Loads your shader.frag and shader.vert.
  - Also contains our sample shaders.
- Use radio buttons to compare with sample solution



Choose shader here

Useful GLSL Variables

- gl_LightSource[i].position.xyz – the position of light source i.
- gl_FrontLightProduct[i] – object that stores the product of a light's properties with the current surface's material properties:
  - Example: gl_FrontLightProduct[i].diffuse == gl_FrontMaterial.diffuse * gl_LightSource[i].diffuse

Part 4. Your Custom Shader

- Anything you want!
- Can earn extra credit!
- Ask TA's for estimated extra credit value of an option.
- See the **OpenGL orange book** in the lab for details + code.
- Can still use sample solution to test (depending on complexity)

- Warnings
  - Don't modify any files except your model file and the required modifications
  - Or, your model might not work in Animator (project 4)

Preparing Your Work Environment

- Make sure that your repository works by:
  - Cloning or pulling the latest updates
  - Building it
  - Tweaking something
  - Committing
- Do this on each work environment you plan to use, even if you aren't going to start work yet:
  - Lab machines
  - Your home computer
  - The sooner we know of a problem, the sooner we can fix it.
  - Remember that we'll grade using the lab machines!

Avoiding Git Conflicts

- In general, never put anything besides source code into source control:
  - Debug and Release folders
  - Modeler.suo
  - Modeler.ncb
  - *.user files
- DO put source files (*.cpp, *.h, *.vcproj, image files, etc.) in the repository
  - Make sure you both **add** AND **commit** the files.

Quick Summary

THINGS TO DO

- Partner A: Modeling
  - Part 1: Surface of revolution
  - Part 2: Hierarchical Modeling
- Partner B: Shading
  - Part 3: Blinn-Phong Shader
  - Part 4: Custom Shader(s)
- You don't *have* to divide work up this way!

WARNINGS

- Don't modify any files except your model file and the required modifications
  - Or, your model might not work in Animator
- Make sure you can pull, commit, and build!

Before You Leave

- Try adjusting the sample model
  - Let us know if you have problems
- COMMIT BEFORE LOGOFF!
  - Your files in C:\User\… will go away when you log out, due to Deep Freeze!