CAP4730 Viewing Transformations and Shading Report

Jonathan Bryan

April 2024

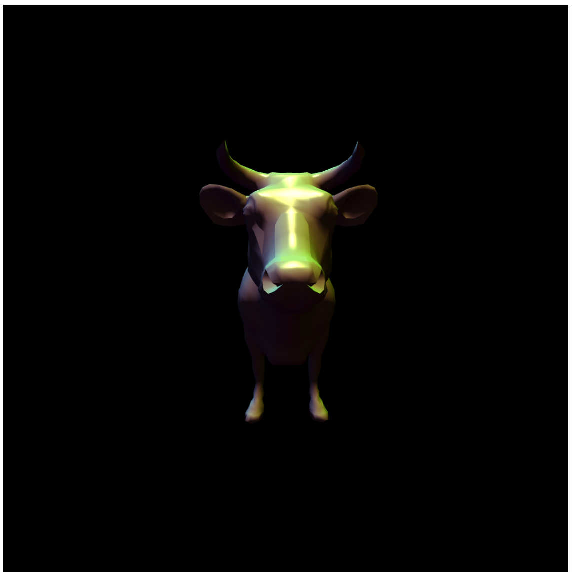
1 Introduction

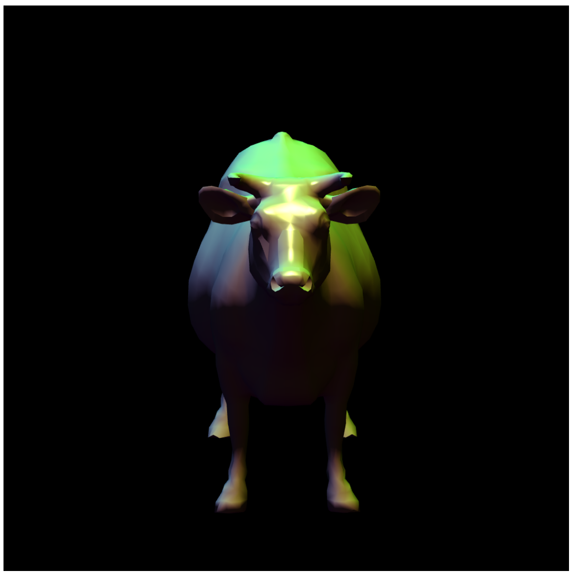
This project covers view and projection within the graphics pipeline along with shading. View and projection matrices are combined with models to give us a scene that changes with perspective parameters. Flat, gouraud, and phong shading styles give our objects color and texture using normal data.

2 Projection

2.1 Perspective

The main parameters used by perspective projection are field of view, near plane, and far plane. For this project the perspective matrix is generated using *glm::perspective*. Changing our field of view creates a fisheye effect where the edges become distorted. The near plane controls how close to our screen objects are rendered. We generally wan this to be a small value so that our scene is not clipped. The far plane is how far back objects will be rendered.


FOV: 100


FOV: 10

2.2 View

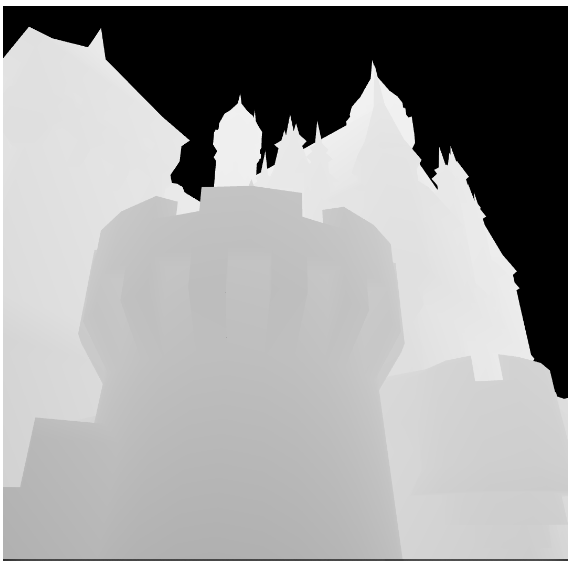
Our view matrix is generated by our camera. It uses *glm::lookAt* to generate a matrix which transforms world space to view space. This function requires eye, center, and up vector. Our eye vector is simply the position of the camera. The center of the camera is the position added with the front facing vector. The up vector is tracked by the camera and changes with the pitch of our camera.

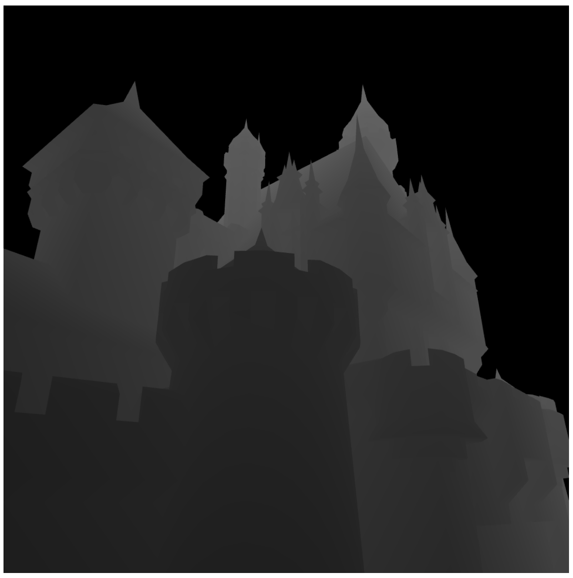
Cow from the front

Cow from under

3 Z-Buffer

The z-buffer is what gives our scene depth. Each vertex has a z value which is important for controlling the rending order. As we move our camera around the depth value needs to change accordingly so that closer objects are always on top of further objects. We can visualize our depth buffer through color to see what depth value objects have. When we directly draw *gl_FragCoord.z* to the screen all of our objects appear white. This happens because depth values are larger than 1 and therefore become white. We can use the values we have for our near and far planes to scale our depth back so that it is in a range of 0 to 1. The equation for that is as follows: $z' = \frac{2n}{f+n-z(f-n)}$. This equation appears in a shader designated for just the depth mode.

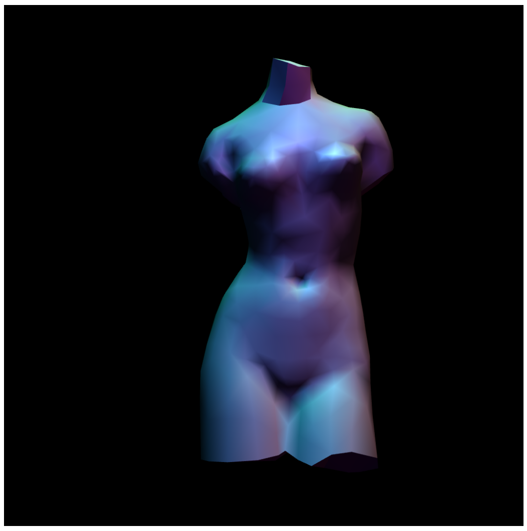

Depth straight from position (z)

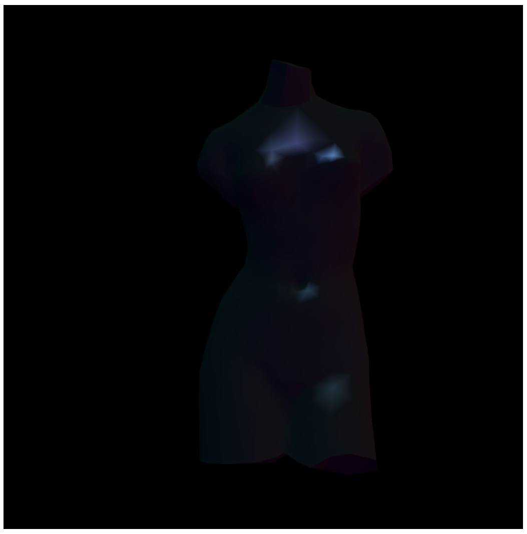

Scaled depth equation (z')

4 Shading

4.1 Gouraud

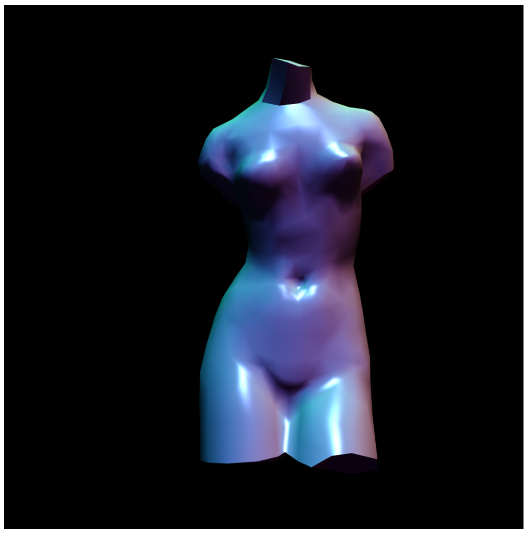
The gouraud shading method involves calculating ambient, diffuse, and specular light per vertex.

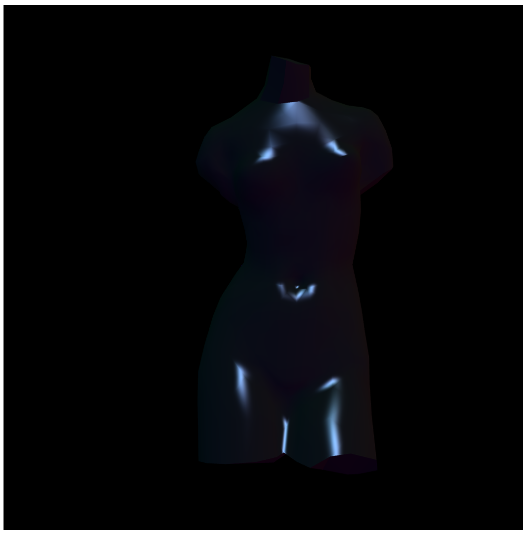

Gouraud shading


Gouraud shading specular component

4.2 Phong

We can get a lot more detail by instead calculating our shading per fragment instead of per vertex. Our normal and position are interpolated in the fragment shader and can be used to do the exact same calculations as the gouraud shader. This method utilizes more resources but has much sharper lighting.


Phong shading


Phong shading specular component

4.3 Flat

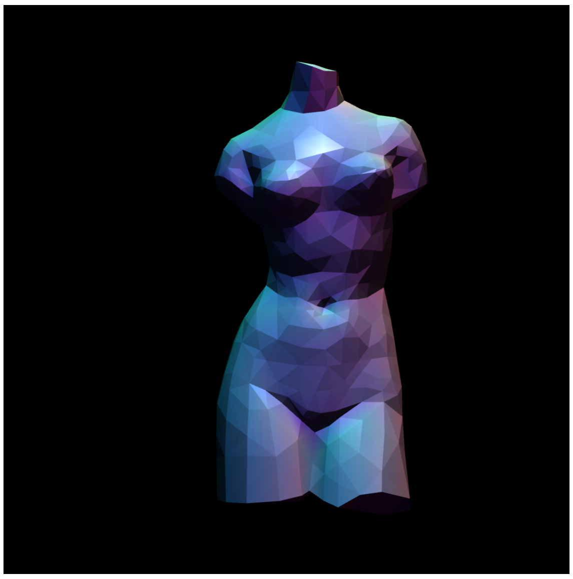
Flat shading uses a single normal per triangle for it's shading calculations. In order to get this normal we must calculate it using the vertex data from our triangle. The equation for the normal is as follows:

$$u = p2 - p1$$

$$v = p3 - p1$$

$$n = u \times v$$

Once we have this value we simply follow gouraud shading. The output of this shading style clearly shows each of the triangles and it's interaction with the light source.



Phong shading

5 Video

A sample video can be found here. It shows all of the different shading styles and depth buffer.

6 References

6.1 ImGui

ImGui (github.com/ocornut/imgui) was utilized in this project for the menu. The interactive manual (pthom.github.io/imgui_manual_online) was helpful when implementing the matrix table representation.

6.2 LearnOpenGL

I read through parts of the LearnOpenGL (learnopengl.com/) tutorials for this project. Code written to store and load shader files was inspired by the ”Shaders” tutorial (learnopengl.com/Getting-started/Shaders). The ”Camera” section (learnopengl.com/Getting-started/Camera) helped me in implementing the camera. The algorithm I ended up using for scaling depth testing came from the ”Depth testing” tutorial (learnopengl.com/Advanced-OpenGL/Depth-testing).