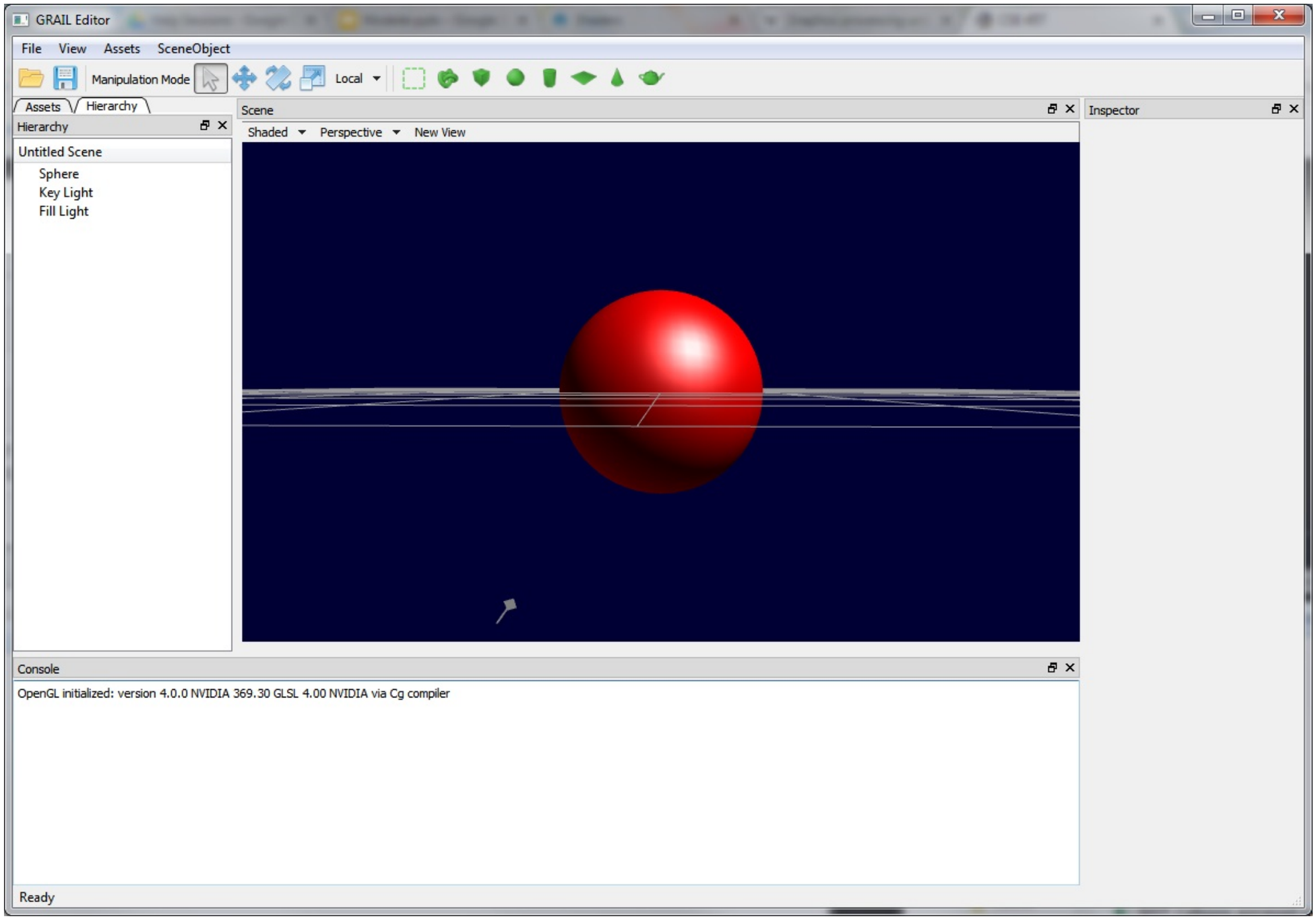
HELP SESSION

MODELER

OUTLINE

- Demo
- Project requirements
- Tips and tricks

DEMO


GRAIL Editor
File
View
Assets
SceneObject
Manipulation Mode
Local
Assets
Hierarchy
Hierarchy
Untitled Scene
Sphere
Key Light
Fill Light
Scene
Shaded
Perspective
New View
Inspector
Console
OpenGL initialized: version 4.0.0 NVIDIA 369.30 GLSL 4.00 NVIDIA via Cg compiler
Ready

REQUIREMENTS

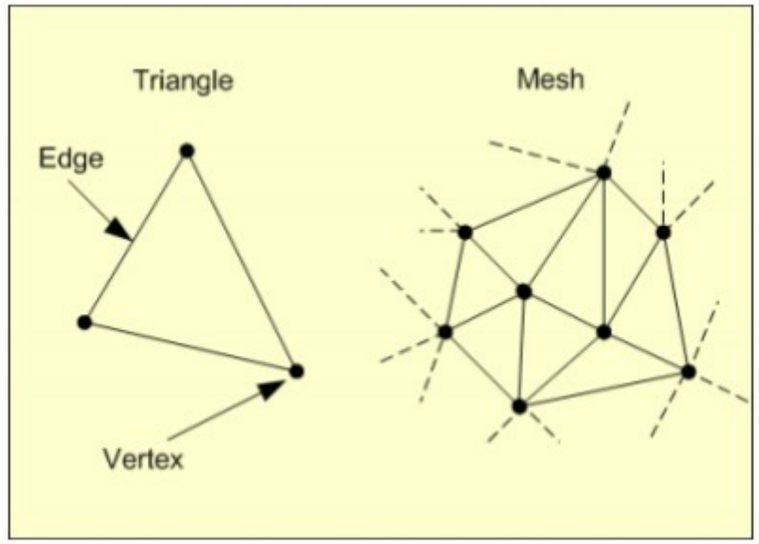
- Surface of Revolution
- Mesh Processing
  - Smoothing / Sharpening
- Hierarchical Modeling
  - At lease two levels of branching
  - Add UI controls
- Blinn-Phong Point Light Shader
  - Implement Point Light
- Additional Shader
  - Implement shaders that is worth as least 3 whistles

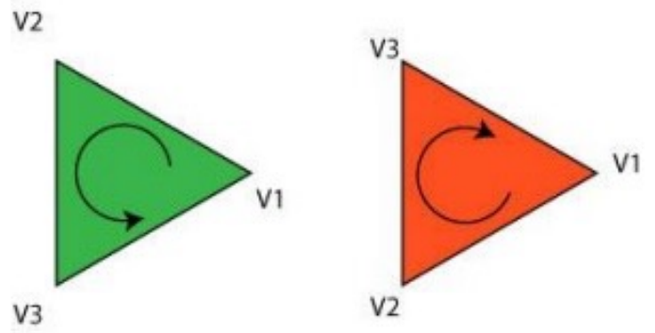
MODELER

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)



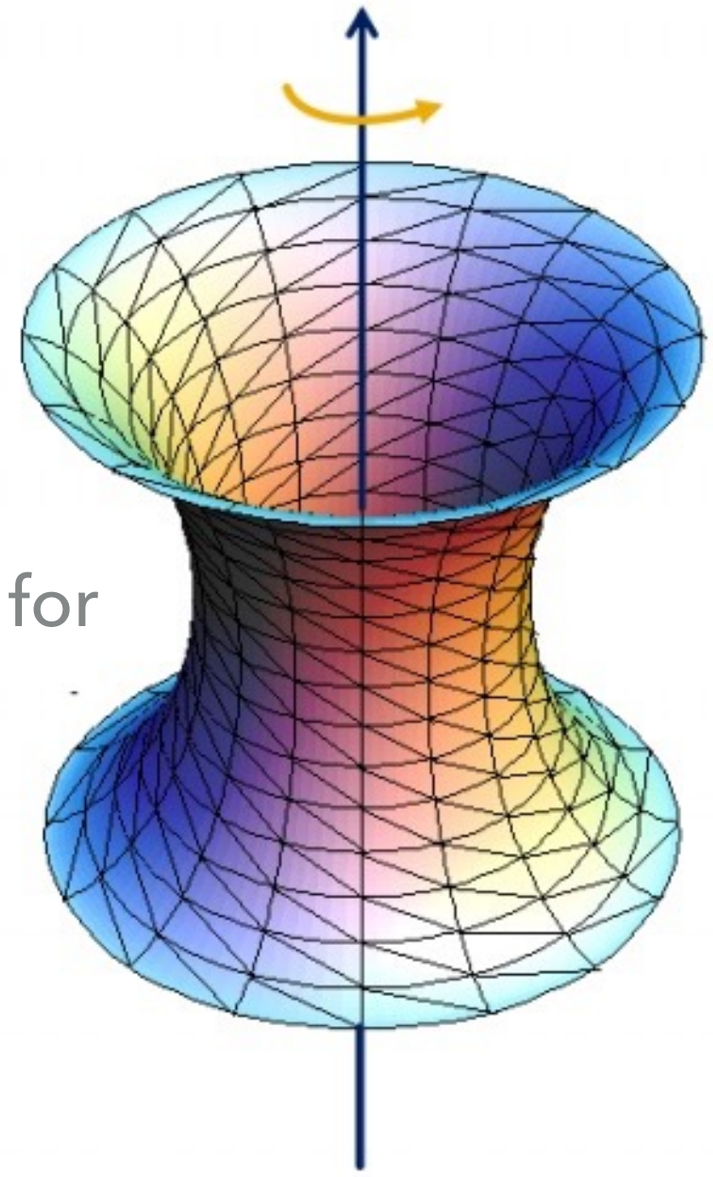


The triangle on the left has a CCW winding order so it will be visible on the screen. The triangle on the right has a CW winding order so you will not see it render on the screen.

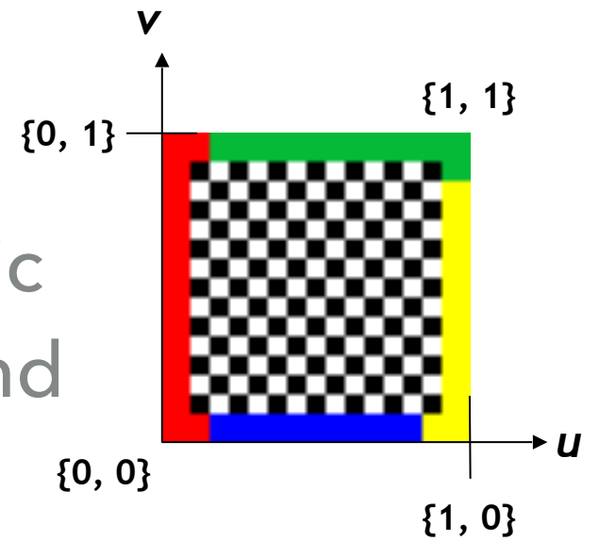
(if backface culling is enabled)

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 computer the UV Texture Coordinates, the basic idea is to remap the arc length (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

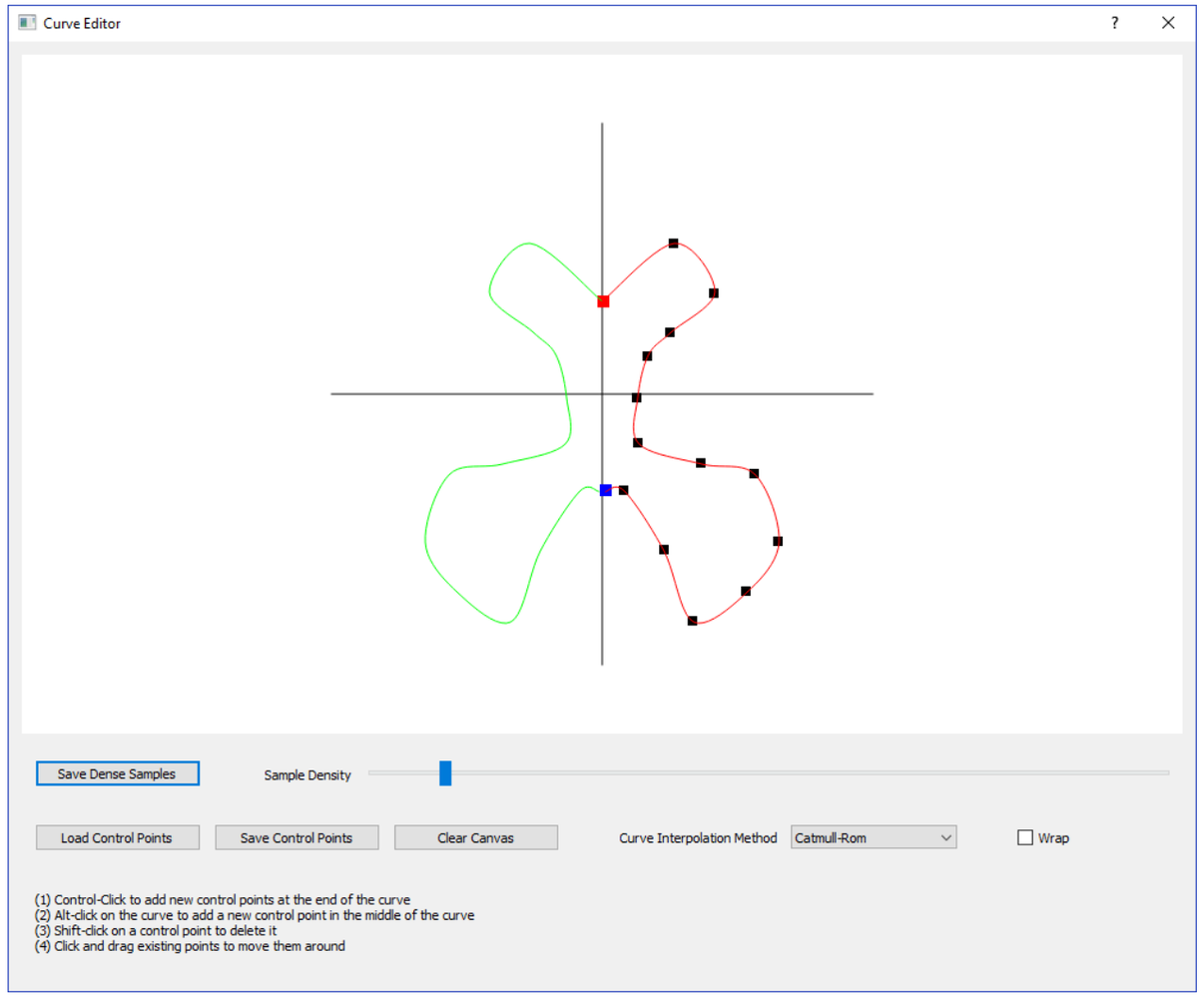
VERIFY YOUR IMPLEMENTATION

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_1/2/4.apts”
3. Observe if the result is the same as the solution

(Optional) Use the curve editor to design 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

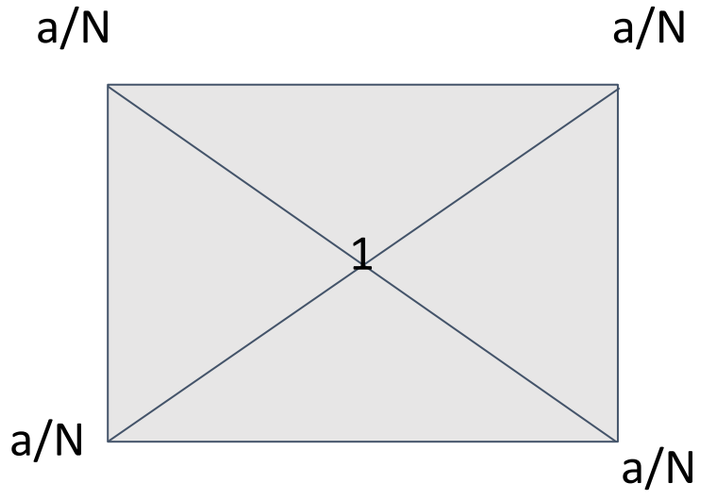


MODELER

MESH PROCESSING

RECOMPUTE VERTEX POSTIONS

- 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

RECOMPUTE VERTEX NORMALS

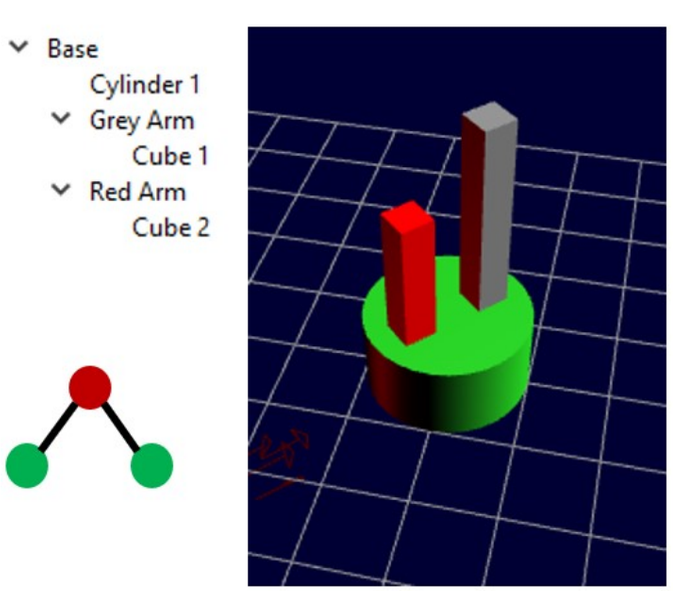
- Recompute vertex normals by averaging the adjacent face normals
  - Use face normal, not vertex normal
  - You can do
    - unweighted average, or
    - face-area-weighted average, which is implemented in solution

MODELER

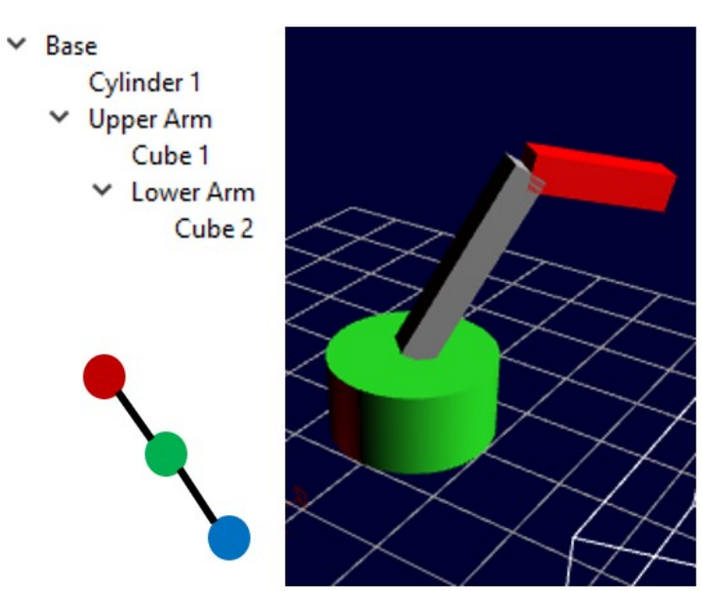
HIERARCHICAL MODELING

HIERARCHICAL MODELING

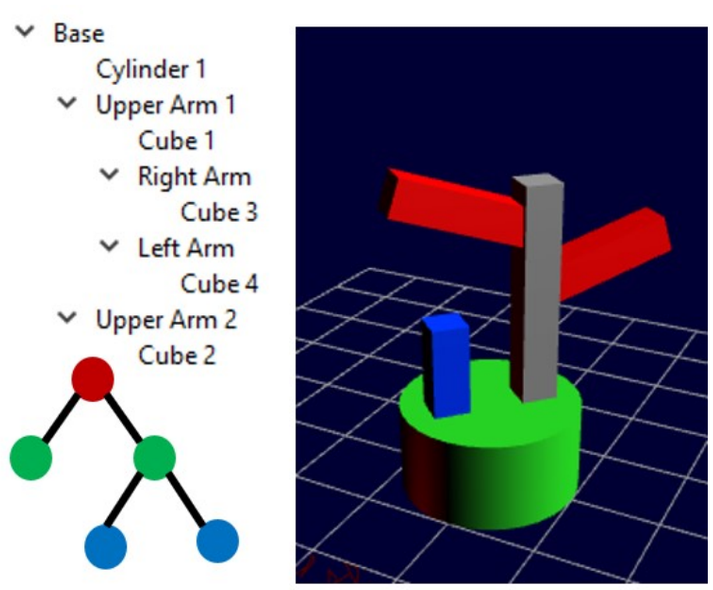
- At least two levels of branching requirement



1 level of branching
Bad Example



0 level of branching
Bad Example

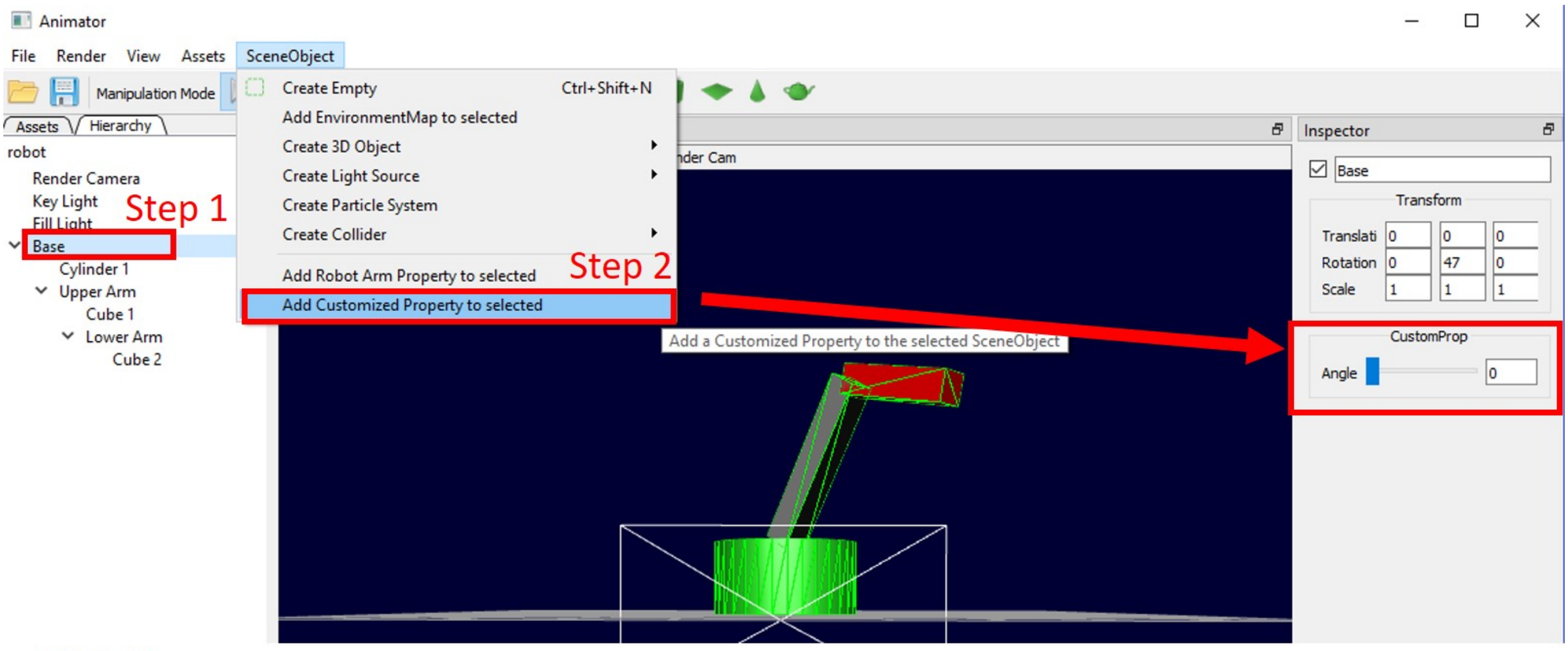


2 levels of branching
Good Example

DEMO

HIERARCHICAL MODELING

- Add customized hierarchical UI controls



HIERARCHICAL MODELING

- Add customized hierarchical UI controls
  - Goto class CustomProp, and edit code snippets

Step 3

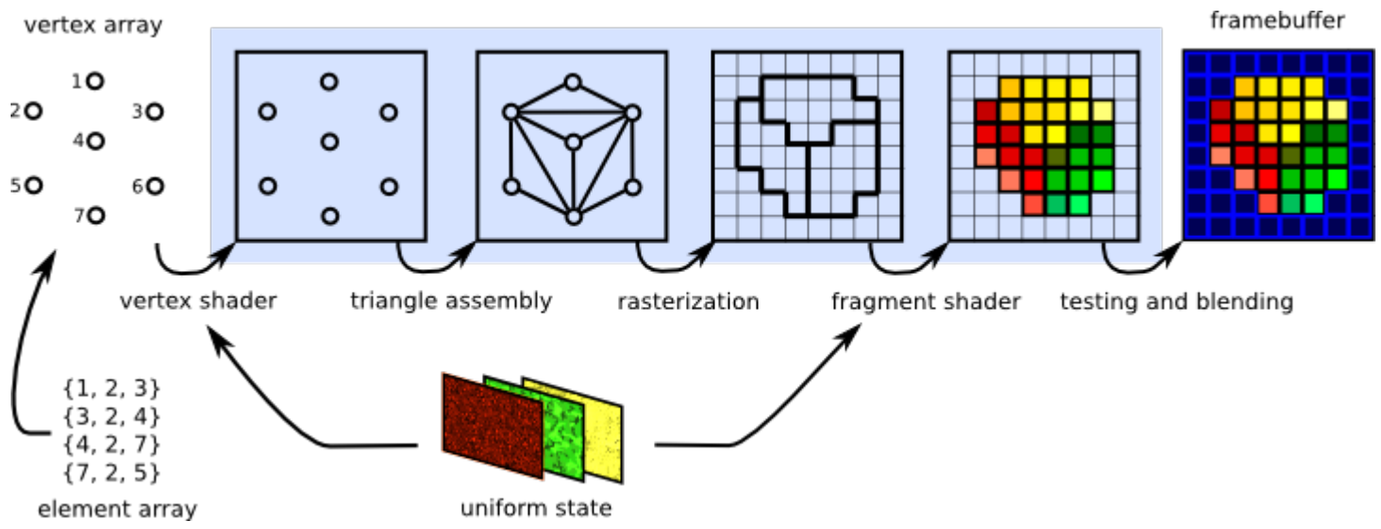
```
void CustomProp::OnAngleChanged(double angle)
{
    if (mp_root == NULL)
        return;

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

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

MODELER

SHADERS

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

- We provide a directional light shader in OpenGL Shader Language (GLSL). You must extend it to support point lights.
- Check your work against the sample solution by loading the test scene
  - point_light_scene.yaml in assets/scene
- 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

- You are required to do 3 whistles worth, but after that you can earn extra credit
- See project page to get an idea about what shader you want to implement

CUSTOM SHADER

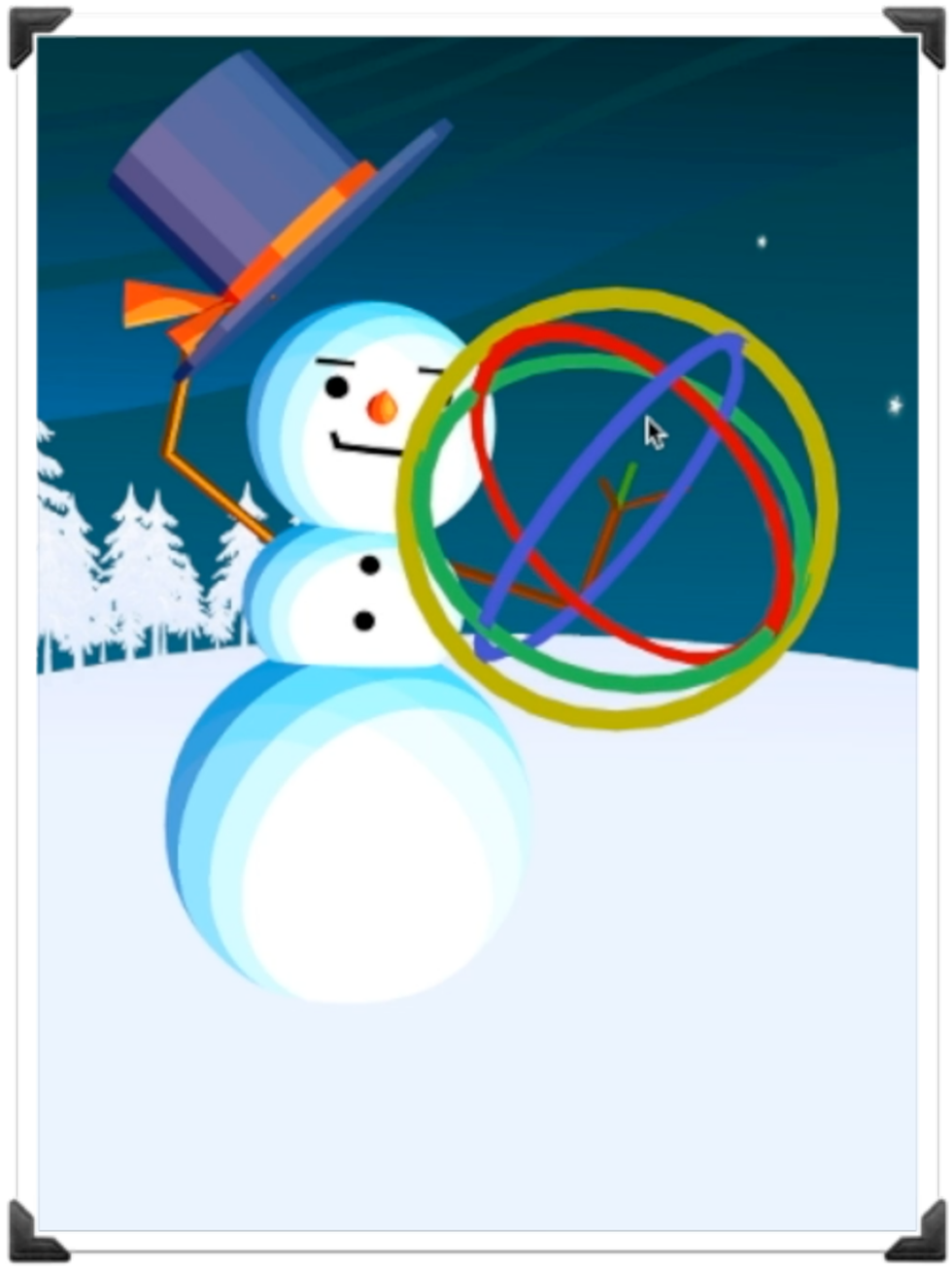
- We have provided several shader skeleton codes and test scenes for you, including
  - Alpha Test Shader (1 whistle)
  - Spotlight Shader (1 bell / 2 whistles)
  - Toon Shader (1 bell / 2 whistles)
- Skeleton shaders are in the folder “assets”
- Test scenes are in the folder “assets/scenes”
- See solution to know how it looks like if implemented correctly

MODELER

ARTIFACTS

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 screen capture (.mp4 format) of you showing off your model!



RESOURCES

- OpenGL
  - https://learnopengl.com/#!Introduction
- Shaders
  - https://learnopengl.com/#!Getting-started/Shaders
  - GLSL Shader Tutorials from the Modeler page
    - http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/
    - GLSL 1.2 is what we're using
  - Toon Shading
    - http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/toon-shading/
  - Normal Mapping
    - https://learnopengl.com/Advanced-Lighting/Normal-Mapping

GOOD LUCK