report_weatherly

April 4, 2023

1 Report - HW4

1.0.1 Chad Weatherly

Problem

For homework 4, the problem was to implement shading into the Gz library.

Method

For my method, I attempted to do the coloring/light intensity aspect within the Gz library. To do this, I updated the `end` method to take in shading, and if shading is turned on, the colors and intensity are calculated based on the lights given to the model and their positions.

I am still a bit confused exactly how the transformations and projections work, but from my understanding, it seems that the primitives are transformed (following the conventions of glLookAt) into a new space, where the viewer is positioned at the origin (0, 0, 0) and the center of the objects is on the negative z-axis. Therefore, in order to calculate colors, I thought that all vectors/vertices should be transformed into this new space, and then colors can be calculated based on the transformed normal vectors, light vectors, and vertices of the triangle. After this, projections can happen and the projected vertices with their colors could be plotted. I created additional methods in GzMatrix to take in Vectors for multiplication as well.

Given the equation,

$L = k_a I_A + k_d(I/r^2)max(0, n \cdot r) + k_s(I/r^2)max(0, n \cdot h)^p$

It made sense in theory, but in practice I was unsure of the implementation, as technically, the lights are infinitely far away, which would make $r^2 \Rightarrow \infty$ and both the 2nd and 3rd terms 0? But then we are only left with the ambient light term, which doesn't make sense. So, I made the term, $(I/r^2) = 1$ to preserve those terms.

Finally, all created images were placed in the /output directory.

Implementation

This class and all of its methods were coded in C++, with the initial declaration of the class and its methods/attributes in the Gz.h and GzFrameBuffer.h files, then implemented in Gz.cpp and GzFrameBffer.cpp.

After the coding was done, compiling was done through the terminal, as I am coding on MacOS with an M2 max chip on VS code. Currently, VS code struggles to compile and link files, but this was circumvented using the terminal. From the HW4 directory, the below code was used to compile the program to an executable object, hw4:

```
clang++ src/main.cpp src/Gz.cpp src/GzFrameBuffer.cpp src/GzImage.cpp src/GzMatrix.cpp src/GzV
```

For using G++:

```
g++ src/main.cpp src/Gz.cpp src/GzFrameBuffer.cpp src/GzImage.cpp src/GzMatrix.cpp src/GzVecto
```

As I am new to C++, I was not able to create a makefile to compile, but the above code should work fine on any machine. Also, the hw3.exe file is included in this directory, and can also be run using ./hw4 after compiling.

Results

Like I stated above, I'm still not totally clear how the transformations and projections work, and the powerpoint slides are not extremely good at explaining in detail what is going on. So, as you can see in the /output folder, I was able to show the teapots, but with the incorrect colors and rotated about 180 degrees. I am unsure why they are rotated, as the vertex points are transformed directly with the `transAll()` method included in the HW4 downloadable file.

[ ]: