GPU Code

Write the Vertex Shader (*.v.glsl)

- ☐ Create attribute input for vertex position
- ☐ [Optional] Create attribute inputs for additional vertex attributes
- ☐ Create uniform input for ModelViewProjection matrix (individually or precomputer)
- ☐ [Optional] Create uniform inputs for additional values
- ☐ [Optional] Create varying outputs to pass to next shader in pipeline (geometry or fragment)
- ☐ [Optional] Manipulate vertex for desired effect
- ☐ Set `gl_Position` equal to transformed vertex
  - ☐ Note: If not using a geometry shader, then `gl_Position` must be set to the transformation into clip space.
  - ☐ Note: If using a geometry shader, then the output of the geometry shader must be in clip space. The transformations to world and clip space can happen in either the vertex and/or geometry shader.
- ☐ [Optional] Perform additional calculations for desired effect
- ☐ [Optional] Set varying outputs to respective values

[Optional] Write the Geometry Shader (*.g.glsl)

- ☐ Specify input primitive type
- ☐ Specify output primitive type and maximum number of output vertices
- ☐ [Optional] Create varying input arrays being passed from vertex shader
- ☐ [Optional] Create varying outputs to pass to fragment shader
- ☐ [Optional] Create uniform inputs for additional values
- ☐ For each output primitive
  - ☐ For each output vertex
    - ☐ Set `gl_Position` equal to vertex position in clip space
    - ☐ [Optional] Set varying outputs to respective values
    - ☐ Emit vertex using `EmitVertex()`
  - ☐ End primitive using `EndPrimitive()`

Write the Fragment Shader (*.f.glsl)

- ☐ [Optional] Create varying inputs being passed from prior shader (vertex or geometry)
- ☐ [Optional] Create uniform inputs for additional values
- ☐ Create `vec4` output for fragment color
- ☐ [Optional] Create additional outputs for desired effect
- ☐ [Optional] Perform additional calculations for desired effect
- ☐ Set fragment color output to respective color
- ☐ [Optional] Set additional outputs for desired effect

CPU Code

Compile Each Shader

- ☐ For each shader: Vertex, Geometry, Fragment
  - ☐ ☐ ☐ Create shader handle using `glCreateShader()`
  - ☐ ☐ ☐ Read shader code from file
  - ☐ ☐ ☐ Send shader code to GPU using `glShaderSource()`
  - ☐ ☐ ☐ Compile shader using `glCompileShader()`
  - ☐ ☐ ☐ Check compile status using `glGetShaderiv()` with value `GL_COMPILE_STATUS`
  - ☐ ☐ ☐ Check shader log using `glGetShaderInfoLog()`

Link the Shader Program

- ☐ Create program handle using `glCreateProgram()`
- ☐ Attach vertex shader using `glAttachShader()`
- ☐ [Optional] Attach geometry shader using `glAttachShader()`
- ☐ Attach fragment shader using `glAttachShader()`
- ☐ Link program using `glLinkProgram()`
- ☐ Check link status using `glGetProgramiv()` with value `GL_LINK_STATUS`
- ☐ Check program log using `glGetProgramInfoLog()`
- ☐ For each shader attached to program
  - ☐ Detach shader from program using `glDetachShader()`
  - ☐ Delete shader from GPU using `glDeleteShader()`
  - ☐ Note: These two calls simply delete the compiled object files from the GPU and frees up the shader handle to be reused. Once the shader is linked into a program, the program contains the executable code to run.

Get Uniform and Attribute Locations

- ☐ For all uniforms in program
  - ☐ Get uniform location using `glUniformLocation()`
- ☐ For all attributes in vertex shader
  - ☐ Get attribute location using `glAttribLocation()`

When rendering geometry

- ☐ Set program to be active using `glUseProgram()`
- ☐ For all uniforms in program
  - ☐ Set uniform value using `glUniform*()`
- ☐ Bind vaod and vbod
- ☐ For all attributes in vertex shader
  - ☐ Enable vertex attribute using `glEnableVertexAttribArray()`
  - ☐ Set up vertex attribute pointer using `glVertexAttribPointer()`
- ☐ [When debugging] Check if program is valid using `glValidateProgram()`
- ☐ [When debugging] Check program log using `glGetProgramInfoLog()`
- ☐ Render geometry using `glDrawElements()` or `glDrawArrays()`
- ☐ For all attributes in vertex shader
  - ☐ Disable vertex attribute using `glDisableVertexAttribArray()`

When cleaning up memory

- ☐ Delete program using `glDeleteProgram()`
- ☐ Check delete status by calling `glGetProgramiv()` with value `GL_DELETE_STATUS`