Notes:

- Be sure to read the Programming and Collaboration Policy on course website.
- Any updates or correction will be posted on Piazza, so check there occasionally.
- You must do your own work independently.
- Read through this document carefully before posting questions on Piazza.
- Submit your assignment on GauchoSpace before the due date.
1 Overview

In this assignment, we will move one step forward to simulate the modern graphics pipeline. We have updated our skeleton code by adding the object loader, vertex shader and fragment shader stage, and also support texture mapping. The task for you in this assignment is to interpolate the attributes of triangle, and use them to implement the Blinn-Phong shading model in the phong_fragment_shader function for only the shading part. If you implement both texture and shading, write your code into the texture_fragment_shader function. And write the color data into our frame buffer.

Functions that you need to modify:

- **phong_fragment_shader in main.cpp**: Compute the fragment color according to blinn-phong reflection model.

- **texture_fragment_shader in main.cpp**: Compute the fragment color according to blinn-phong reflection model. Use the texture color as the kd coefficient in the formula.

- **rasterize_triangle(const Triangle& t) in rasterizer.cpp**: Interpolate the needed attributes, and pass them to the fragment shader payload.

Initially, the code is compilable, and you can run the program directly by saying ./Rasterizer, and it will initiate with an already implemented normal shading. Here, you should be able to see the normals of the shape. After you implement your own shaders phong_fragment_shader and texture_fragment_shader, you should look for a variable called active_shader inside the main function. That variable sets the shader to be passed to the rasterizer. Go ahead and update this variable to your shader, and voila! You’re ready to run your cool shader.

2 Getting started

Same as the previous assignments, you can either choose to work in your own system, or to use the virtual machine. Download the skeleton code package for assignment 4. When building the project from command line, this time we require you do the following, create a folder named build under SoftwareRasterizer directory:

$ mkdir build
$ cd ./build

$ cmake .

$ make

This will generate the executable name Rasterizer. Whenever you making changes to the code and want to see the new result, you need to type make again.

We have made several changes to the framework:

1) We included a third party .obj files loader library to load more complex model from files. This exist in the OBJ_Loader.h file. You don’t have to understand how this loader works in detail. Just be aware of that it generate a vector of triangles for us, what we call TriangleList in the program. Meanwhile, we also generate the texture from the image. **Note: If you want to load other objects, you have to change the path manually for now.**

2) Texture class, we introduce a new texture class to create textures from image. And provide the interface to perform the texture color lookup function: Vector3f getColor(float u, float v)

3) We created the Shader.hpp header file and define the fragment_shader_payload, which contains the attributes are used by your fragment shaders. Currently there are three fragment shaders in the main.cpp. The fragment_shader is the example shader that shade the fragment according to its normal vector. The other two are leave for you to implement.

4) The main rendering pipeline starts in
   
rasterizer::draw(std::vector<Triangle> &TriangleList).
We perform a series of transformation here. Usually it is the job for the vertex shader. Then we call the rasterize_triangle function.

5) The rasterize_triangle function is similar to what you did in assignment 3. But instead set a fixed color, we write the color computed by our fragment shader into the frame buffer. This require you set up the fragment shader payload first using the interpolated attributes and call the fragment shader to get result.

After you used the code from last assignment and use the default normal fragment shader, you will see this:
The shading result after you implement blinn-phong shading will like:

![Blinn-Phong Shading Example](image.png)

Result after implement textured shading will like:

![Textured Shading Example](image.png)
3 Grading and Submission

Grading:

- [5 points] Submission is in the correct format, include all necessary files. Code can compile and run.

- [10 points] Attributes interpolation:
  Given the attributes at each vertex, we need to interpolate them to get the attribute for the pixels inside. Interpolate color, normal, texture coordinates, view space position (shading position) correctly and pass them to the fragment_shader_payload structure.

- [20 points] Blinn-phong reflection model:
  Implement the shading model correctly in the fragment shader, which is represented by the phong_fragment_shader function in main.cpp.

- [5 points] Texture mapping:
  Copy the code from the phong_fragment_shader to texture_fragment_shader, replace the kd coefficient with the color from your texture mapping result. You only need to sample the nearest neighbour value for now.
• [Bonus 3 points] Load more models: Find other usable .obj files for our framework. Submit the image you rendered and save the models in the /models folder.

• [Bonus 5 points] Bilinear texture interpolation: Use the bilinear interpolation to sample the texture value, implement a new function in the Texture class: Vector3f getColorBilinear(float u, float v). Call it from your fragment shader. Submit two comparison rendered images.

Submission:
After you finish the assignment, clean your project, remember to include the CMakeLists.txt in your folder, whether you modified it or not. Add a short README.md file in the directory, write down your full name and perm number inside it. Tell us whether you did the bonus part or not, and **briefly describe what you have implemented in each function.** Then compress the entire folder and rename it with your name, like “Lei.zip”. Submit the .zip file on GauchoSpace.