report_weatherly

February 18, 2023

1 Report - HW1

1.0.1 Chad Weatherly

Problem

For homework 1, the problem was to create the GzFrameBuffer class in C++, used by the larger Gz class which contains all the code to draw a 2D image using a Z-buffer so that the closest objects are drawn, asuming they have no transparency.

Method

For my method, I created the GzFrameBuffer class with the following methods:

- `initFramSize()`, which takes in width and height values and initializes the buffer screen with these dimensions.
- `toImage`, which outputs the buffer as a `GzImage` object.
- `clear`, which clears the buffer screen at z-value `clear_depth` or lower with the `clear_color`.
- `setClearColor`, which sets the `clear_color` attribute with an input `GzColor` object.
- `setClearDepth`, which sets the `clear_depth` attribute with an input `GzReal` (float) object.
- `get_depth`, which gets the depth of the buffer image at entry (x, y).
- `set_depth`, which sets the depth of the buffer image at entry (x, y) with value z.
- `drawPoint`, the essential function of GzFramBuffer. Once the frame buffer is initialized, individual pixels are updates with `drawPoint` at pixel (x, y) with a specified color and given a depth-value at that pixel, if the depth (z-value) of the new vertex is greater than the current pixel depth at (x, y). If not, then the pixel is left unchanged.

Along with the methods, the following private attributes were also added:

- `GzInt w`, the width of the image frame.
- `GzInt h`, the height of the image frame.
- `GzColor clear_color`, the RGBA color value used to clear the buffer below the z-value `clear_depth`.
- `GZInt clear_depth`, the z-value used to clear the frame buffer
- `GzImage internal_buffer`, the image buffer that is constantly updated with the methods above.
- `vector<vector<GzInt> > pixel_depth`, a 2D vector that keeps track of the depth at each pixel location.

The GzFrameBuffer class essentially has two main types of actions, initialize/clear and drawing a point.

When the buffer is initialized, it automatically creates and clears the image frame. After initialization, the image frame can be cleared again with different colors or to different depths.

Drawing a point is the essential part of the class, as it is actually doing the work of drawing given colors on the given vertices by the Gz class. This image is constantly updated as Gz passed new vertices and colors to GzFrameBuffer.

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 GzFrameBuffer.h file, then implemented in GzFrameBuffer.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 HW1 directory, the below code was used to compile the program to an executable object, hw1.exe:

```
clang++ src/main.cpp src/Gz.cpp src/GzImage.cpp src/GzFrameBuffer.cpp -o hw1
```

For using G++:

```
g++ src/main.cpp src/Gz.cpp src/GzImage.cpp src/GzFrameBuffer.cpp -o hw1
```

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 hw1.exe file is included in this directory.

Results

The algorithm was able to successfully create bitmap images (also in this directory) with different buffer depths and buffer colors, which can be seen below: