Modeler Help Session

Due: Thursday, April 28th *by the stroke of midnight!*
TA: Jeff Booth

Help Session Overview

- Checking out, building, and using the sample solution
- Part 1: Rendering a Sphere
- Part 2: Hierarchical Modeling
- Part 3: gluLookAt()
- Part 4: Blinn-Phong Shader
- Part 5: Custom Shader

Checking Out Your Code

- Go to the Modeler course page for detailed check-out directions.
- Repository path:
  - svn+ssh://**Your CSE NetID**@attu.cs.washington.edu/projects/instr/11sp/cse457/modeler/**Your Group ID**/source

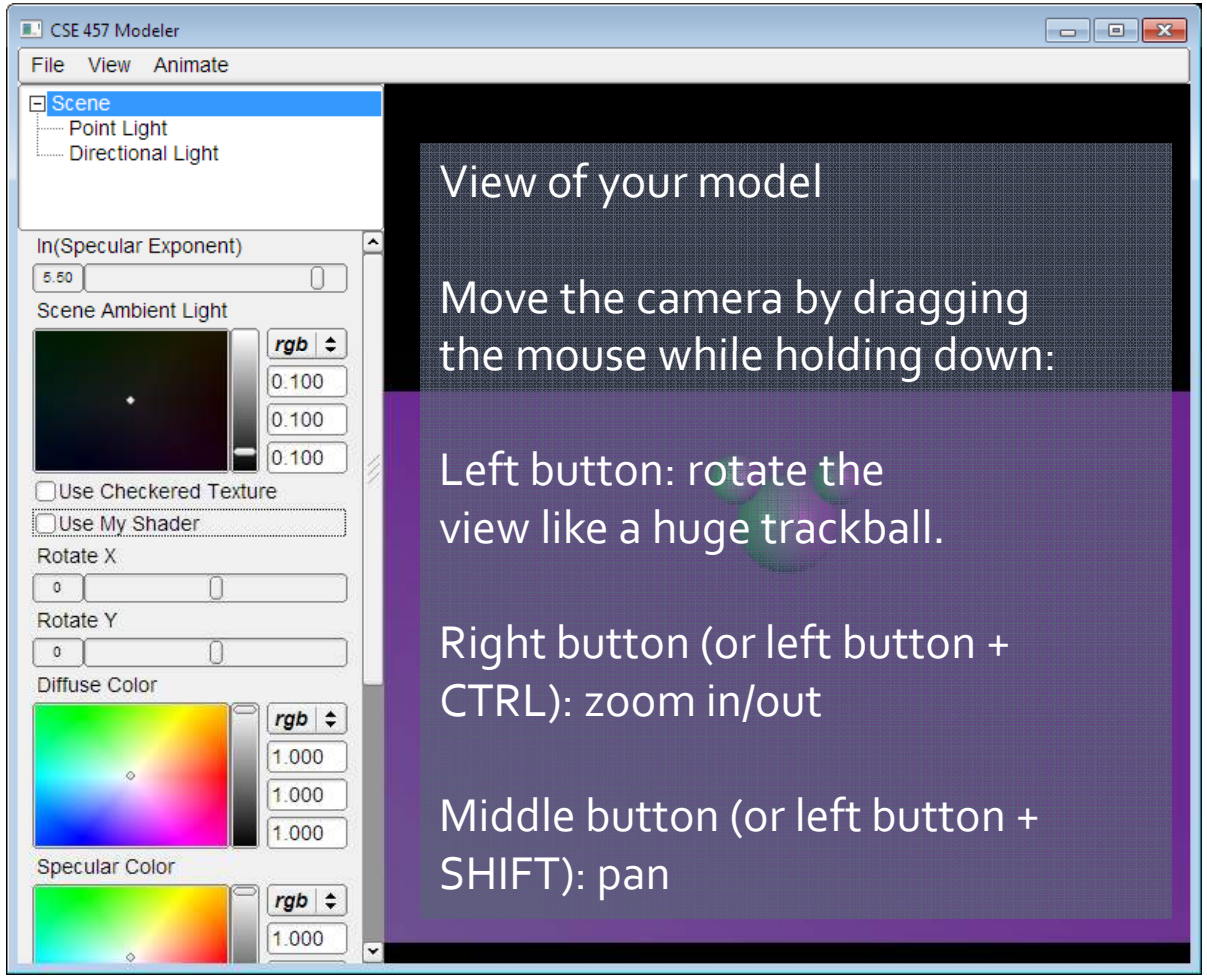
Building in Visual Studio

- 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
- Let us know if it doesn't build!

Introducing Modeler

Control Groups

List of Controls

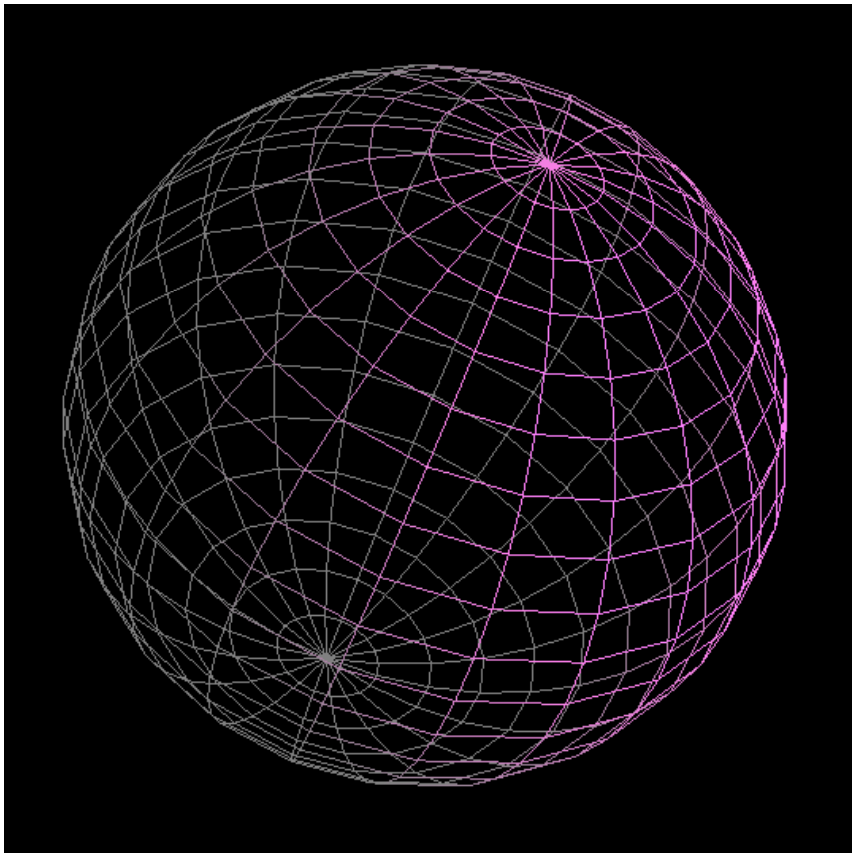


Dividing Up The Work

- Partner A: Modeling
  - Part 1: Hierarchical Modeling
  - Part 2: Custom Primitive
- Either Partner:
  - Part 3: gluLookAt()
- Partner B: Shading
  - Part 4: Blinn-Phong Shader
  - Part 5: Custom Shader
- NOTE: this division of labor is just a suggestion!

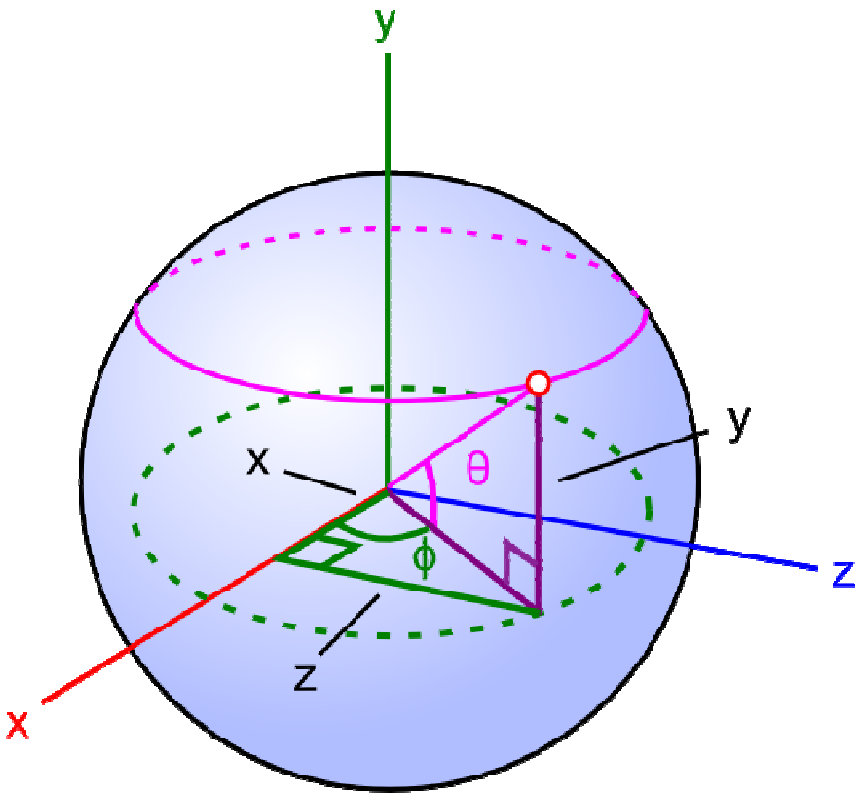
Part 1: Rendering a Sphere

- You will write OpenGL code to draw a sphere.
- Each vertex must have an appropriate:
  - Texture coordinate pair
  - Vertex normal
  - Position
- Replace code for drawSphere() in modelerdraw.cpp
  - The **divisions** variable determines number of slices



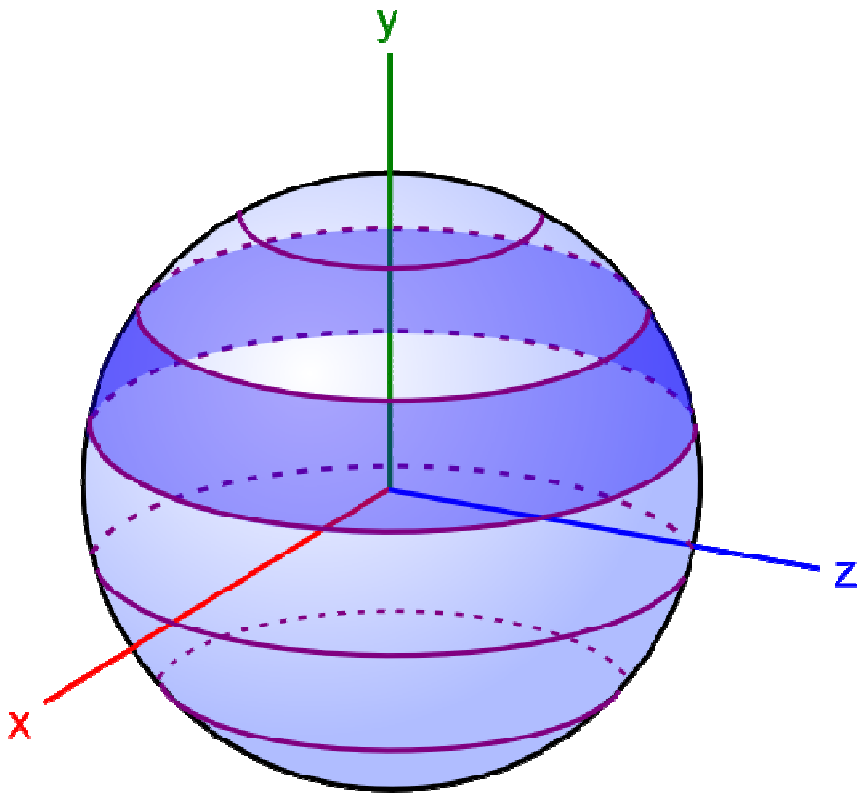
Parameterizing a Sphere

- Determine (x,y,z) coordinates of each point using sphere radius, latitude θ and longitude ϕ
- For trig:
  - Give **degrees** to all GL functions
  - Give **radians** to C++ math functions (sin(), cos(), etc.)



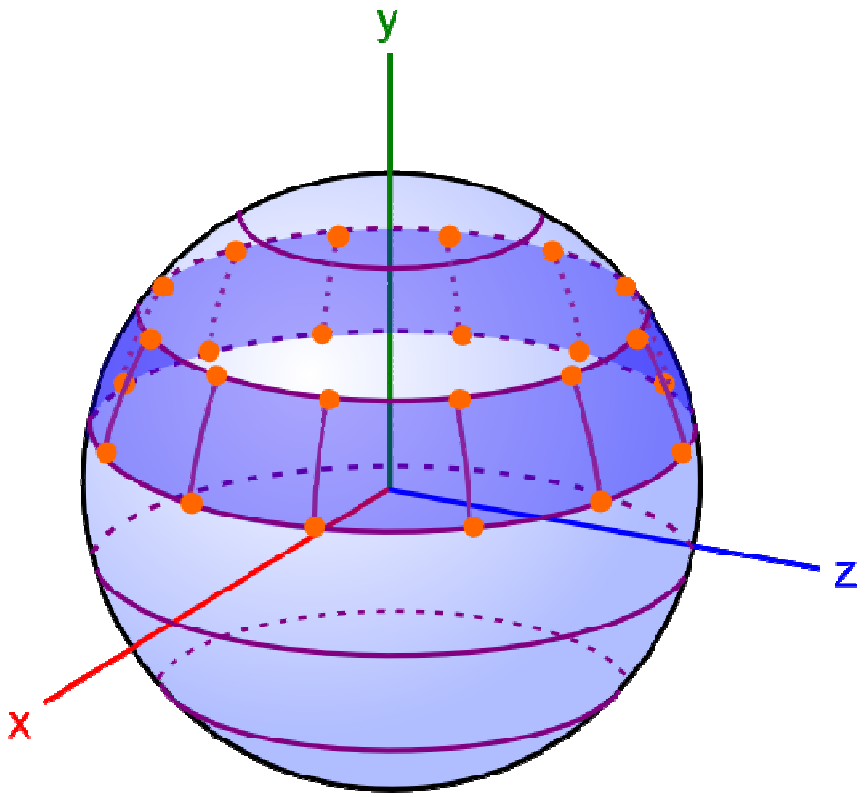
Slicing It Into Polygon Strips

- Divide sphere into "rings" (purple lines) by latitude
- # of rings = **divisions** variable
- Fill in the area between each ring (dark blue region) with a strip of polygons



Drawing Each Polygon Strip

- Divide slices into quadrilaterals by longitude
- # of slices = **divisions** variable!
- Connect the dots with OpenGL quadrilaterals or triangles.



Drawing with OpenGL

```
glBegin(DRAW_TYPE);
    …
    glNormal3f(0, 1, 0);
    glTexCoord2f(0,0);
    glVertex3f(1, 2, 3);
    …
glEnd();
```

- Tell OpenGL what primitive you're drawing with glBegin()
  - GL_TRIANGLES
  - GL_TRIANGLE_STRIP
  - GL_TRIANGLE_FAN
  - GL_QUADS
  - GL_QUAD_STRIP

Using Strip Primitives

- Use strip primitives like GL_QUAD_STRIP for connected polygons
- If you send 12 points to graphics card:
  - GL_QUADS draws 3 quads
  - GL_QUAD_STRIP draws 5 quads by reusing points for more than one quad.
  - Order matters – see diagram!

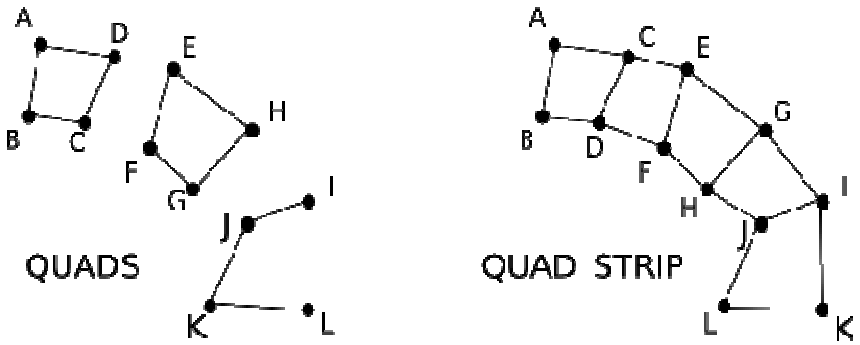
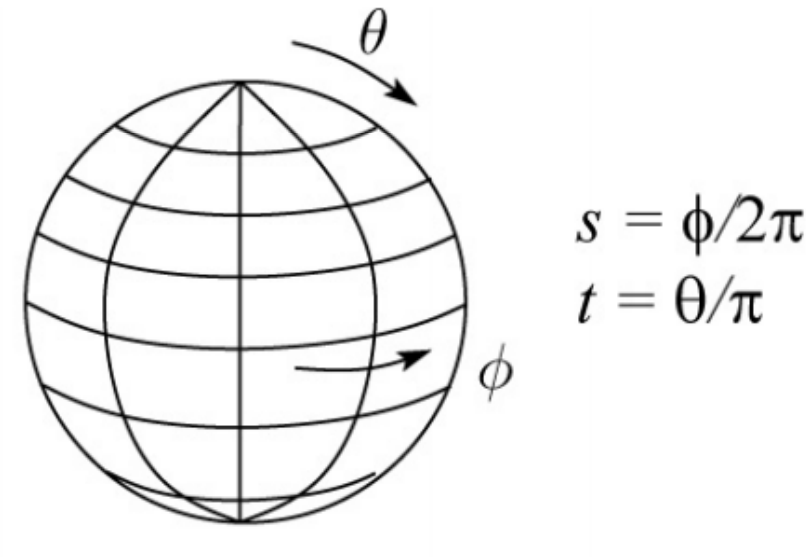


Diagram comparing quads drawn by GL_QUADS and GL_QUAD_STRIP, given the same points (from http://math.hws.edu/graphicsnotes/c3/s2.html)

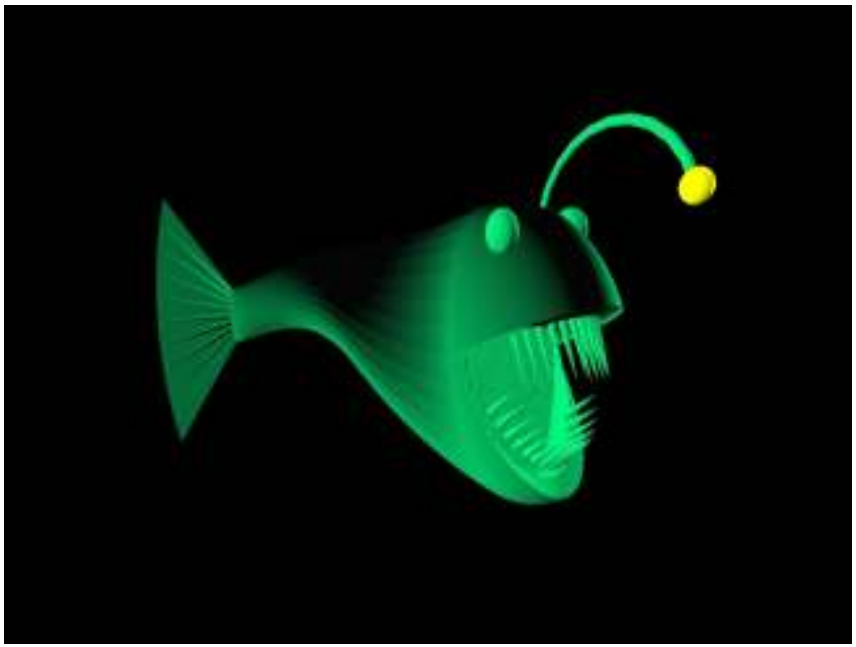
Spherical Texture Mapping

- See lecture slides for spherical texture mapping
  - Basic idea: use latitude and longitude as texture coordinates



Extra Credit: Cool Surfaces

- Surfaces of Rotation
- Smooth Surfaces
- Swept Surfaces
- Rail Surfaces
- Non-Linear Transformations
- Heightfields
- **Most are easy once you implement the sphere!**



Smooth fishy surface *(Michael Kidd and Igor Tolkov, Spring 2010)*

Part 2: Hierarchical Modeling

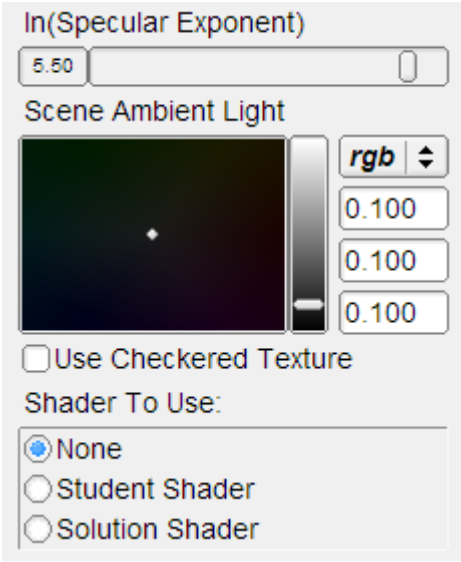
- You must make a **character** with:
  - 2 levels of branching
  - Something drawn at each level
  - Meaningful controls
    - Otherwise, you will be overwhelmed when you animate it!
- You will need to:
  - Extend the Model class
  - Override the draw() method
  - Add properties that Modeler users can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

Building a Scene

- In sample.cpp, the Scene class extends Model
  - draw() method draws the green floor, sphere, and cylinder
  - Add and replace drawing commands of your own
- Where are the drawing commands?
  - Modelerdraw.cpp
    - drawBox
    - drawCylinder
    - drawSphere

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

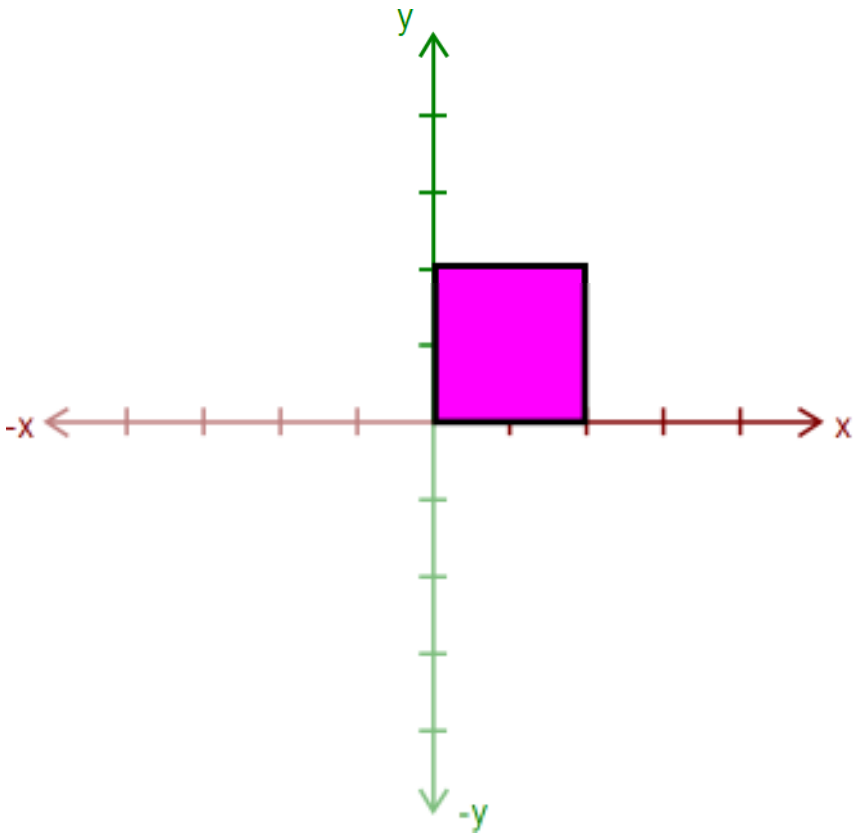
- Just two of them: projection and modelview. We'll modify modelview.
- Matrix applied to all vertices and normals
- These functions multiply transformations: glRotated(), glTranslated(), glScaled()
- Applies transformations in REVERSE order from the order in which they are called.
- Transformations 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().

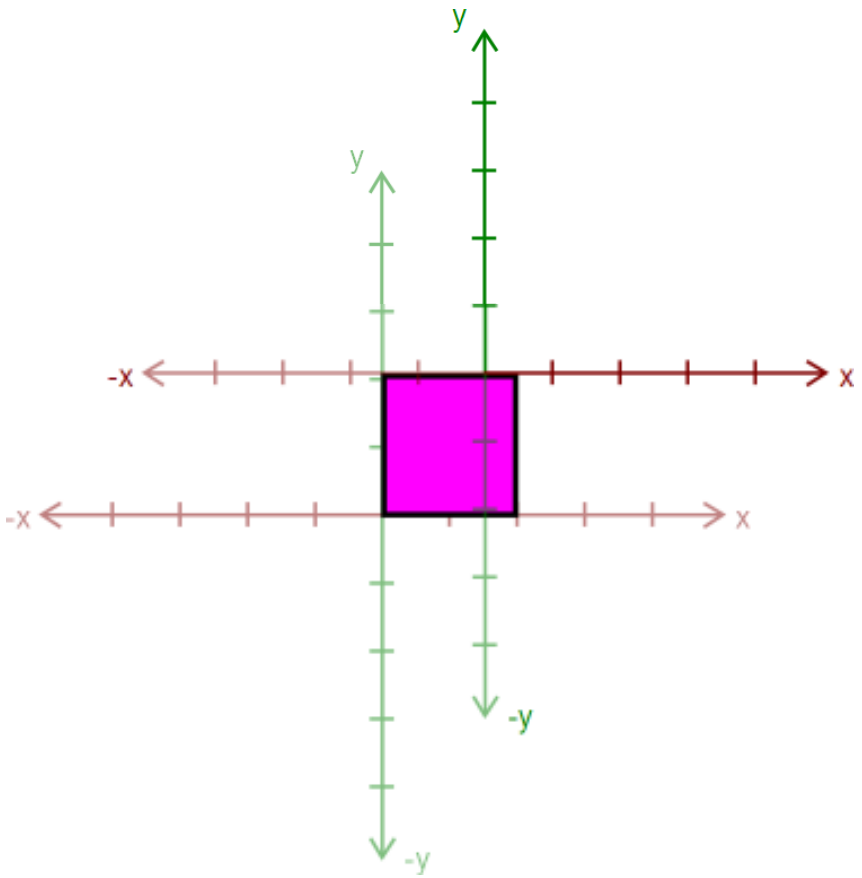
Hierarchical Modeling in OpenGL

- 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.



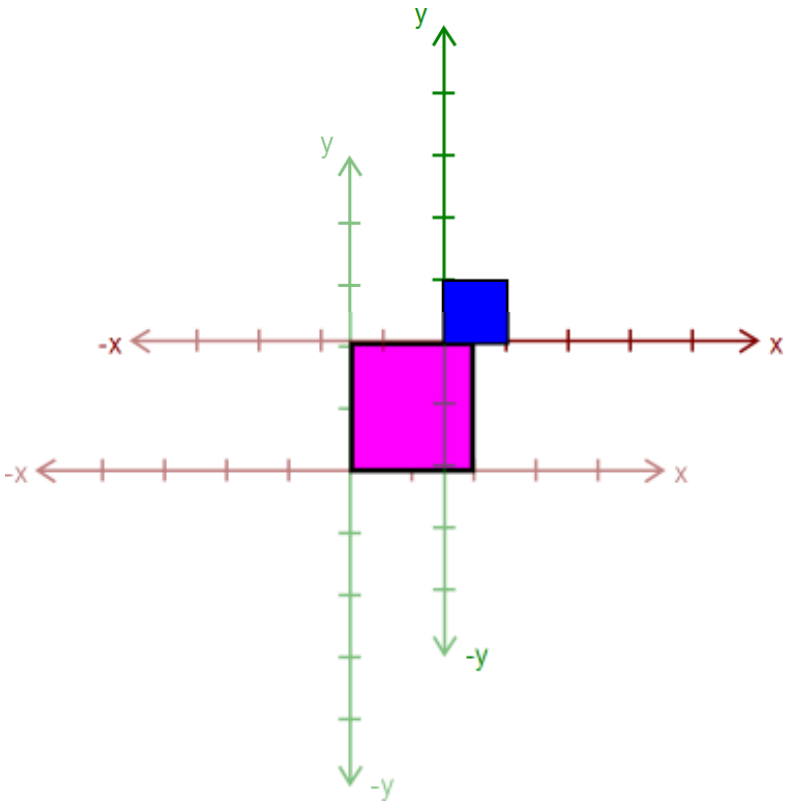
Hierarchical Modeling in OpenGL

- 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!



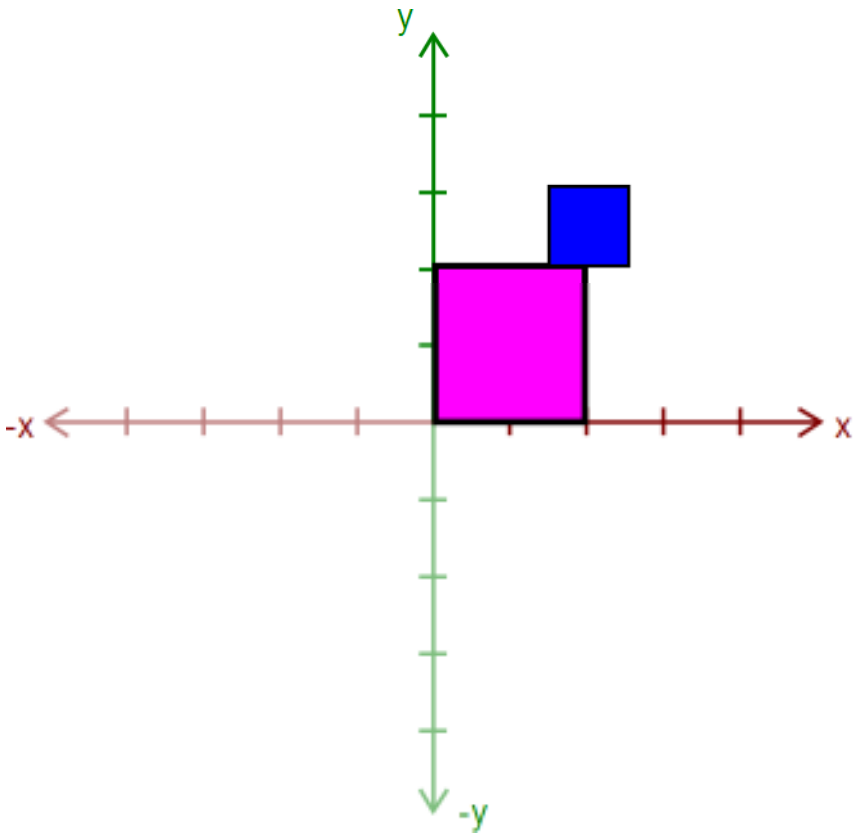
Hierarchical Modeling in OpenGL

- 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



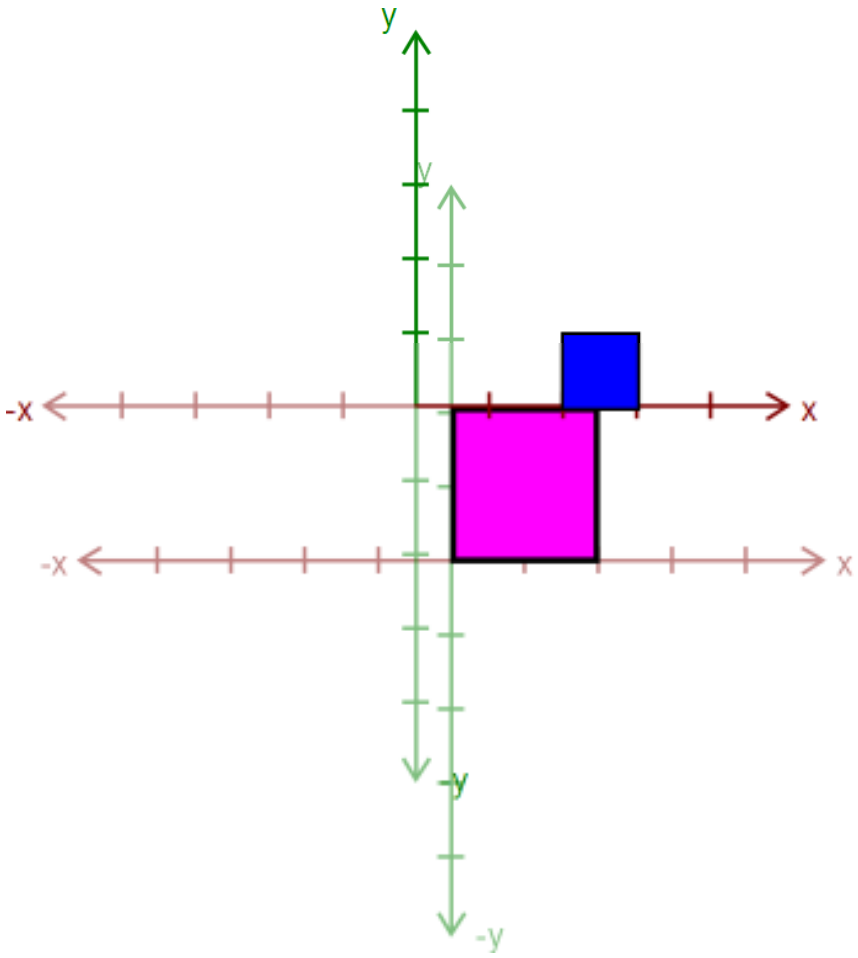
Hierarchical Modeling in OpenGL

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



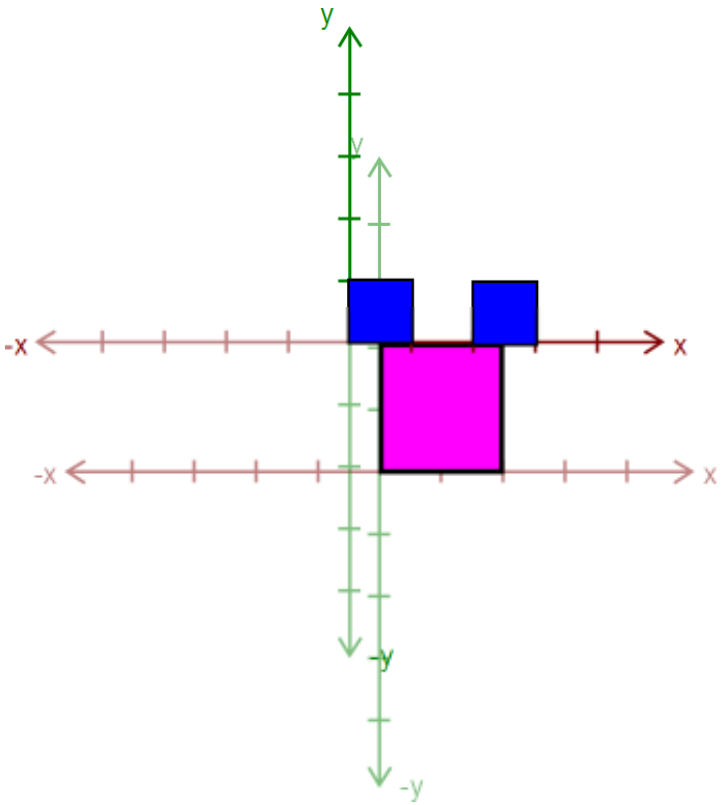
Hierarchical Modeling in OpenGL

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



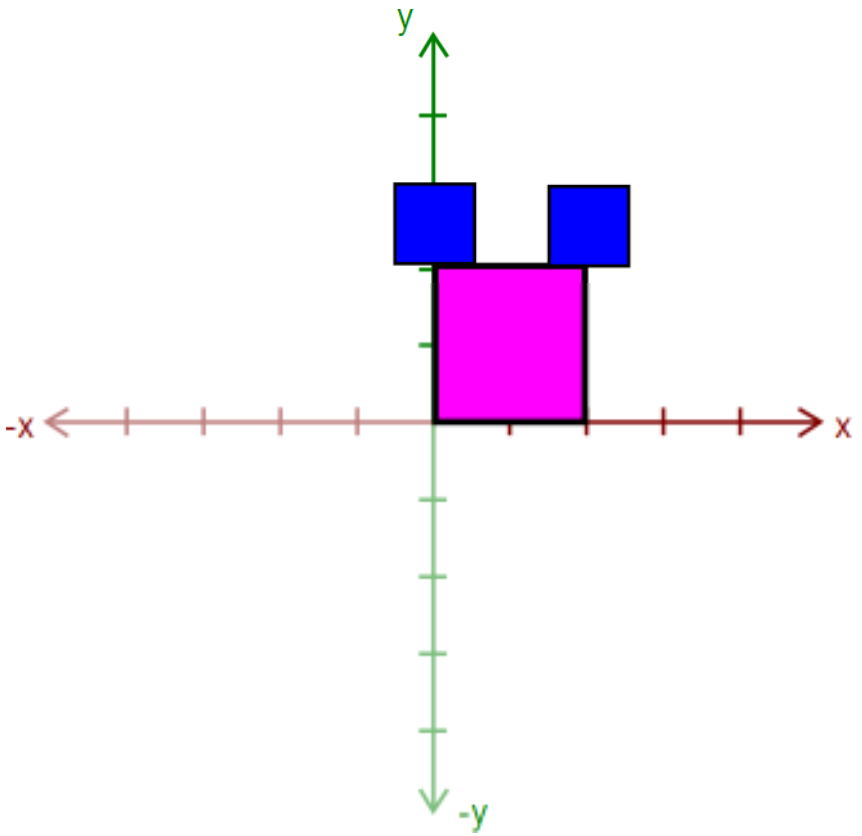
Hierarchical Modeling in OpenGL

- 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?

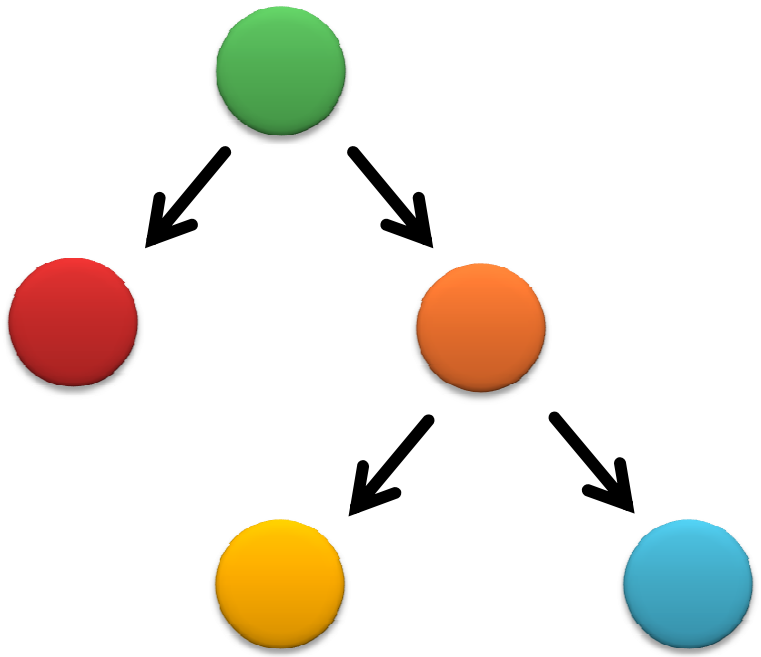


Rule: A Pop For Every Push

- 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() for green, so you can draw orange after drawing red
  - Do the same for orange
- You must draw something at each level.

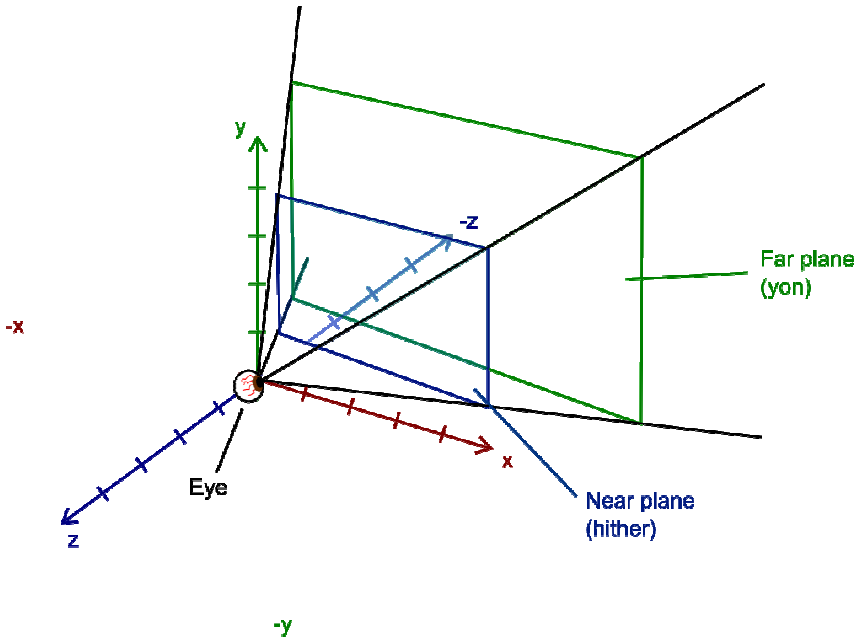


Multiple-Joint Slider

- 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!

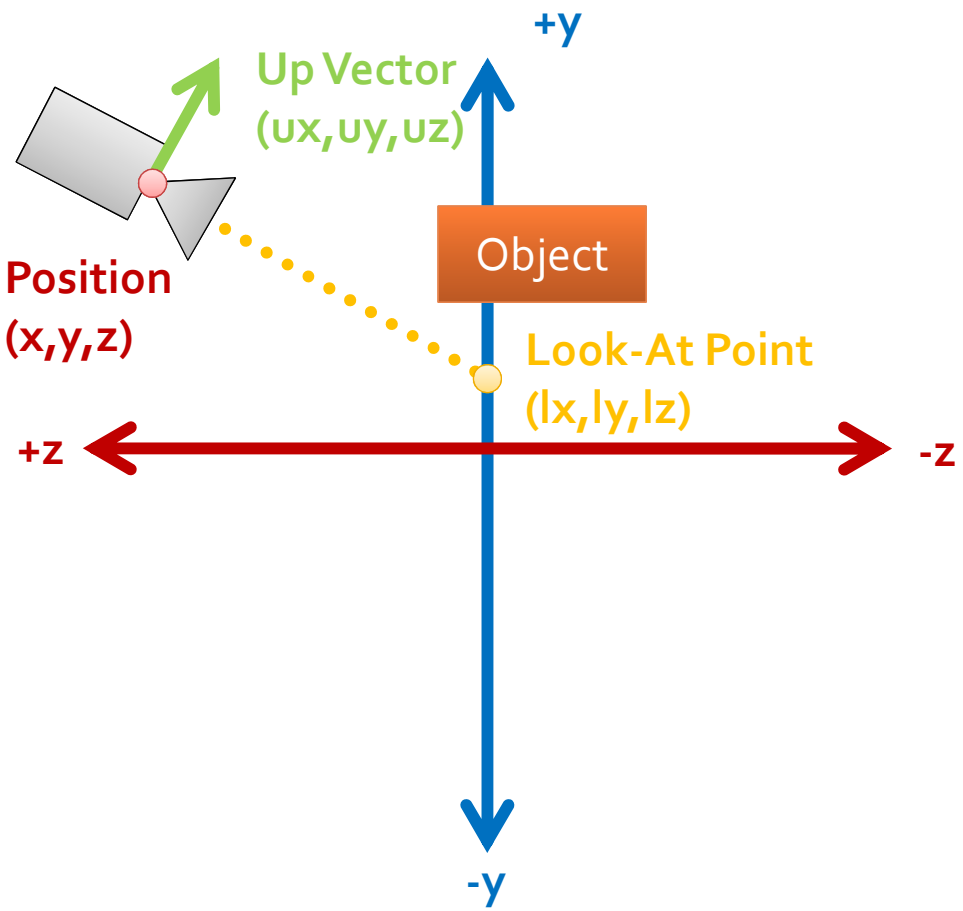
Part 3. gluLookAt

- OpenGL's Camera/Eye
  - **Position:** The origin
  - **Direction:** Looking down the –z axis
  - **Up Vector:** Y-axis corresponds to "up"
- Since we can't move the camera, we move the world instead – it has the same effect.
- A function called gluLookAt() does this.
  - You will replace the call to gluLookAt() in camera.cpp with code that does the same thing.



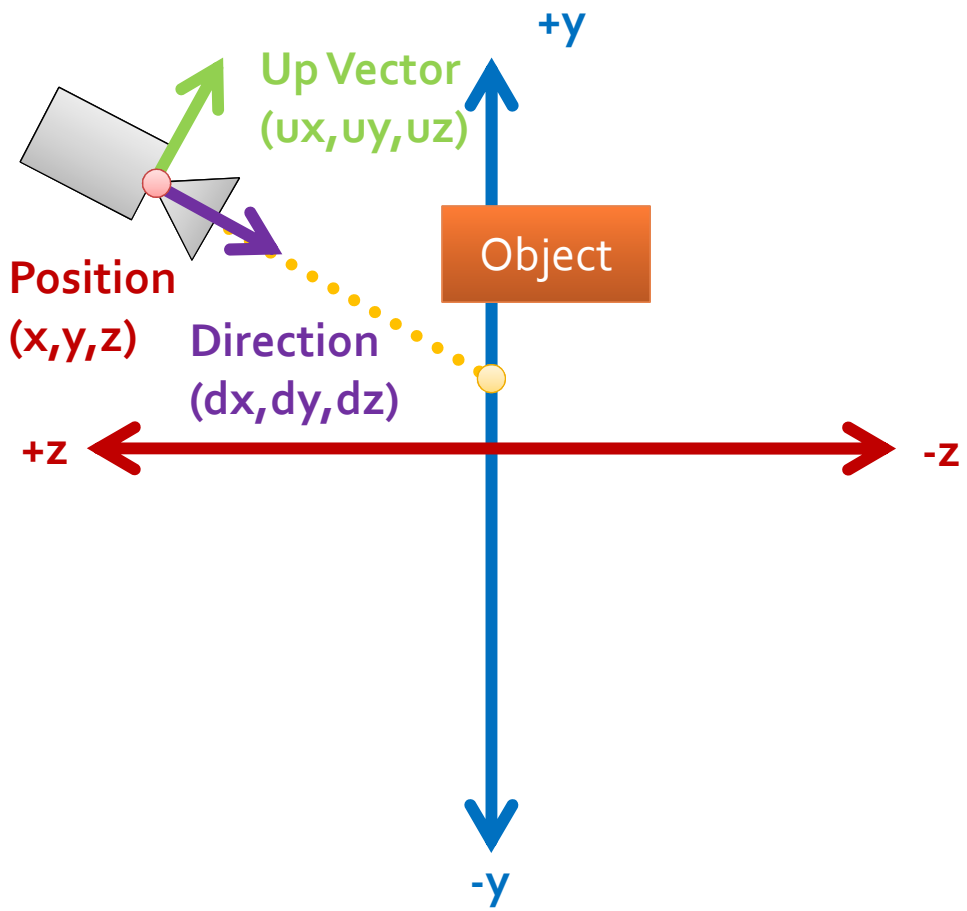
Starting In World Space…

- You are given the camera's:
  - Position
  - Up-vector
  - Look-at point
- Everything is in world space.
- Here's a side view (looking down –x axis)



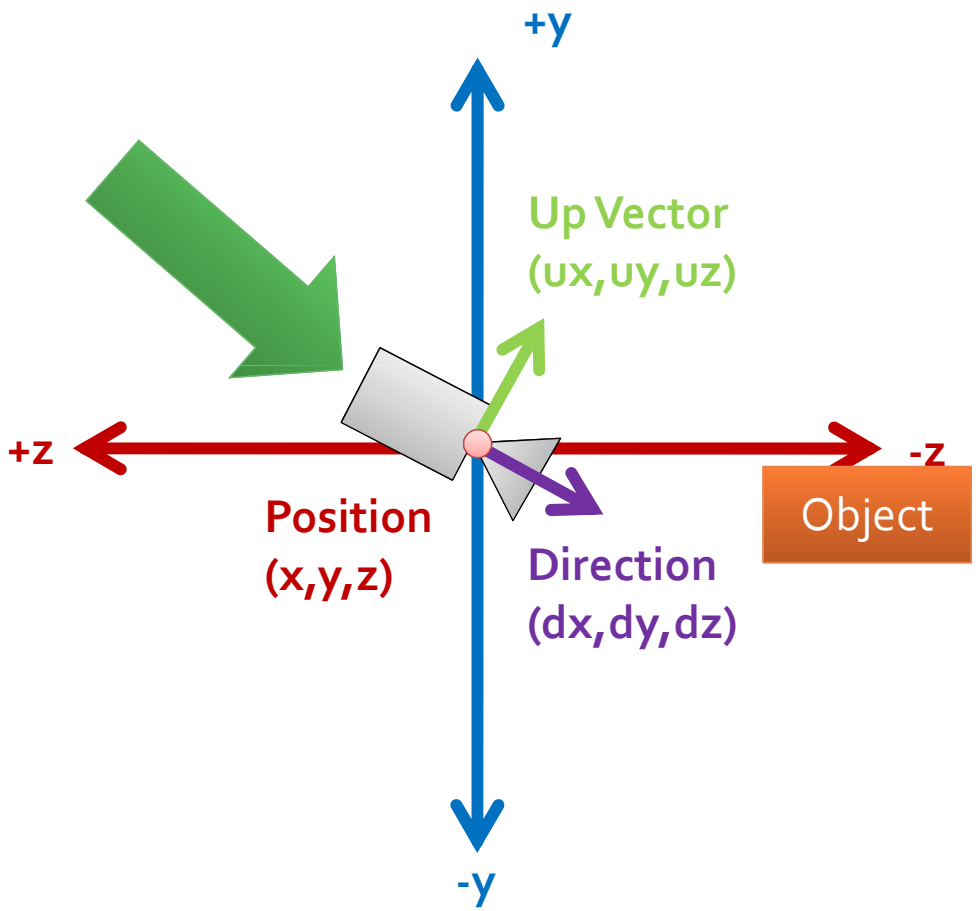
Get Direction

- Use the position and look-at point to get direction
- Ending point – starting point = vector from start to end
- Normalize it



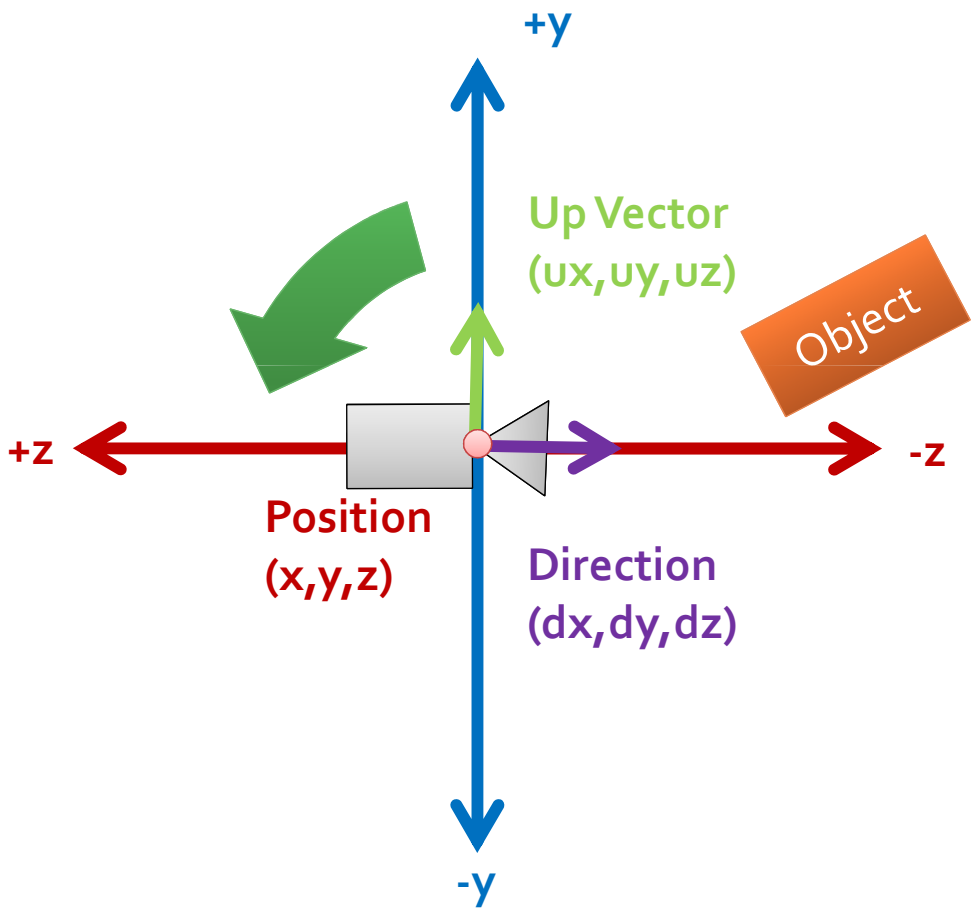
Line Up Camera With Origin

- Apply a **translation** to all vertices, so that the camera's center lines up with the origin.



Rotate World to Line Up Vectors

- Up vector → +y
- Direction → -z
- How?
  - glRotatef() – do the rotations manually
  - glMultMatrixf() – create a custom rotation matrix *(preferred)*



gluLookAt Notes

- See lecture slides for gluLookAt()
  - Make sure you *understand* how works
  - Lots of "magic code" on the Internet
  - You might be asked about it during grading
- Mat.h has a useful matrix class, but you shouldn't need it.

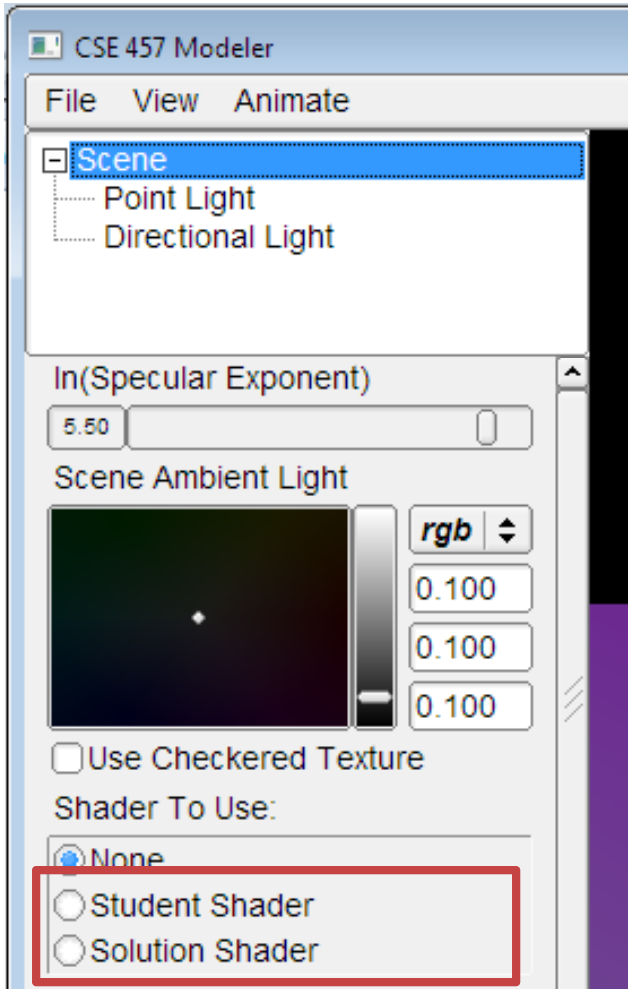
Part 4. Blinn-Phong Shader

- 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 5. 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)

Preparing Your Work Environment

- Make sure that your repository works by:
  - Checking it out
  - 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.

Avoiding SVN 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.
  - TortoiseSVN: when you commit, make sure all the files you added have a checkmark.

Quick Summary

THINGS TO DO

- Partner A: Modeling
  - Part 1: Rendering a Sphere
  - Part 2: Hierarchical Modeling
- Either Partner:
  - Part 3: gluLookAt()
- Partner B: Shading
  - Part 4: Blinn-Phong Shader
  - Part 5: Custom Shader
- 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 check out, 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!