CS3388B, Winter 2023

Problem Set 6

Due: March 3, 2023

Exercise 1.

When rendering translucent (semi-transparent) objects, why must you render objects further away from the camera before objects closer to the camera? Consider the following two code segments to help explain why.

```
glMatrixMode(GL_PROJECTION);
glm::mat4 P = glm::perspective(glm::radians(45.0f), screenW/screenH, 0.001f, 1000.0f);
glLoadMatrixf(glm::value_ptr(P));

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glBegin(GL_TRIANGLES);
glColor4f(1.0f, 0.0f, 0.0f, 0.5f);
glVertex3f(0.0f, 1.0f, -3.0f);
glVertex3f(-1.0f, 1.0f, -3.0f);
glVertex3f(-1.0f, -1.0f, -3.0f);
glVertex3f(0.0f, -1.0f, -3.0f);
glVertex3f(0.0f, 1.0f, -3.0f);
glVertex3f(-1.0f, -1.0f, -3.0f);

glColor4f(0.0f, 1.0f, 0.0f, 0.3f);
glVertex3f(-0.5f, 0.0f, -5.0f);
glVertex3f(-0.5f, -1.0f, -5.0f);
glVertex3f(1.f, -1.0f, -5.0f);
glVertex3f(1.f, 0.0f, -5.0f);
glVertex3f(-0.5f, 0.0f, -5.0f);
glVertex3f(1.f, -1.0f, -5.0f);
glEnd();
```

```
glMatrixMode(GL_PROJECTION);
glm::mat4 P = glm::perspective(glm::radians(45.0f), screenW/screenH, 0.001f, 1000.0f);
glLoadMatrixf(glm::value_ptr(P));

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glBegin(GL_TRIANGLES);
glColor4f(0.0f, 1.0f, 0.0f, 0.3f);
glVertex3f(-0.5f, 0.0f, -5.0f);
glVertex3f(-0.5f, -1.0f, -5.0f);
glVertex3f(1.f, -1.0f, -5.0f);
glVertex3f(1.f, 0.0f, -5.0f);
glVertex3f(-0.5f, 0.0f, -5.0f);
glVertex3f(1.f, -1.0f, -5.0f);

glColor4f(1.0f, 0.0f, 0.0f, 0.5f);
glVertex3f(0.0f, 1.0f, -3.0f);
glVertex3f(-1.0f, 1.0f, -3.0f);
glVertex3f(-1.0f, -1.0f, -3.0f);
glVertex3f(0.0f, -1.0f, -3.0f);
glVertex3f(0.0f, 1.0f, -3.0f);
glVertex3f(-1.0f, -1.0f, -3.0f);
glEnd();
```

We must draw primitives further away first or else blending will not work correctly. We need a background color to blend against. Therefore, we need the objects further away to be rendered first and then blend the nearer objects against the ones behind it.

Exercise 2.

Why is the default value for `glClearDepth` equal to 1? (This is not directly in the notes; use some crticial thinking about the role of the depth buffer and where a fragment's depth value comes from).

The depth of a fragment is calculated after perspective division. Thus, in normalized device coordinates. In NDC, $z = 1$ is furthest away from the camera. So, we specify a default value of the depth buffer so is as far away as possible.

Exercise 3.

Consider that we want to specify the vertices and colors of a triangle using a single array. Let $v1 = (0,0,0)$ be red. Let $v2 = (1,-0.5,0)$ be green. Let $v3 = (-1,-0.5,0)$ be blue. Modify L10.cpp (under OWL Resources/ClassDemos) to render this triangle as follows. You can also translate L10.cpp to Python and then modify it to fulfill the following.

Give one array of floats which gives the position of $v1$, then the color of $v1$, then the position of $v2$, then the color of $v2$, then the position of $v3$, then the color of $v3$. Thus, this array should have 18 entries. Then, specify two vertex attribute pointers using `glVertexAttribPointer` to access this one array and draw the triangle.

Exercise 4.

Repeat Exercise three, except now use `glDrawElements` rather than `glDrawArrays`. You will need to specify a new array of vertex indices: $\{0,1,2\}$

Submit two source codes to OWL for exercises 3 and 4 along with you answers to Exercises 1 and 2.