Assignment 3 Pipeline and Shading

1 Overview

In this assignment, we will implement the graphic pipeline. We add the Object Loader function, Vertex Shader and Fragment Shader. We also support texture mapping.

Tasks:

1) Edit function rasterize_triangle(const Triangle& t) in rasterizer.cpp: implement the interpolation method (similar to assignment 2) to interpolate the normal, color, and texture value.
2) Edit function get_projection_matrix() in main.cpp: copy the projection matrix from previous assignments. After your implementation of this two steps, you can compile your code and run ./Rasterizer output.png normal to see the result of normal map.
3) Edit phong_fragment_shader() in main.cpp: implement the Blinn-Phong model to calculate fragment color.
4) Edit texture_fragment_shader() in main.cpp: in Blinn-Phong model, treat the texture color as kd, implement the texture shading fragment shader.
5) Edit bump_fragment_shader() in main.cpp: implement

the bump mapping. (Carefully read the notations in the source code).

6) Edit displacement_fragment_shader() in main.cpp: implement the displacement mapping.

2. Compile

The compiling steps are same with previous assignments. After you compile the code, you will get a executable file named as Rasterizer.

To utilize texture shader,

Run ./Rasterizer output.png texture

To utilize normal shader:

Run ./Rasterizer output.png normal

To utilize blinn-phong shader:

Run ./Rasterizer output.png phong

To utilize bump shader:

Run ./Rasterizer output.png bump

To utilize displacement shader:

Run ./Rasterizer output.png displacement

3. Code frame

1) We utilize a library to help use load .obj file. This

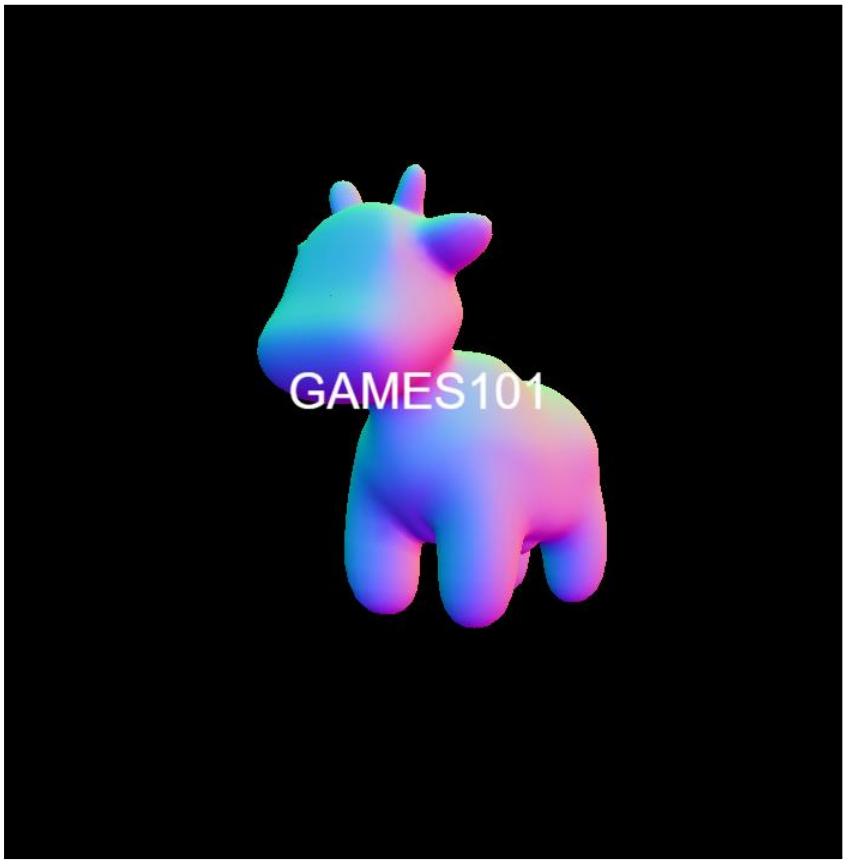
library will load .obj file and pass the values to the vector named as TriangleList. Each triangle has its own normal value and texture coordinates. (The textures will be loaded at the same time, if you want to try other models, please change the loading path).

2) We implemented a new class called Texture, this class provide a function to get texture color with texture coordinates: Vector3f getColor(float u, float v).
3) WE create a class Shader.hpp and defined fragment_shader_payload, the Fragment Shader parameters are included.in this function. In main.cpp, three fragment shaders are included, the fragment_shader will utilize normal as the color to do shading, the others will be implemented by you.
4) The pipeline will start in rasterizer::draw(std::vector<Triangle> &TriangleList). We will do transformations, then we call rasterizer_triangle.
5) The function rasterize_triangle is similar to the one in assignment2. The difference is the setting value will not be an constant. It will use the barycentric coordinates to calculate the normal, color, texture,

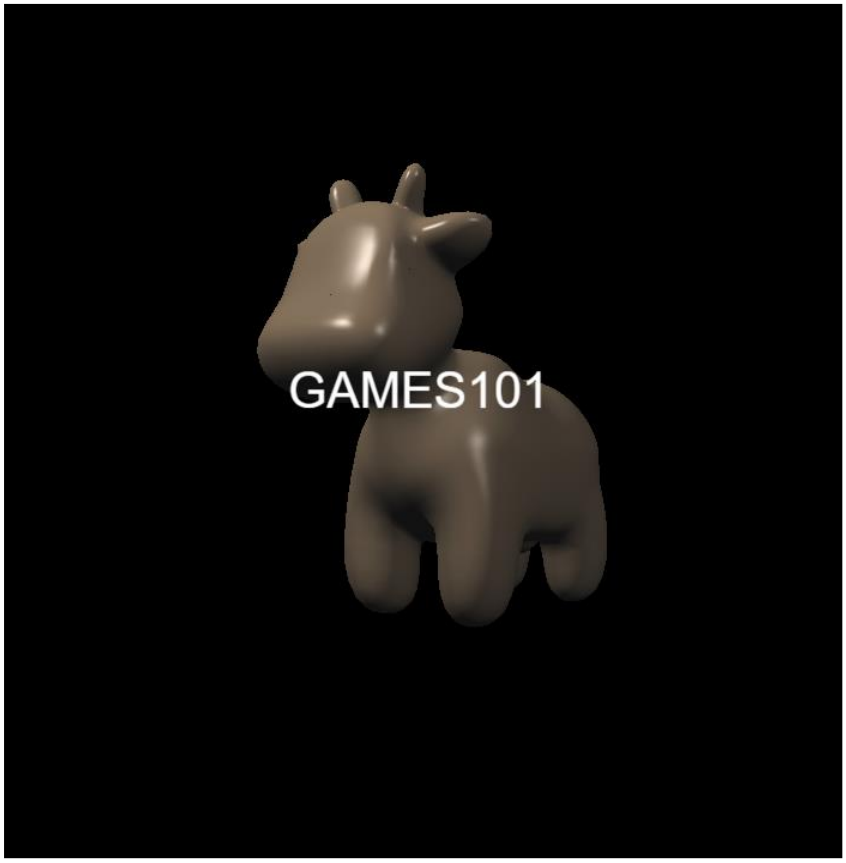
and shading colors. You should calculate the color and pass the color to framebuffer. To do that, you should use the result from interpolation to set fragment shader payload and use fragment shader to calculate the final color value.

4. Results:

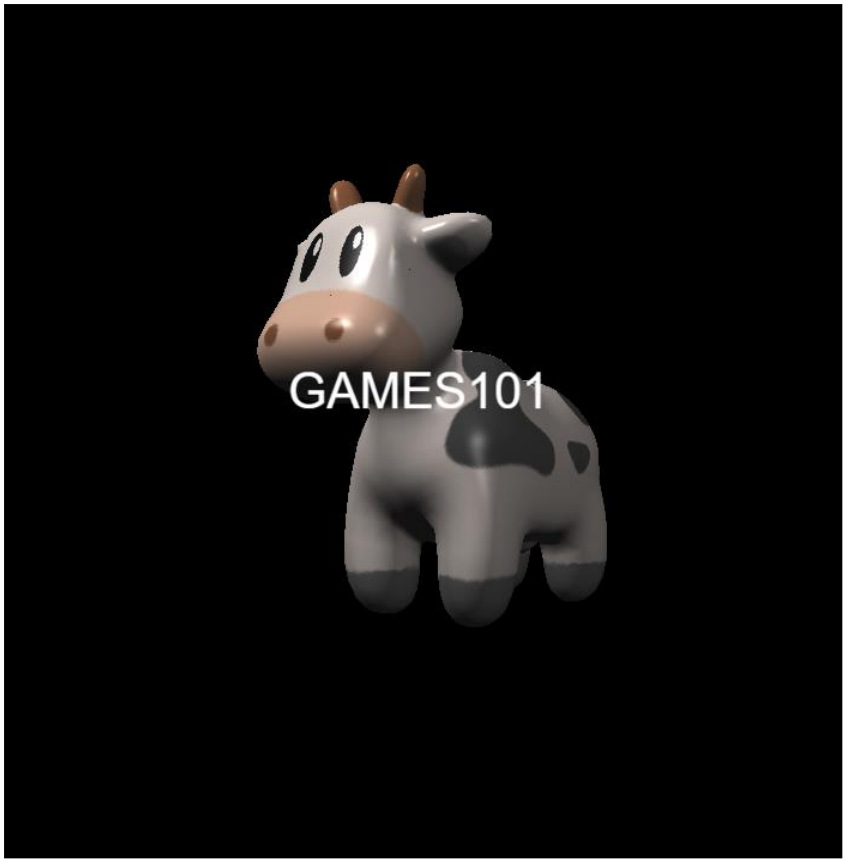
Normal shader result:



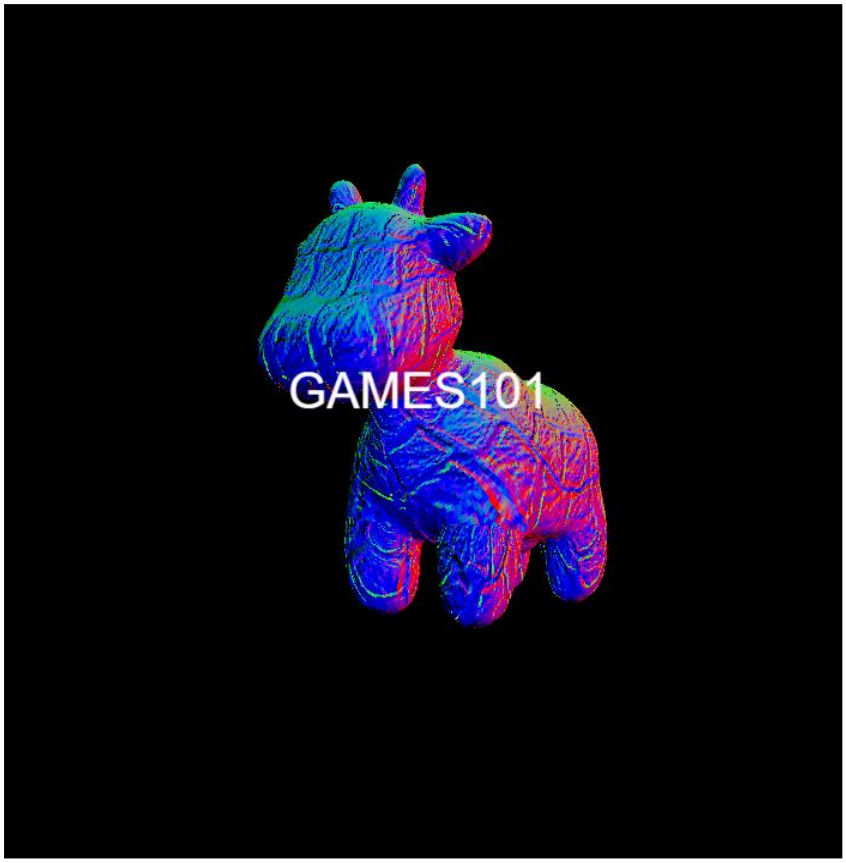
Blinn-Phong result:



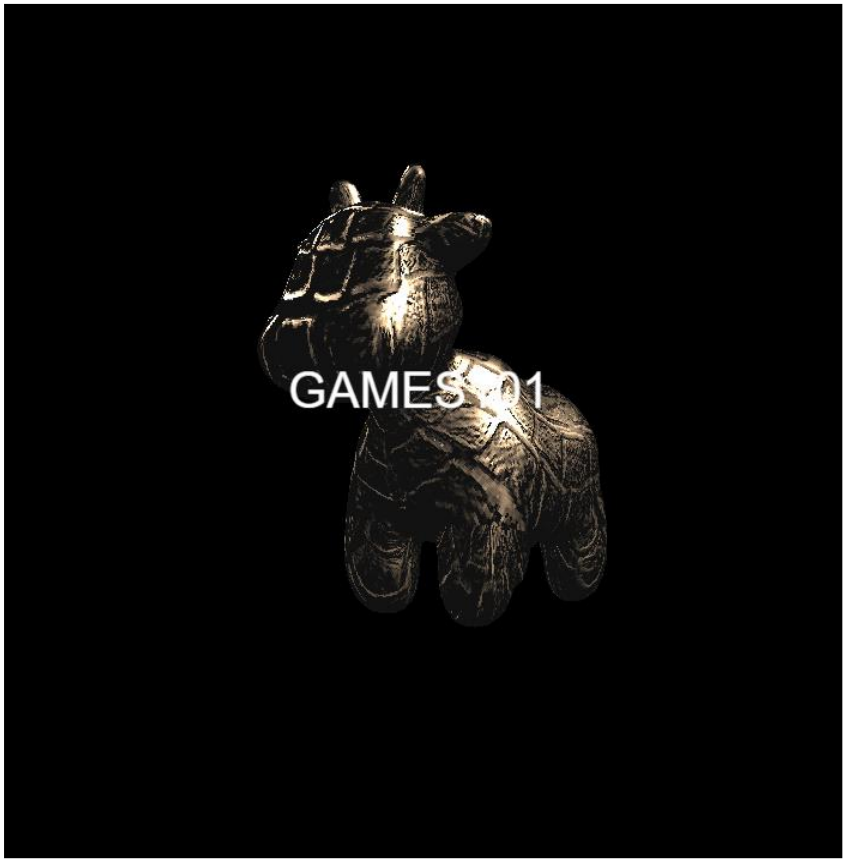
Texture mapping result:



Bump mapping result:



Displacement mapping result:



5. Grading

[30 points] submit all necessary files and the code can be compiled without any error.

[20 points] implement correct interpolation algorithm

[20 points] implement the Blinn-phong model

[20 points] implement the texture mapping

[10 points] implement the bump mapping and displacement mapping.

[extra credits (20 points)] try more obj files.