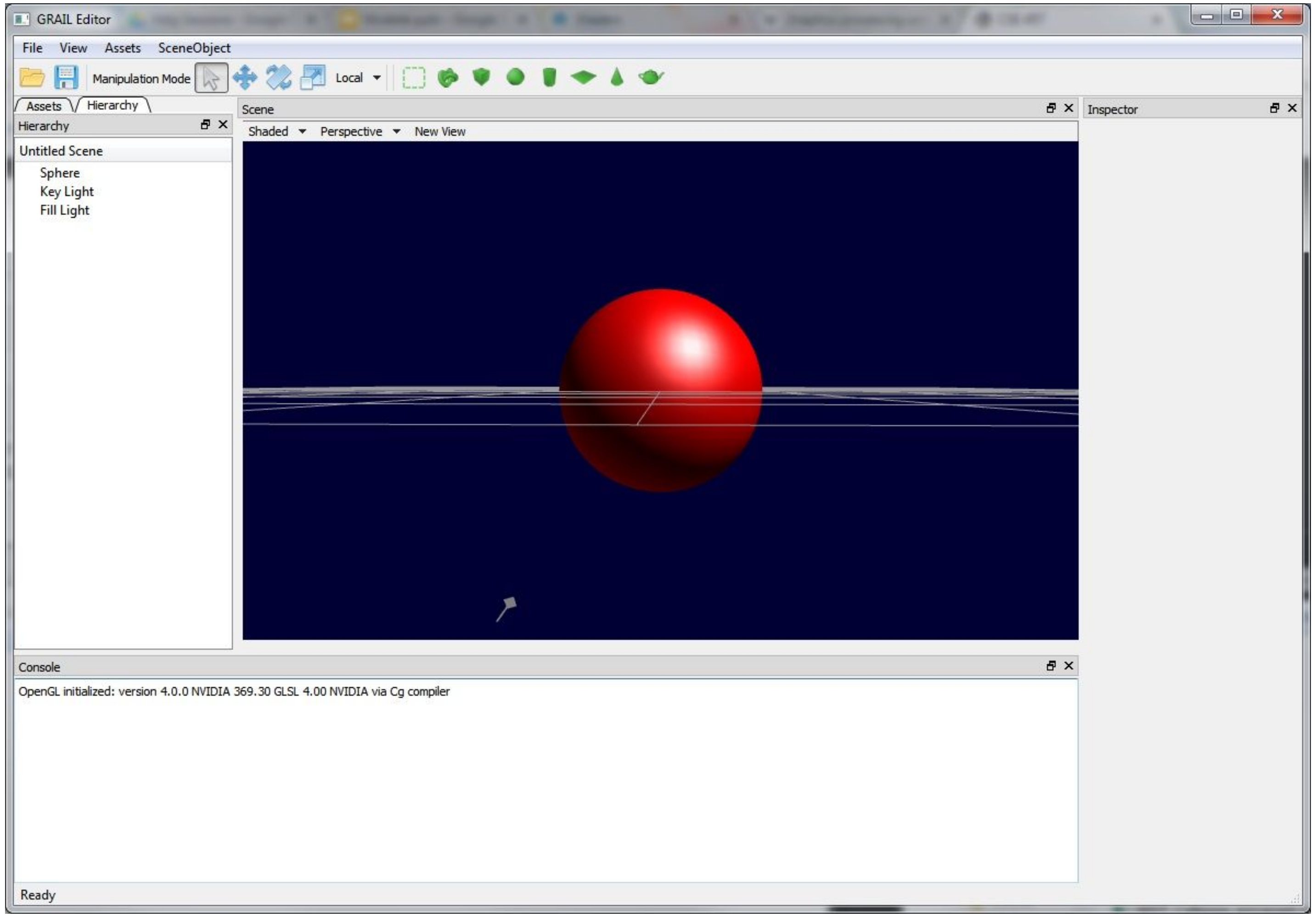
Modeler Help

Demo



OpenGL initialized: version 4.0.0 NVIDIA 369.30 GLSL 4.00 NVIDIA via Cg compiler

Requirements

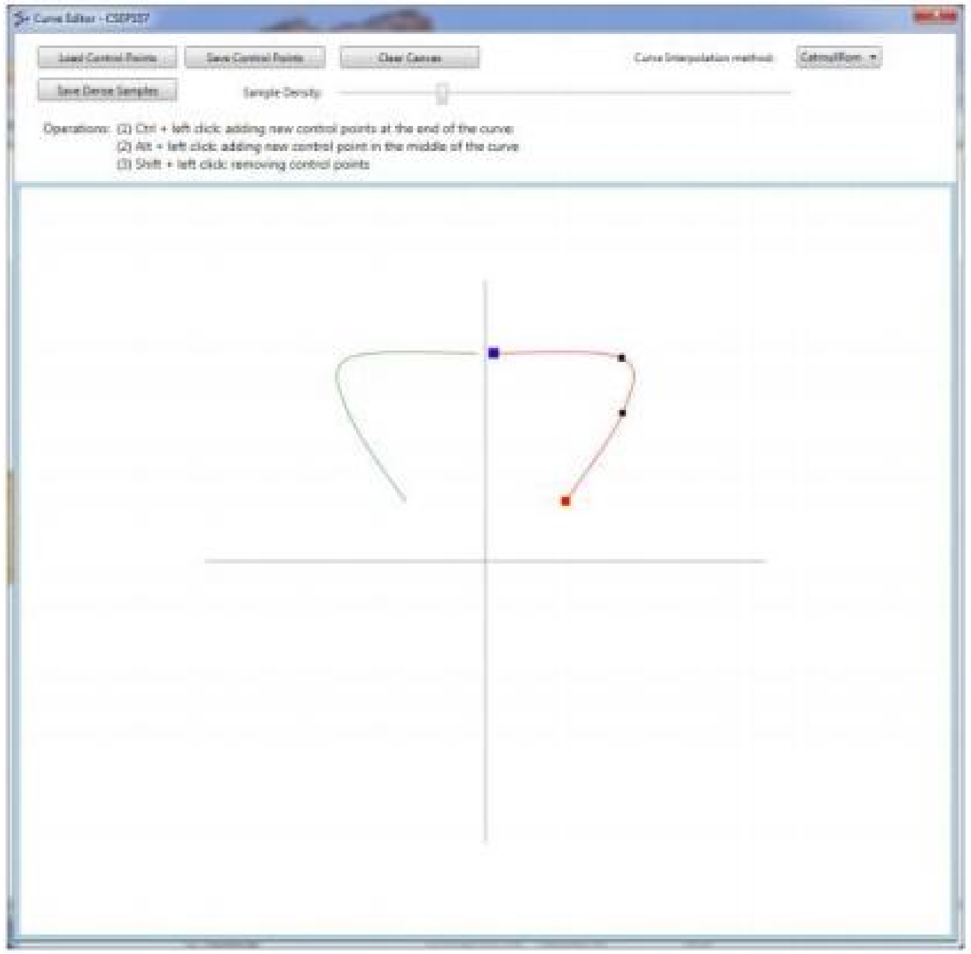
- Hierarchical Modeling
  - Transform Component
  - Scene Graph Traversal
- Surface of Revolution
  - Procedural Triangle Mesh
- Mesh Processing
  - Filtering for Meshes
- Blinn-Phong Point Light Shader
  - Directional Light is implemented for you
- Additional Shader

Artifact

- Create your own hierarchical model out of primitives
  - At least two levels of branching
  - One per partner (if you have one)
- Add textures, materials, shaders, better lighting
- Disable properties you want to be locked (certain channels of rotation maybe)
- Submit a short screencapture (.mp4 format) of you showing off your model!

Curve Editor

- If it helps to have a simpler profile curve while testing, you can draw your own profile curve.
- The **Curve Editor** tool is linked on the project page.
- Ctrl+Left click to add points on one side
- Save the dense point samples into a .apts file.
- **Load Revolution Curve File** in Modeler



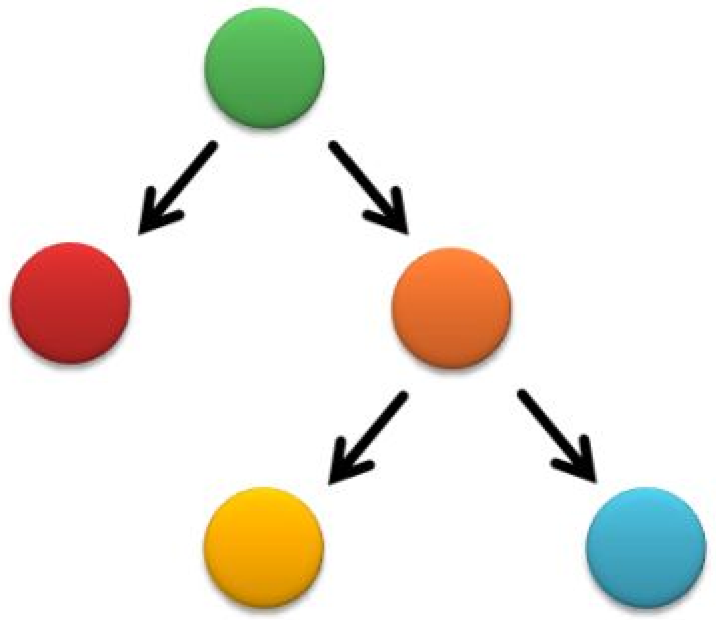
Operations: (1) Ctrl + left click: adding new control points at the end of the curve
(2) Alt + left click: adding new control point in the middle of the curve
(3) Shift + left click: removing control points

Transform

- Class that supplies a transformation matrix via GetMatrix()
    - Used by the rest of the program
- Keeps track of XYZ Translation, Rotation, Scale
- Manipulated by setting Translation, Rotation, Scale or via Translate/Rotate methods that modify current state

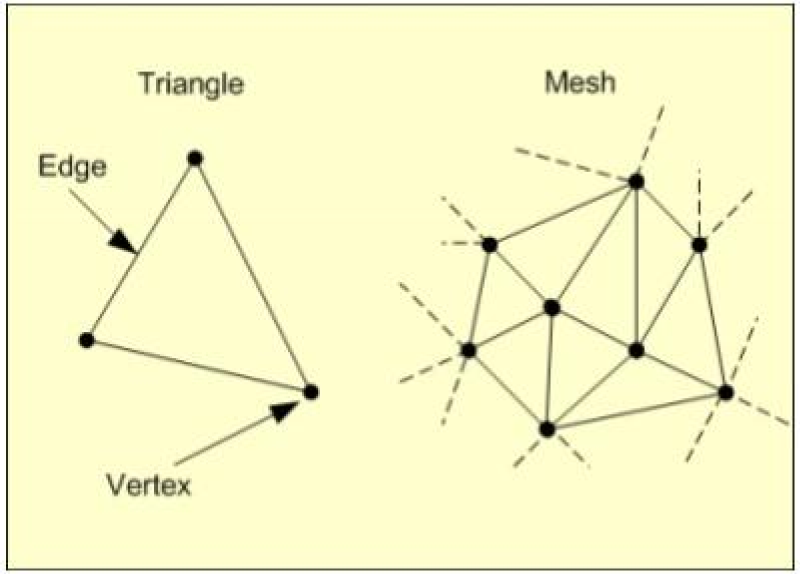
Scene Graph Traversal

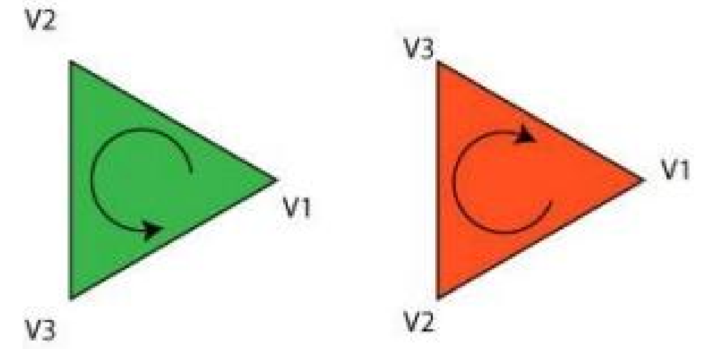
- OpenGL Renderer traverses scene graph depth first
- Before rendering each node, must push the current model matrix onto a stack
- After rendering the node, must pop the model matrix from the stack



Building a Mesh

- A bunch of connected triangles
- Triangle built from three vertices
- Vertex:
  - 3d Position
  - Normal Vector
  - UV Texture Coordinates
- Four giant arrays:
  - Vertex Positions
  - Vertex Normals
  - Vertex UVs
  - Triangles (indices into vertex arrays)
    - Defined as CCW (order matters)



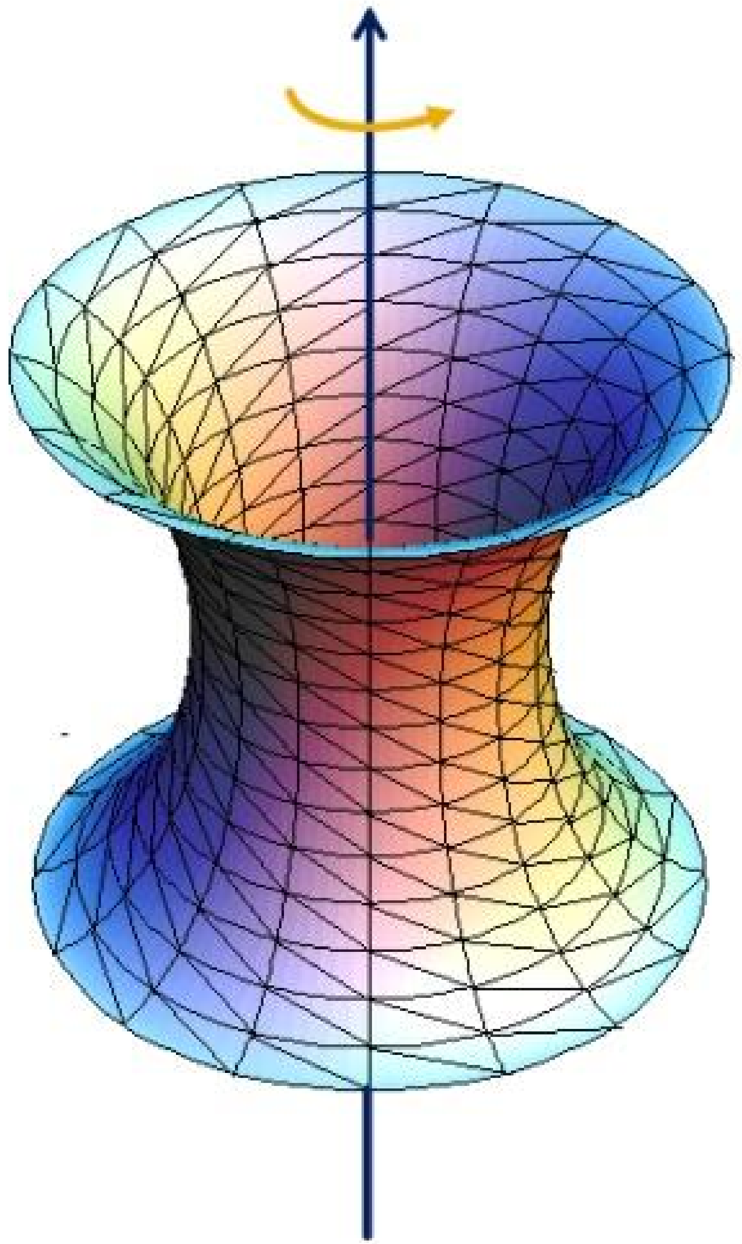


The triangle on the left has a CCW winding order so it will be visible on the screen. The triangle on the right has a CW winding order so you will not see it render on the screen.

(if backface culling is enabled)

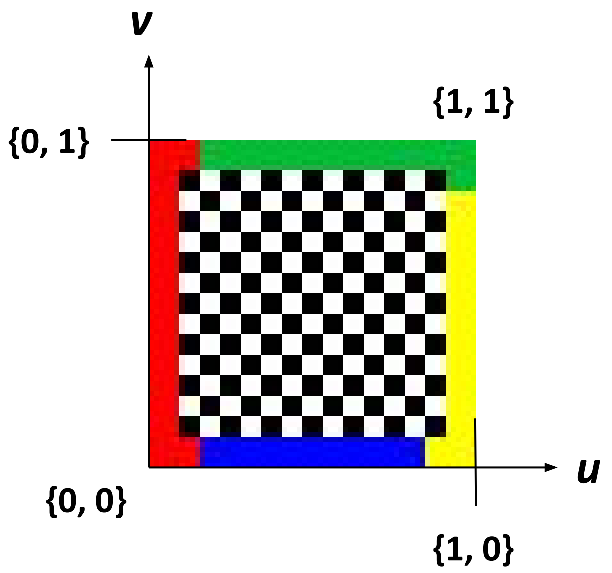
Surface of Revolution

- Divide the surface into “bands”
- Compute vertex positions and normal
  - Using sin(), cos(), in C++ code
  - See the “Surfaces of Revolution” lecture slides for how
- Connect the vertices as triangles
- Computer the texture coordinates



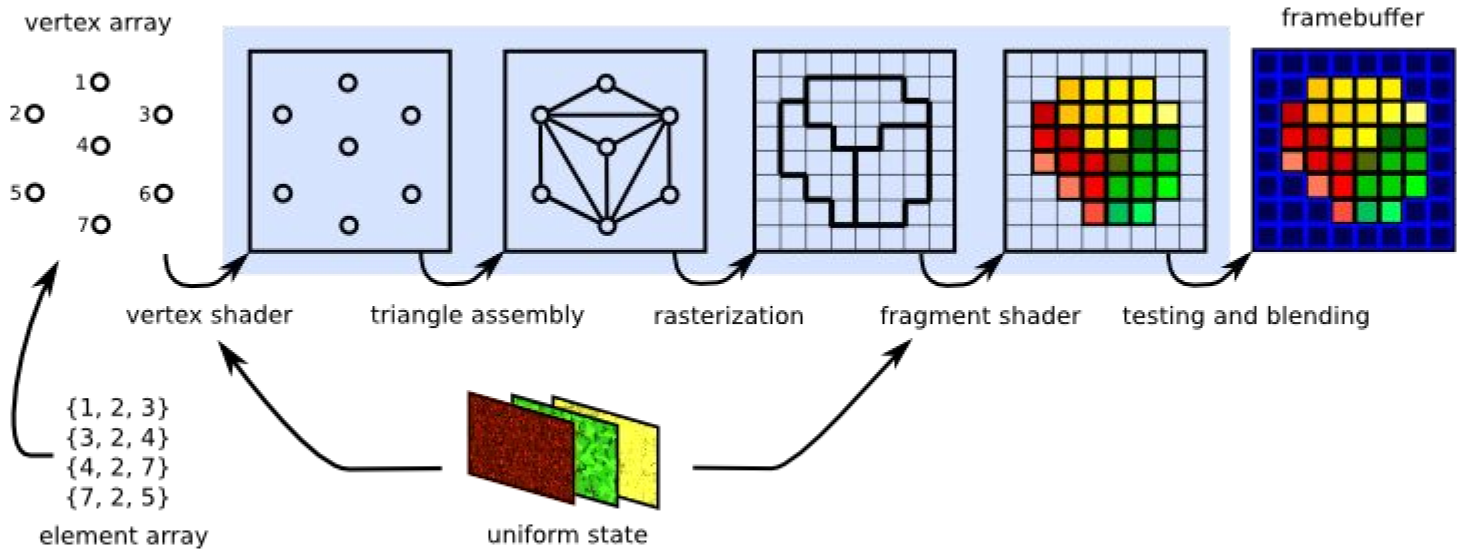
Texture Mapping

- To compute the UV Texture Coordinates, the basic idea is to remap the arclength (curve distance) and longitude to the range 0-1.
  - i.e. longitude for a vertex on the surface can be from 0-360 degrees. The **u** coordinate can be from 0-1.
  - See the lecture slides on “Texture Mapping” for a more detailed explanation
- Each vertex for your surface of revolution must have:
  - Vertex Position
  - Vertex Normal
  - Texture Coordinate Pair

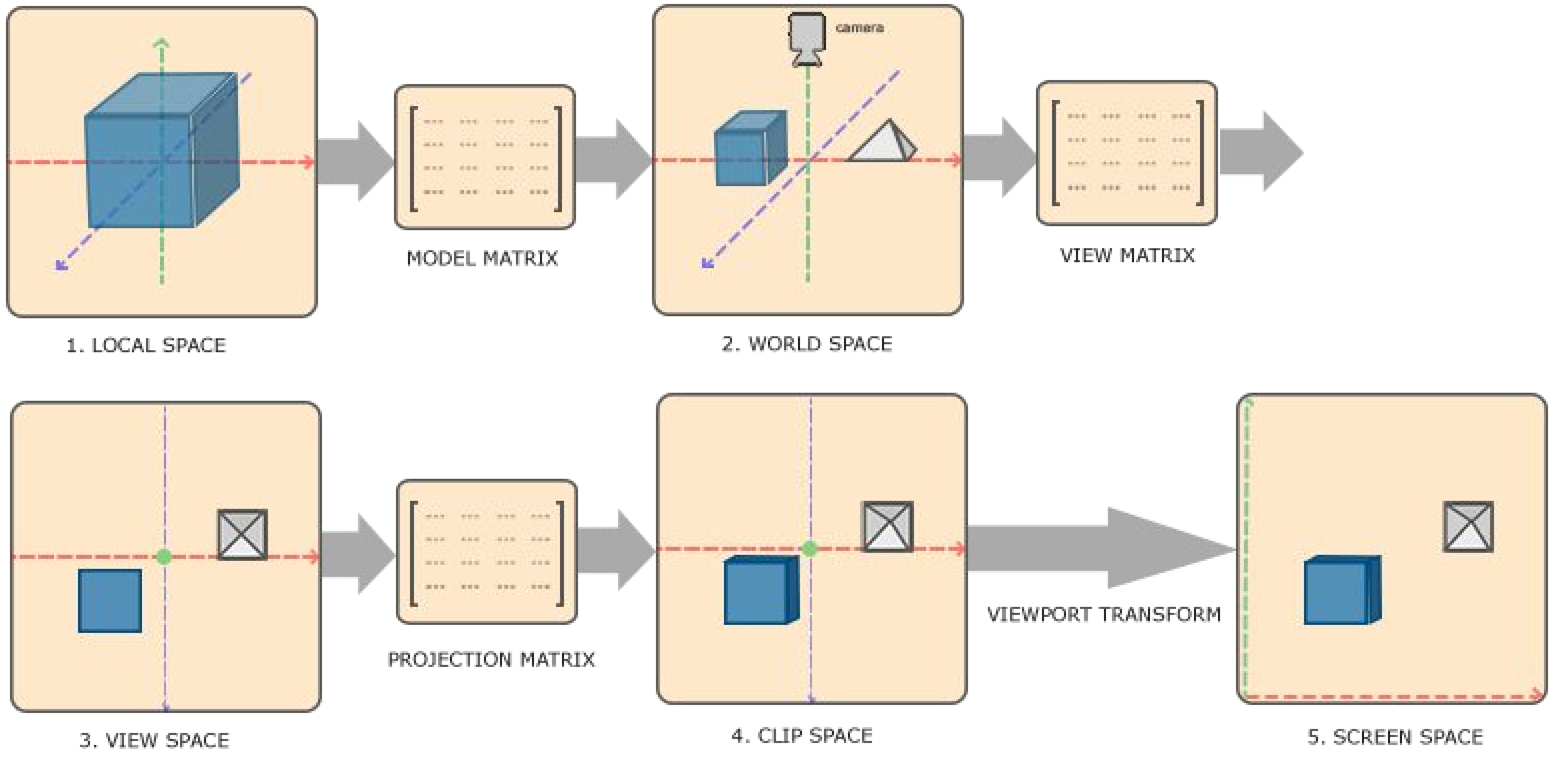


Shading

- GPU is a giant pipeline with many stages
  - Massively parallel compared to CPU
  - Hundreds/Thousands of threads
- Some stages are programmable with "Shaders"



Coordinate Systems



https://learnopengl.com/#!Getting-started/Coordinate-Systems

Blinn-Phong Shader

- We provide a directional light shader in OpenGL Shading Language (GLSL). You must extend it to support point lights.
- Check your work against the sample solution by creating a new material in the sample solution and loading your shader
- Shaders are hard to debug as there is no “print” statement in GLSL. You must use colors to identify what went wrong, and think about why they might appear that way.

Custom Shader

- Anything you want! (ask us first)
- You are required to do 3 whistles worth, but after that you can earn extra credit.
- Shader Resources:
  - http://www.lighthouse3d.com/tutorials/
  - Unity manual on normal mapping
  - Shadertoy has fragment shaders that run on a flat plane only (no vertices other than the 4 for the plane)

Tips

- Start simple when you draw. Draw a few triangles first to make sure it shows up. Then draw a band of triangles.
- Make sure to check out the GLSL Shader Tutorials from the Modeler page. Core GLSL is more architecture (has an explanation of the pipeline and shading stages), and GLSL 1.2 is what we're using (has toon shader tutorial etc).
- More help: https://learnopengl.com/#!Getting-started/Shaders
- More general OpenGL help: https://learnopengl.com/#!Introduction