CS6060- HW2

Max Thrun

Fall 2013

For this homework the first thing I did was write a Python script that allows you to trace an image and generate a list of coordinates.

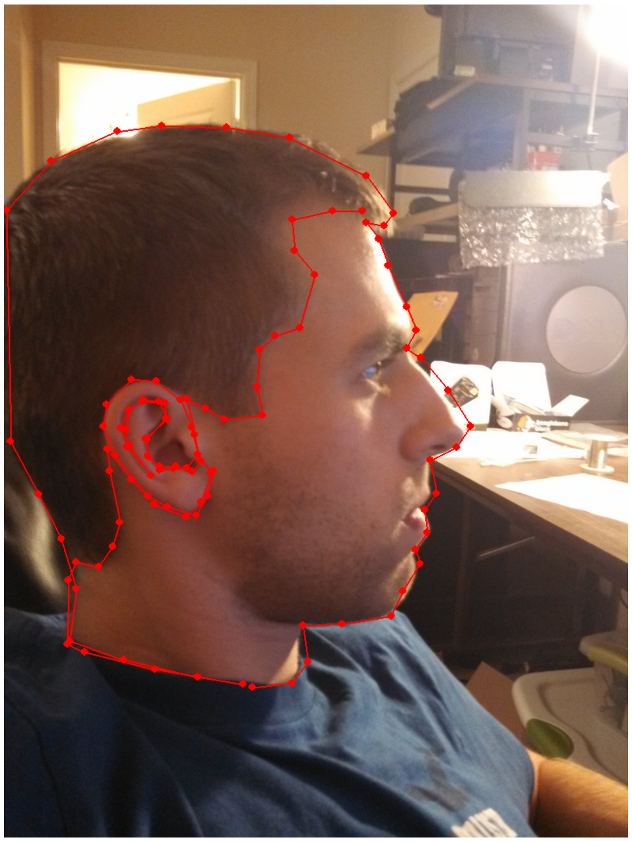
Figure 1: Python Tracing Program

The output of the script gives you a `data.h` file with an array that can be loaded directly into a vertex buffer. This method allows you to quickly gather a list of points and for my program I used about 200.

```
static const GLfloat lines[] = {
    -3.250000f,3.435000f,0.0f,
    -2.940000f,3.575000f,0.0f,
    -2.640000f,3.625000f,0.0f,
    -2.340000f,3.695000f,0.0f,
    -2.020000f,3.755000f,0.0f,
    ...
```

Listing 1: Output of Python tracing program

The window and OpenGL contex are created using SFML instead of GLUT. The reason for this is that SFML is cross-platform and significantly more up to date than GLUT. It offers a variety of methods that make setup extremely easy. To create a window and an OpenGL context all that is needed is:

```
int main()
{

    sf::ContextSettings contextSettings;
    contextSettings.depthBits = 32;

    sf::RenderWindow window(sf::VideoMode(800, 600), "Homework 2", sf::Style::D
    window.setVerticalSyncEnabled(true);

    window.setActive();

    ...
```

Listing 2: Window Creation

Next I create a vertex buffer object and load the output of my tracer program into the buffer.

```
GLuint vbo;
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(lines), lines, GL_STATIC_DRAW);
```

Listing 3: Loading VBO

I then load the vertex and fragment shaders and get the ID for the model-view-projection matrix uniform as well as the vertex position variable. Once I have the vertex position ID I enable it and then setup the position buffer attributes. In this case we simply have 3 floats per attribute and they are tightly packed.

```
GLuint vbo;
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(lines), lines, GL_STATIC_DRAW);

GLuint shader_id = LoadShaders("vert.glsl", "frag.glsl");
GLint mvp_id = glGetUniformLocation(shader_id, "mvp");
GLint posAttrib = glGetAttribLocation(shader_id, "position");
glEnableVertexAttribArray(posAttrib);
glVertexAttribPointer(posAttrib, 3, GL_FLOAT, GL_FALSE, 0, 0);
```

Listing 4: Setting up shaders

Next, using GLM, I create three matrices for the projection, view, and model. The projection matrix is setup for a 45 degree FOV and a 4/3 aspect ratio. The near and far clip are set to 0.1 and 100, respectively. I set the view matrix to position that camera 10 units back from origin on the Z axis. The target is the origin and positive Y is our up vector. Since the MVP matrix wont change per frame I calculate it here once, before the main render loop.

```
glm::mat4 projection = glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.0f);

glm::mat4 view = glm::lookAt(
        glm::vec3(0,0,10),     // position
        glm::vec3(0,0,0),      // target
        glm::vec3(0,1,0)       // up
        );

glm::mat4 model = glm::mat4(1.0f);

glm::mat4 mvp = projection * view * model;
```

Listing 5: Matrix Setup

The main render loop is simple. I first poll for events using some APIs provided by SFML. The only ones I am handling is the window close and escape key in order to exit the program. I then tell openGL to use our shader program that we loaded earlier and I send it the MVP matrix. I then simply draw all the points in the vertex buffer as a line strip. Finally the display is flipped with the SFML function `window.display()`.

```
while ( window.isOpen() ) {
    sf::Event event;
    while (window.pollEvent(event)) {
        if (event.type == sf::Event::Closed)
            window.close();
        if (event.type == sf::Event::KeyPressed) {
            switch(event.key.code) {
                case sf::Keyboard::Escape:
                    window.close();
                    break;
            }
        }
    }

    glUseProgram(shader_id);
    glUniformMatrix4fv(mvp_id, 1, GL_FALSE, glm::value_ptr(mvp));
    glDrawArrays(GL_LINE_STRIP, 0, sizeof(lines)/sizeof(float)/3);
    window.display();
}
```

Listing 6: Render Loop

The vertex and fragment shaders are extremely simple. The vertex shader simply transforms the model space coordinates that are in our vertex buffer to screen coordinates by multiplying them by the model view projection matrix. The fragment shader simply sets all vertices, and all the points interpolated between them, to red.

```
#version 330 core

in vec3 position;
uniform mat4 mvp;

void main(){
        gl_Position = mvp * vec4(position,1);
}
```

Listing 7: Vertex Shader

```
#version 330 core

out vec3 color;

void main()
{
        color = vec3(1,0,0);
}
```

Listing 8: Fragment Shader

With that we achieve the final program output.

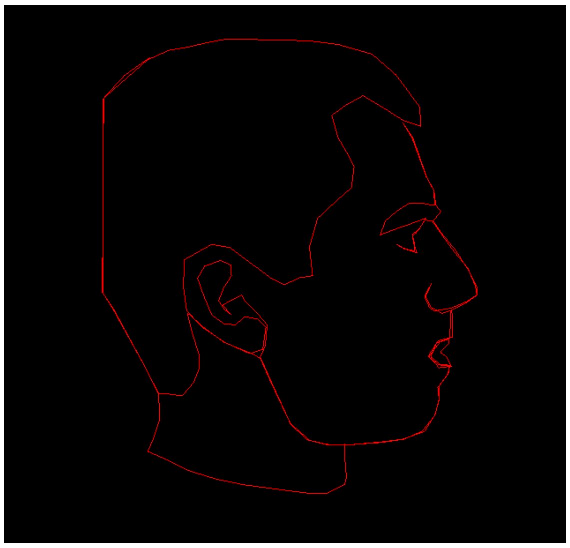

Figure 2: Final output

Listing 9: Python Tracing Program

```
import sys, pygame
pygame.init()

SCALE = 100.0

image = pygame.image.load(sys.argv[1])
screen = pygame.display.set_mode(image.get_size())

points = []

while 1:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
            print pygame.mouse.get_pos()
            points.append(pygame.mouse.get_pos())
        if event.type == pygame.KEYDOWN and event.key == pygame.K_u:
            if len(points) > 0:
                points.pop()
        if event.type == pygame.KEYDOWN and event.key == pygame.K_s:
            with open("data.h", "w") as f:
                f.write("static const GLfloat lines[] = {\n")
                for x,y in points:
                    y = image.get_size()[1] - y # flip the y
                    x -= image.get_size()[0]/2.0
                    y -= image.get_size()[1]/2.0
                    x /= SCALE
                    y /= SCALE
                    f.write("%ff,%ff,0.0f,\n" % (x,y));
                f.write("};\n");
            print "Wrote data.h with %d points" % len(points)

    screen.blit(image, image.get_rect())

    for p in points:
        pygame.draw.circle(screen, (255,0,0), p, 5)

    if len(points) > 1:
        pygame.draw.lines(screen, (255,0,0), False, points, 2)

    pygame.display.flip()
```

Listing 10: Full program

```
#define GLEW_STATIC
#include <GL/glew.h>
#include <SFML/Graphics.hpp>
#include <SFML/OpenGL.hpp>
#include <iostream>
#include <ctime>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include "shader.h"
#include "data.h"

int main()
{

    sf::ContextSettings contextSettings;
    contextSettings.depthBits = 32;

    sf::RenderWindow window(sf::VideoMode(800, 600), "Homework 2", sf::Style::D
    window.setVerticalSyncEnabled(true);

    window.setActive();

    glewExperimental = GL_TRUE;
    glewInit();

    GLuint vbo;
    glGenBuffers(1, &vbo);
        glBindBuffer(GL_ARRAY_BUFFER, vbo);
        glBufferData(GL_ARRAY_BUFFER, sizeof(lines), lines, GL_STATIC_DRAW);

    GLuint shader_id = LoadShaders("vert.glsl", "frag.glsl");
    GLint mvp_id = glGetUniformLocation(shader_id, "mvp");
    GLint posAttrib = glGetAttribLocation(shader_id, "position");
    glEnableVertexAttribArray(posAttrib);
    glVertexAttribPointer(posAttrib, 3, GL_FLOAT, GL_FALSE, 0, 0);

        glm::mat4 projection = glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.0f

    glm::mat4 view = glm::lookAt(
            glm::vec3(0,0,10),      // position
            glm::vec3(0,0,0),        // target
            glm::vec3(0,1,0)         // up
            );

        glm::mat4 model = glm::mat4(1.0f);

        glm::mat4 mvp = projection * view * model;

    while ( window.isOpen() )
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            // Close window : exit
            if (event.type == sf::Event::Closed)
                window.close();

            if (event.type == sf::Event::KeyPressed) {
                switch(event.key.code) {
                    case sf::Keyboard::Escape:
                        window.close();
                        break;
                }
            }
        }

                glUseProgram(shader_id);
        glUniformMatrix4fv(mvp_id, 1, GL_FALSE, glm::value_ptr(mvp));

        glDrawArrays(GL_LINE_STRIP, 0, sizeof(lines)/sizeof(float)/3);

        window.display();
    }

    return 0;
}
```