COSC6372 HW1 – Frame Buffer

Jiechang Guo UHID: 2084258

February 8, 2023

1 Problem

The problem for this assignment is to use the z-buffer to draw several rectangles to an image with corner coordinates, colors, and depth information from a file. The rendering for the rectangles will follow the rule that the pixel with a larger z value will be drawn on the top of the image. If two pixels share the same z value, the pixel drawn later will be on top.

2 Method

In order to render the rectangles with a z-buffer and save the result to an image, the frame buffer module with depth buffer in the Gz library needed to be implemented. A GzFrameBuffer class contains two buffers including a color buffer to store rendered pixel color and a depth buffer to store depth value in order to do depth test. Also, clear color and default depth set by the user are stored too, to set the color of the background and the default depth value to do the depth test. The status of the clear specific buffer is stored in the interface class Gz.
A simple rasterization process was done by the fillRec function, thus the only thing the framebuffer needs to handle is to determine the color of each pixel according to the z-buffer.
The process is done in the draw function. First, a clipping process is done to make sure the pixel coordinate is inside the image, then if the depth test is enabled, check the z value of the pixel, if the value is equal to or greater than the value in the z-buffer then update the color buffer to the color of the point and rewrite the z-buffer to current z value, otherwise, do nothing.

3 Implement

The code is implemented using C++ on MacOS. The following section will go through the implementation in detail. The declaration of the class is as followed.

```
//Frame buffer with Z-buffer ------------------------------------------------
class GzFrameBuffer {
public:
        //The common interface
        void initFrameSize(GzInt width, GzInt height);
        GzImage toImage();

        void clear(GzFunctional buffer);
        void setClearColor(const GzColor& color);
        void setClearDepth(GzReal depth);

        void drawPoint(const GzVertex& v, const GzColor& c, GzFunctional status);

private:
        //pixel values for the final image
        vector<vector<GzColor> > colorBuffer;
        //depth value to do depth test
        vector<vector<GzReal> > depthBuffer;
        //buffer size, image size
        GzInt width_,height_;
        //preset clear color
        GzColor clearColor_;
        //preset depth
        GzReal clearDepth_;
};
```

3.1 Initialization

During the initialization step, the user will call the initFrameSize function, in the implementation of the GzFrameBuffer class, the color buffer and depth buffer will be initialized according to the set width and height. The code

is as followed,

```
void GzFrameBuffer::initFrameSize(GzInt width, GzInt height)
{
    width_ = width;
    height_ = height;
    colorBuffer.resize(width,vector<GzColor>(height));
    //cout<< colorBuffer.size() << endl;
    depthBuffer.resize(width,vector<GzReal>(height));
}
```

3.2 Setting up rendering options

Before rendering the primitive, the user will set up a couple of rendering options for the Gz object, including setting the clear color which is the background color and setting the default depth value to clip everything behind that depth. Whenever the user calls gz.clear(GZ_COLOR_BUFFER — GZ_DEPTH_BUFFER), GzFrameBuffer object will clear the buffer with a preset value according to the user's command.

```
void GzFrameBuffer::clear(GzFunctional buffer)
{
    //clear color buffer with set clear color
    if(buffer & GZ_COLOR_BUFFER)
    {
        for(GzInt i = 0; i < width_; i++)
        {
            for(GzInt j = 0; j < height_; j++)
            {
                colorBuffer[i][j] = clearColor_;
            }
        }
    }

    //clear depth buffer with set default depth
    if(buffer & GZ_DEPTH_BUFFER)
    {
        for(GzInt i = 0; i < width_; i++)
        {
            for(GzInt j = 0; j < height_; j++)
            {
                depthBuffer[i][j] = clearDepth_;
            }
        }
    }
}

void GzFrameBuffer::setClearColor(const GzColor &color)
{
    clearColor_ = color;
}

void GzFrameBuffer::setClearDepth(GzReal depth)
{
    clearDepth_ = depth;
}
```

3.3 Rendering primitives

As mentioned before, the rasterization process is done in the fillRec function, which pushes all the pixel points in a queue and set the primitive type as GZ_POINTS. After calling gz.end(), every pixel points in the queue will be popped out and rendered using the drawPoint function in the GzFrameBuffer, which followed the method we mentioned before.

```
void GzFrameBuffer::drawPoint(const GzVertex &v, const GzColor &c, GzFunctional status)
{
    //clip the points outside the image
    if(v[0] > width_ || v[1] > height_ || v[0] < 0 || v[1] < 0)
    {
        return;
    }
    //assign pixel value here
    if(status & GZ_DEPTH_TEST)
    {
        //depth test
        //draw the point with larger or equal z value
        if(v[2] >= depthBuffer[v[0]][v[1]])
        {
            colorBuffer[v[0]][v[1]] = c;
            //write the depth buffer
            depthBuffer[v[0]][v[1]] = v[2];
        }
    }
    else
    {
        //without depth test
        colorBuffer[v[0]][v[1]] = c;
    }
}
```

3.4 Ouput rendering result

The result of rendering will be saved in the BitMap image, all we need to do in the GzFrameBuffer object is create a GzImage object and set the pixel value from colorBuffer to the image. Then, during the main function, the BitMap can be saved to the disk.

```
GzImage GzFrameBuffer::toImage()
{
    GzImage image(width_,height_);
    for(GzInt i = 0; i < width_; i++)
    {
        for(GzInt j = 0; j < height_; j++)
        {
            image.set(i,j,colorBuffer[i][j]);
        }
    }
    return image;
}
```

4 Result

Before rendering the rectangles without depth test, the clear color was set to black. Thus, the background in 1 is black. Also, the gz.disable(GZ_DEPTH_TEST) is called to disable the depth test, thus, the rectangles will be drawn according to the order in the file, the rectangle drawn latter will be on the top of the rectangle.

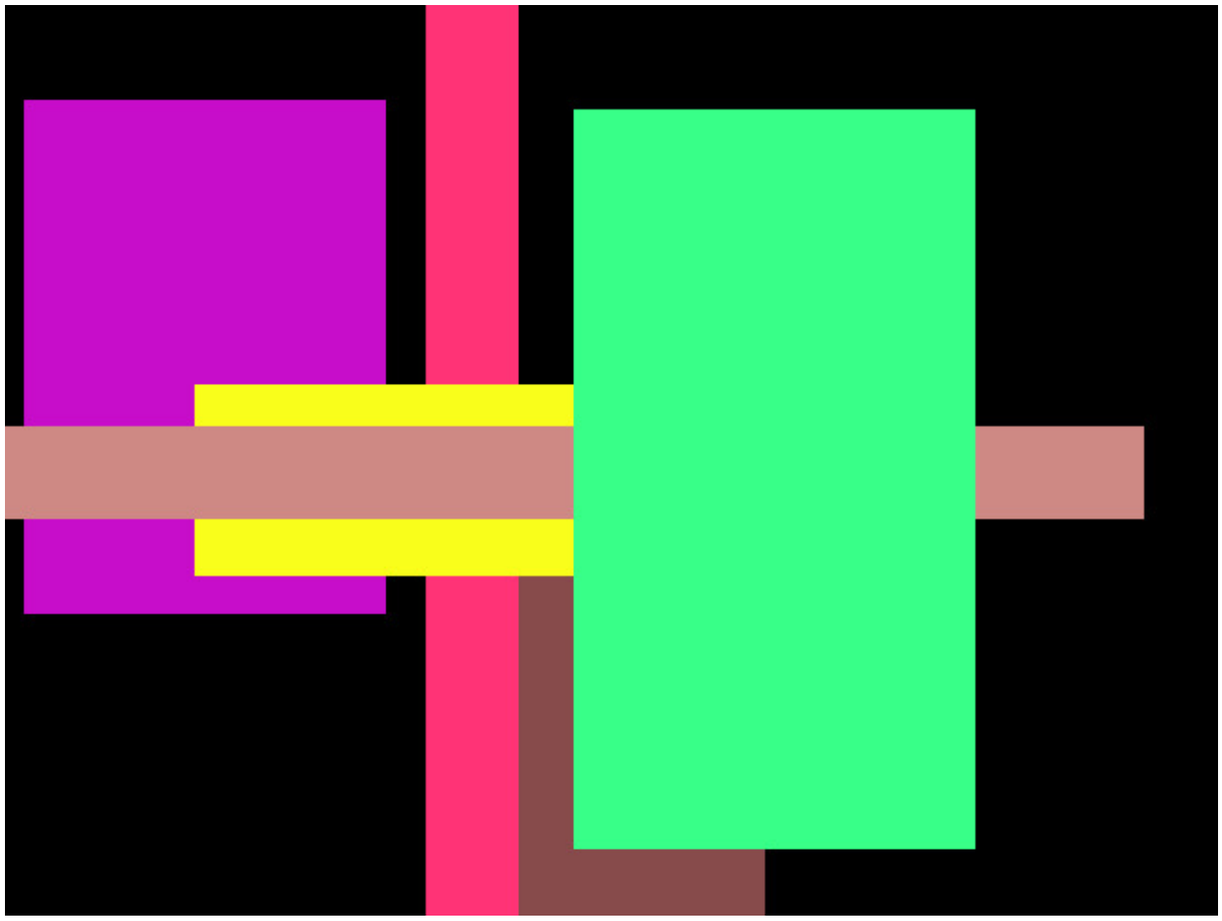

Figure 1: No Depth Test

To draw the rectangles with depth test, the clear color was set to white and the GZ_DEPTH_TEST is enabled, thus, the rectangles will be drawn according to their depth value. The most obvious difference between the result with and without depth test is the brown rectangle on the bottom of the image is disappeared after depth test, this is because the rectangle has the z value equal to -23, is less than the default depth value.

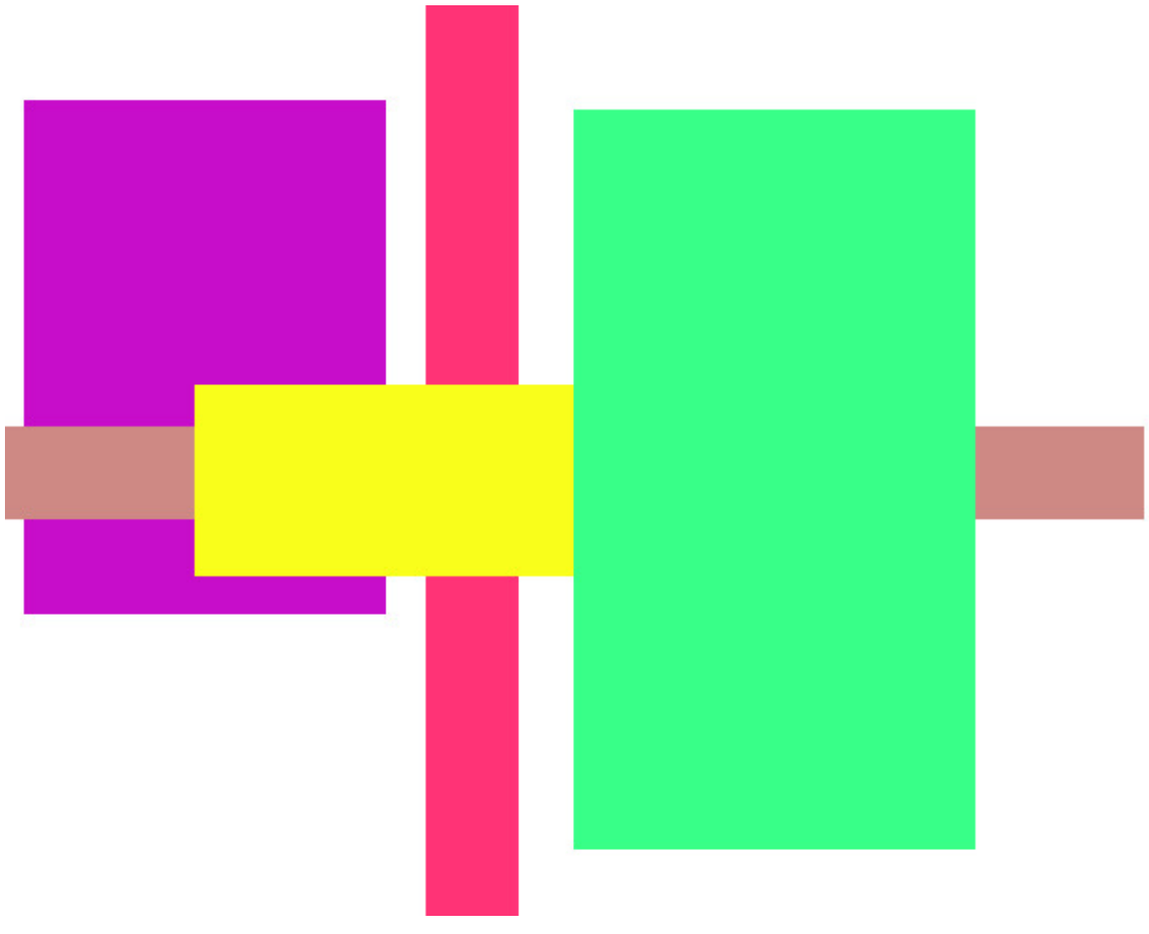

Figure 2: With Depth Test