COSC 471: Computer Graphics

Spring 2023, Assignment 4

Pipeline and Shading

(Total points: 100)

Due date: May 14th, 2023

(end of the day)

Submission Instructions

1. Compile your solutions and generate screenshots of your output, then write your understanding of output in a file. Compress source codes output and your explanation into one compressed file.
2. Rename the compressed file following this notation:
   FirstnameLastnamePA4.zip (or any format you prefer)
   Do not use a space in the filename.
3. Upload and submit the compressed file through Blackboard.
4. **NO** late submission is accepted.

---

1. Overview

In this programming assignment we continue to further simulate modern graphics technology. We add **`Object Loader`** (used for load the 3D model), **`Vertex Loader`** and **`Fragment Loader`**, and support texture mapping too.

In this assignment, you need to complete the following tasks and modify the following functions:

- Modify function **`rasterize_triangle()`** in **`rasterizer.cpp`**: here you need to implement similar interpolation as last assignment 2, which interpolate normal, color and texture values.
- Modify function **`get_projection_matrix()`** in **`main.cpp`**: fill in the previous projection matrix implemented in the previous assignment here, and you can run **`./Rasterizer output.png normal`** to check the results of normal.
- Modify function **`phong_fragment_shader()`** in **`main.cpp`**: implement Blinn-Phong reflection model here to calculate Fragment Color.
- Modify function **`texture_fragment_shader()`** in **`main.cpp`**: based on the implementation of Blinn-Phong model, treat texture color as the ***kd*** in the formula to implement Texture Shading Fragment Shader.
- Modify function **`bump_fragment_shader()`** in **`main.cpp`**: based on the implementation of Blinn-Phong model, read the comments of this function carefully and implement Bump mapping.
- Modify function **`displacement_fragment_shader()`** in **`main.cpp`**: based on the Bump mapping, implement displacement mapping.

2. Compiling

2.1 Compiling and Run

After downloading the skeleton codes provided with this assignment, compile it as previous assignments:

```
Mkdir build
cd ./build
cmake ..
make
```

This will generate an executable file named **`Rasterizer`**. When run this executable file, you can pass the image name as first parameter, and for the second parameter you can pass like this:

- texture: this will run the code with **texture** shader.
  For example: **`./Rasterizer output.png texture`**
- normal: this will run the code with **normal** shader.
  For example: **`./Rasterizer output.png normal`**
- phong: this will run the code with **Blinn-Phong** shader.
  For example: **`./Rasterizer output.png phong`**
- bump: this will run the code with **bump** shader.
  For example: **`./Rasterizer output.png bump`**
- displacement: this will run the code with **displacement** shader.
  For example: **`./Rasterizer output.png displacement`**

Note: After you modify the codes, you need to re-compile using make to see the new results.

2.2 Skeleton codes

Compared with last assignment, we have made some modifications for the skeleton codes:

1. We have introduced a third-party .obj file loader to read in complicated models, and this library file is in OBJ_Loader.h file. You don't need to fully understand how it works, but just know it is a library which will pass us a Vector named TriangleList, in which each triangle has corresponding normal and texture coordinates. Besides, the texture related to the model will also be loaded. Note: if you try to load in other model, you have to change model directory and path manually.

2. We have introduced a new Texture class to generate textures from images, and provided an interface to check texture colors: **`Vector3f  getColor(float u, float v).`**

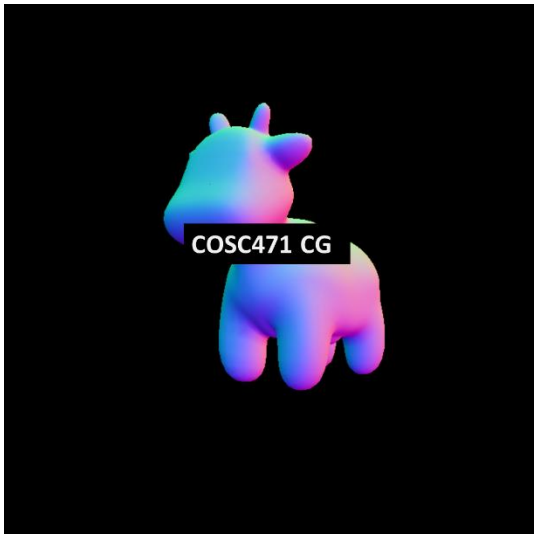
3. We have built a header file **`Shader.hpp`** and defined **`fragment_shader_payload`**, which includes parameters for Fragment Shader. Now main.cpp has three Fragment Shaders, which are fragment_shader (shader using normal) and two shaders you need to implement.

4. The main rendering pipeline starts with **`rasterizer::draw(std::vector<Triangle> &TriangleList)`**. We then again make some transformations, which is completed normally by Vertex Shader. After that, we then call function **`rasterize_triangle`**.

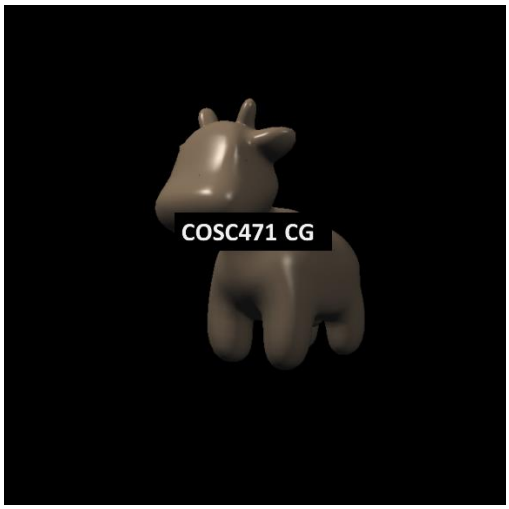
5. Function **`rasterize_triangle()`** is similar to the one in your assignment 2. The difference is that the set value is not longer a constant but set normal, color, texture and shading colors using interpolation based on Barycentric Coordinates. Recall last time we use [alpha, beta, gamma] to compute z value, this time you will use it to other interpolations. All you need to do is compute the color after interpolation, and write the color got from Fragment Shader to framebuffer. This requires you to first set the Fragment shader payload with the result of the interpolation, and call the fragment shader to get the results.

2.3 Run and Results

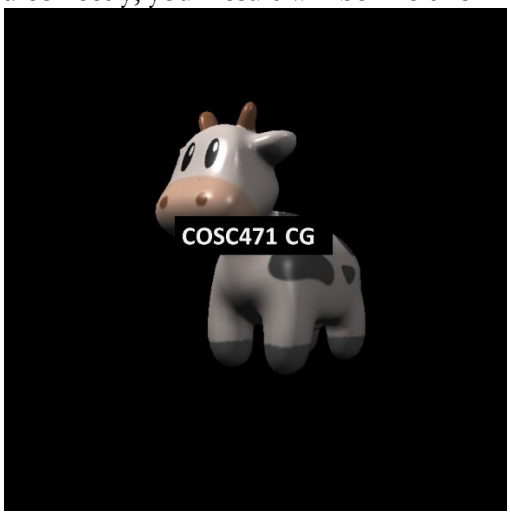
When you copy the code from last assignment as explained before and complete the modification required (please read the descriptions carefully), you should be able to run using the default normal shader and get the result like this:



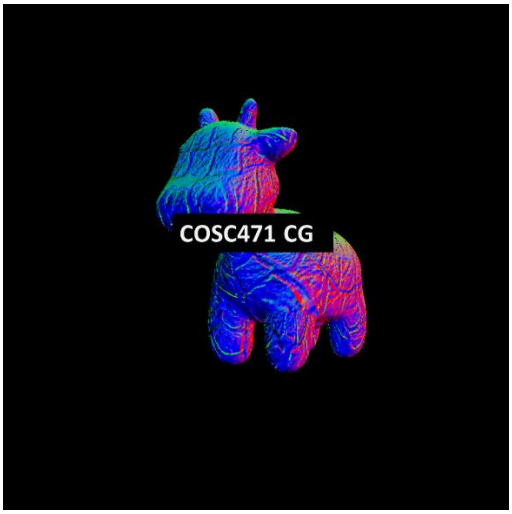
When you have implemented Blinn-Phone reflection model correctly, the result will be like this:



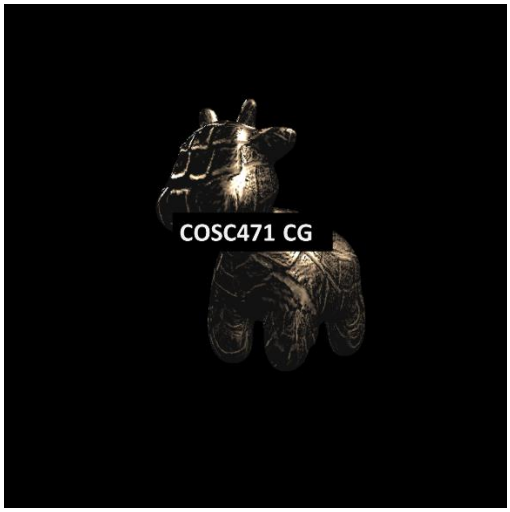
If the texture is implemented correctly, your result will be like this:



When Bump Mapping is implemented, you should be able to see visualized bump vector like this:



When Displacement Mapping is implemented correctly, you should be able to see output like this:



3. Assignment submission and evaluation

3.1 Submission

Read the assignment description carefully to make sure you understand the programing assignment correctly. Modify the skeleton codes as required. Compile your solutions and generate screenshots of your output, then write your understanding of output in a file. **Please create an image folder and save all your images in this folder**. Compress source codes output, images and your explanation into one compressed file named **FirstnameLastnamePA3.zip (or any format you prefer).** Submit the compressed file through Blackboard before deadline.

When you complete the assignment, please check your project clearly and make sure your submission folder includes **CMakeLists.txt file** and all the source files no matter you changed them or not. In addition, please include a **README.md** file to explain if you complete the bonus question (if you complete bonus question, please include screenshot of it too), or explain this in the result analysis and screenshot file.

3.2 Evaluation

This programming assignment has two parts: basic parts and bonus part. Here is the grading details:

- Correctly implement interpolations for color, normal, texture and shading position and pass them to fragment_shader_payload (20%)

- Correctly implement Blinn-Phong reflection model (20%)
- Correctly implement Texture Mapping (20%)
- Correctly implement Bump Mapping and Displacement Mapping (20%)
- Compile and run codes correctly and submit all files correctly (10%)
- Screens shot and result analysis (10%)
- **Bonus part**: try other models, you can find them in the models folder. (10%)