Modeler

Help Session

Requirements

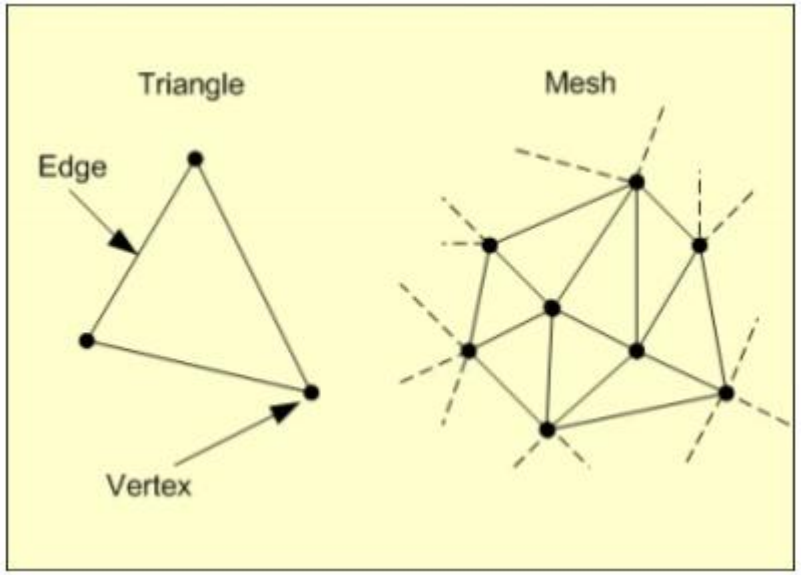
- Surface of Revolution
- Mesh Smoothing / Sharpening
- Hierarchical Modeling
  - At least two levels of branching
  - Add hierarchical UI controls
- Blinn-Phong Point Light Shader
  - Focus on Point Light
  - Directional Light is implemented for you
- Additional Shader
  - Create shaders that is worth at least 3 whistles

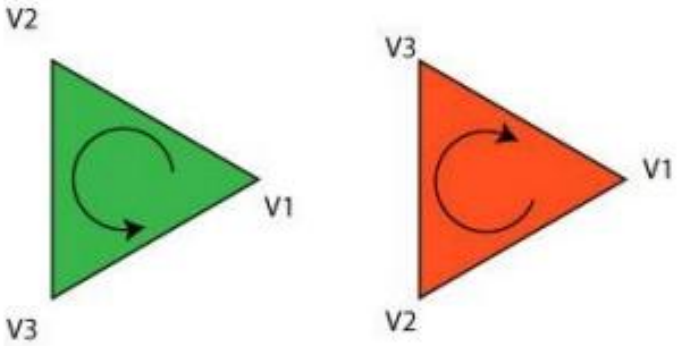
Demo

Surface of Revolution

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)



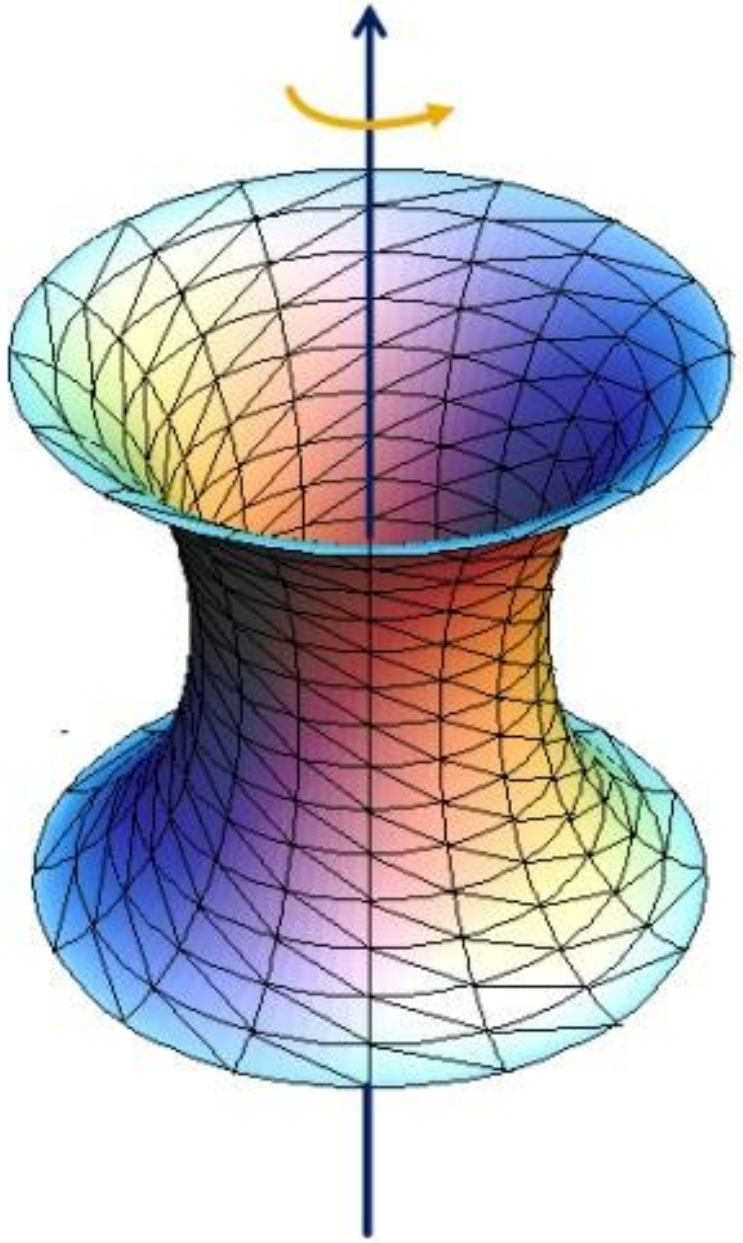


The triangle on the left has a CCW winding order so it will be visible on the screen. The triangle on the right has a CW winding order so you will not see it render on the screen.

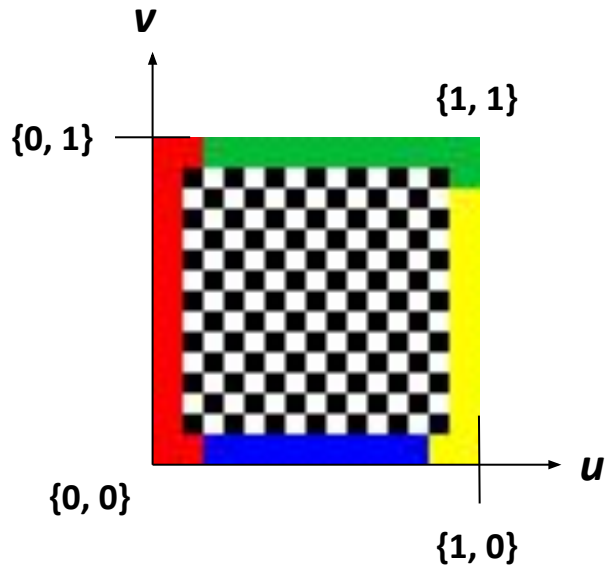
(if backface culling is enabled)

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
- Compute 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

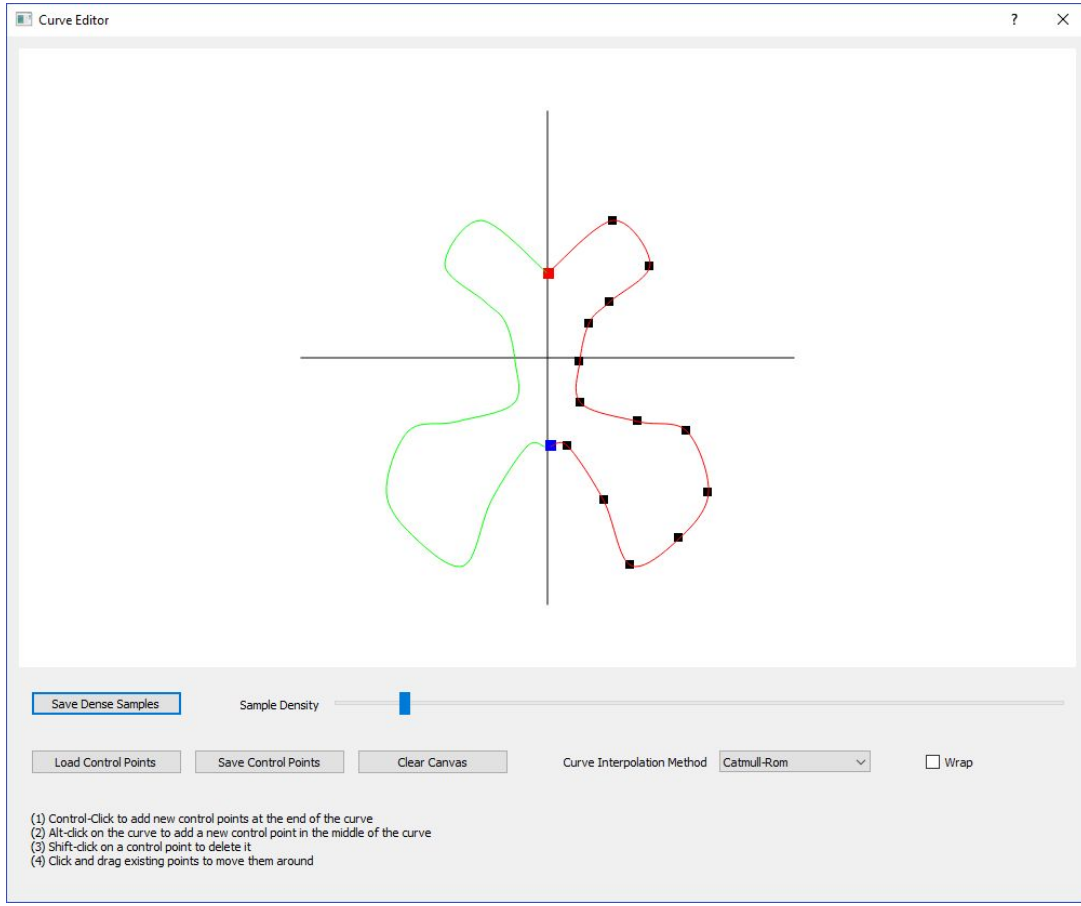
Verify Your Implementation

1. Go to [SceneObject -> Create 3D Object -> Surface of Revolution] to create a new 3D object
2. Select the object your just created, and change the curve property as "assets/curve/sample_curve_1.apts" or "assets/curve/sample_curve_1.apts"
3. Observe if the result is the same as the solution

4. (Optional) Use the curve editor to create a curve on your own and create a new mesh!

Curve Editor (Optional)

- Open the curve editor in the **solution** application (not in your application)
  - [File -> Open Curve Editor]
- Ctrl+Left click to add points on one side
- Save the dense point samples into a .apts file, by clicking [Save Dense Samples]
- Open the saved .apts file in your program to create a new mesh.

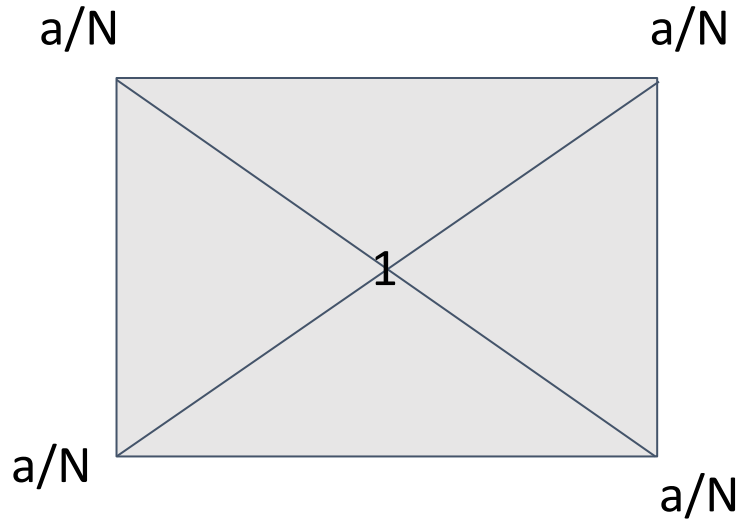


Mesh Smoothing

Mesh Smoothing

- For each vertex of a mesh, take a weighted sum of the vertex and its neighbors to produce a new mesh.
- Control Parameters
  - a: neighbor weight (typically in [-0.5, 0.5])
  - iter: total iterations applied

- Normalization: remember to divide every filter weight by the sum of all the weights



N: # of total neighbors
a: weight

Verify Your Implementation

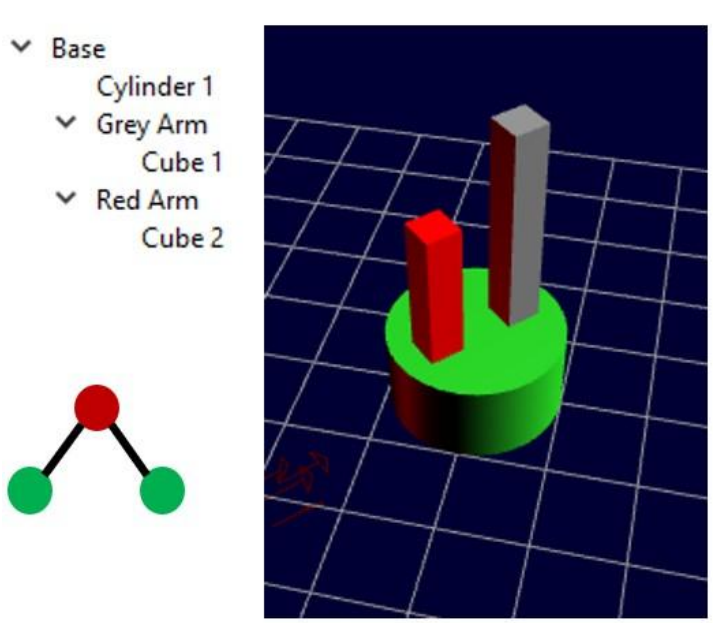
- We suggest using “spikey” mesh to verify your implementation

- Here are steps:
  - Go to [SceneObject->Create 3D Object->Mesh] to create a new 3D mesh
  - The default created mesh is a cube, so you need to change TriangeMesh::Mesh property (on the inspector page) as “Spikey”
  - go to [Assets -> Mesh Processing -> Filter Selected] to apply mesh smoothing to see if the result is the same as the solution.

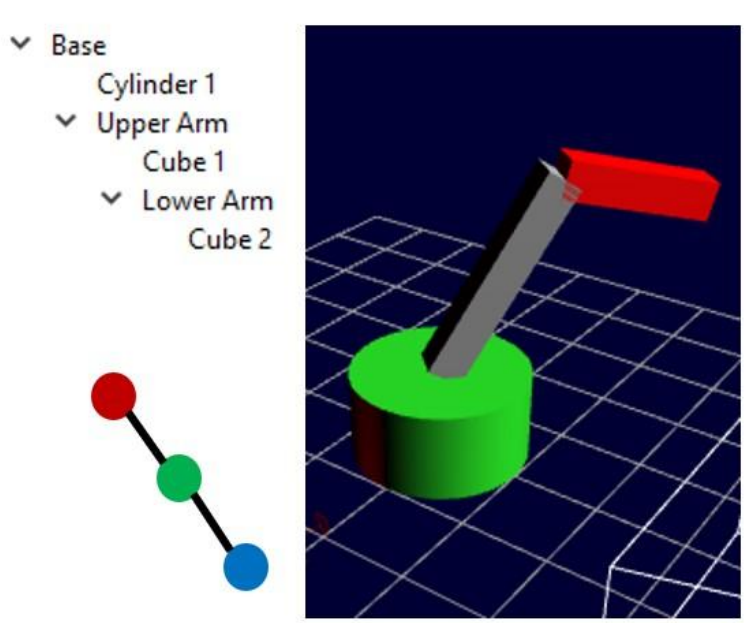
Hierarchical Modeling

Hierarchical Modeling

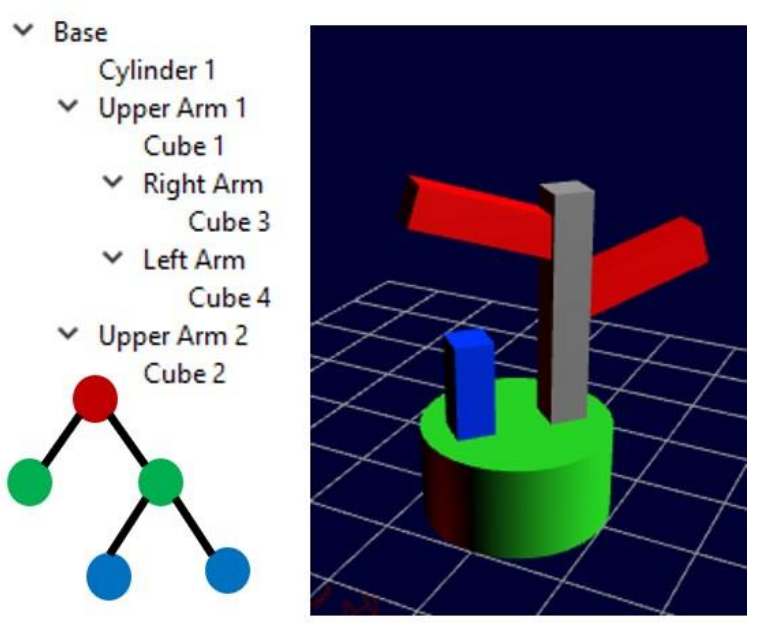
- At least two levels of branching requirement



1 level of branching
Bad Example



0 level of branching
Bad Example



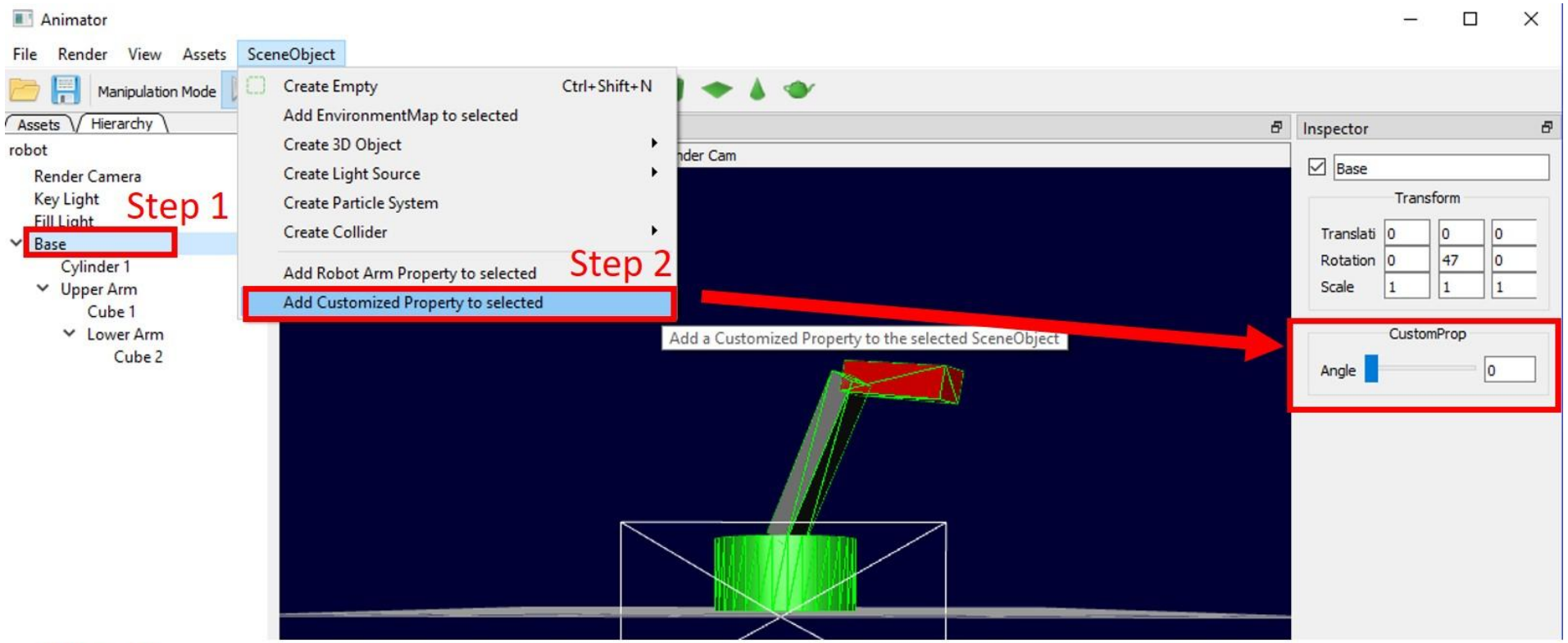
2 levels of branching
Good Example

Demo

Quick Demo ...

Hierarchical Modeling

- Add customized hierarchical UI controls



Hierarchical Modeling

- Add customized hierarchical UI controls

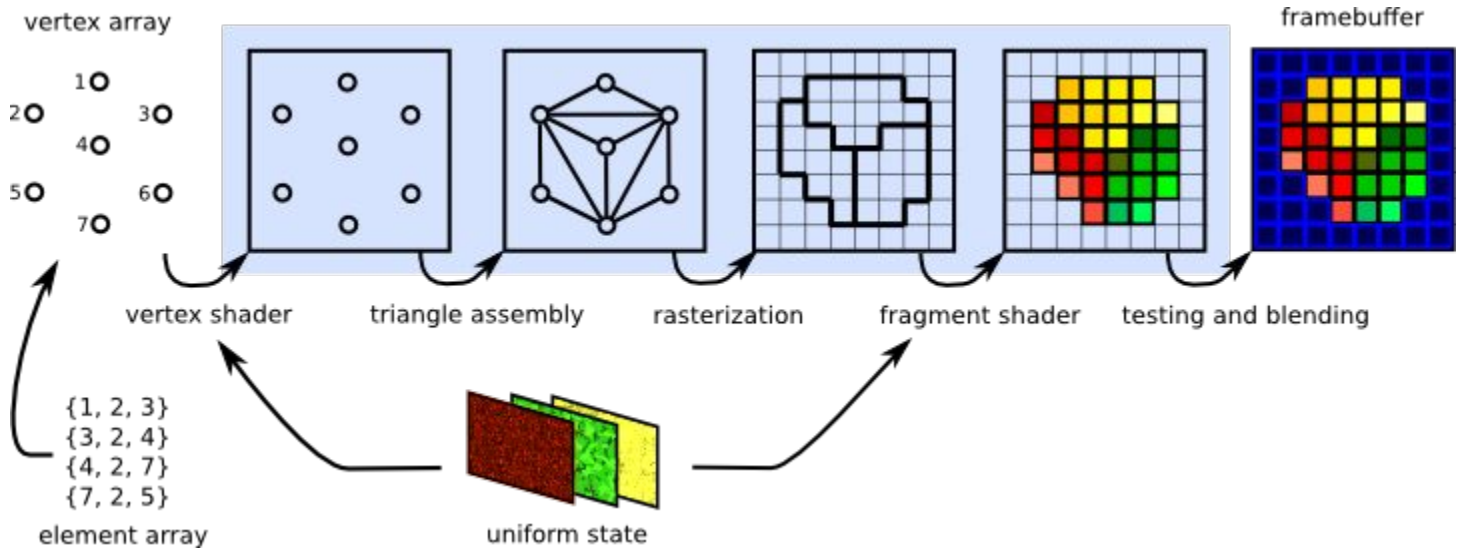
Step 3

```
void CustomProp::OnAngleChanged(double angle)
{
    if (mp_root == NULL)
        return;

    std::cout << "angle = " << angle << std::endl;

    // Modify the code snippet below to realize your slider control
    /*//////////////////////////////////////////////////////////////
    //
    ....
    //////////////////////////////////////////////////////////////*/
}
```

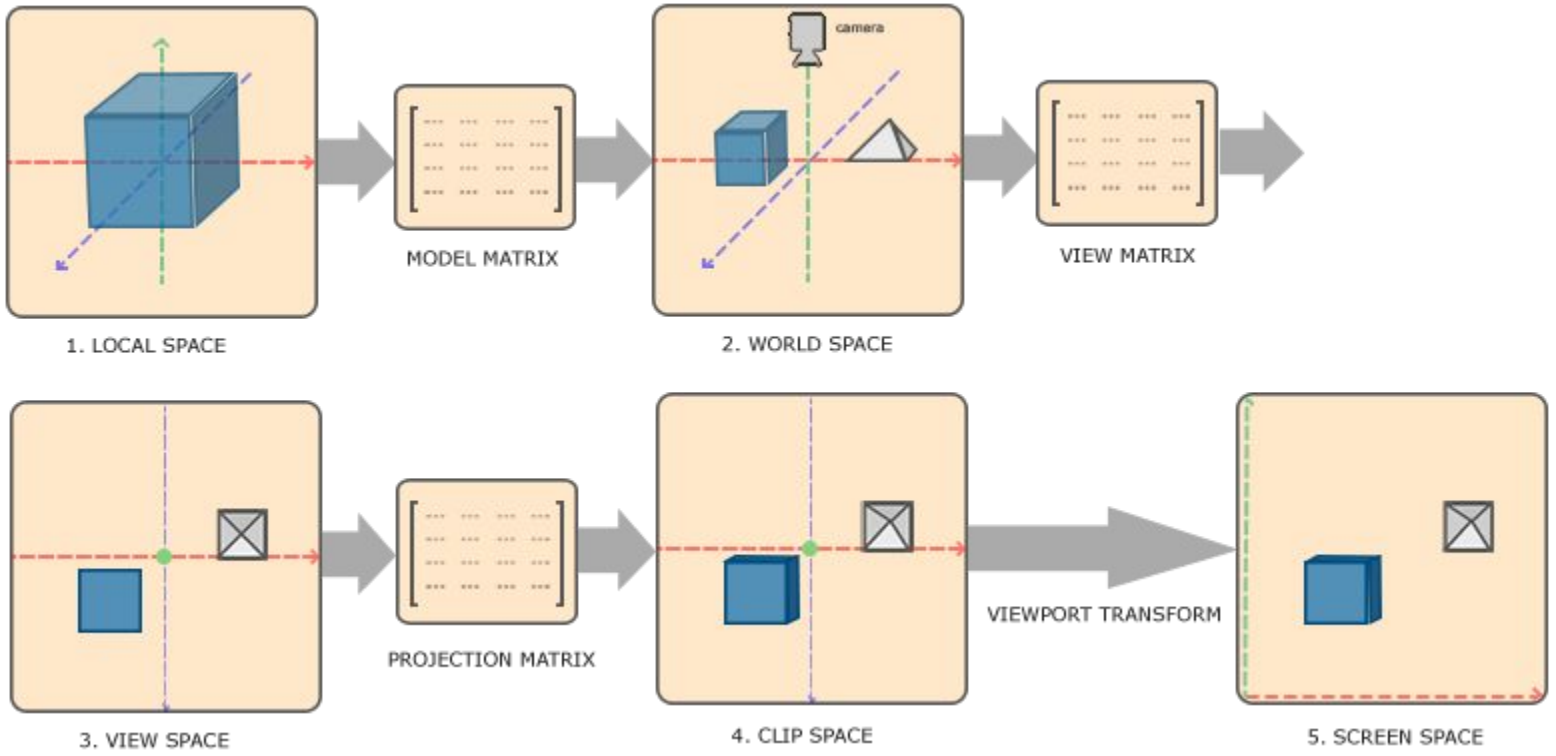
Blinn-Phong Shaders

Blinn-Phong Shader
- Shading

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



Blinn-Phong Shader
- Coordinate Systems



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

Blinn-Phong Shader - Details

- 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 if not listed in the project page)
- 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)

Artifact and Resources

Artifact

- Create your own hierarchical model out of primitives
  - At least two levels of branching
- Add textures, materials, shaders, better lighting

- Submit a short screencapture (.mp4 format) to show off your model!
  - The due date will be after the deadline of modeler. Will announce soon.

Resources

- 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