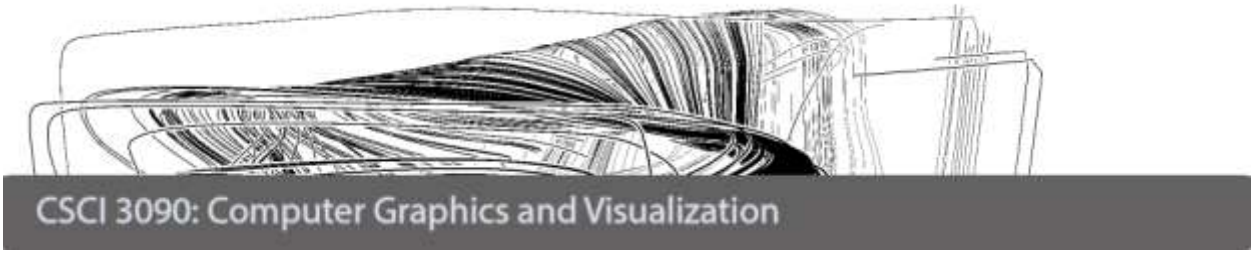


Laboratory Four: Projections

Introduction

In this laboratory you will explore orthographic and parallel projections. We will start with the program from laboratory one activity one (the single colored cube) and make three cubes in different positions. We will then display these cubes using the two projections to explore the differences between them.

Modifications

Start with the program from laboratory one activity one. It provides the vertices and normal vectors for the cube. We will display the cube three times from different position. This requires a modification of the displayFunc procedure as follows:

```
void displayFunc() {
	glm::mat4 model;
	glm::mat4 view;
	glm::mat4 viewPerspective;
	int modelLoc;
	int normalLoc;
	int viewLoc;
	int colourLoc;

	model = glm::mat4(1.0);

	view = glm::lookAt(glm::vec3(eyex, eyey, eyez),
				glm::vec3(0.0f, 0.0f, 0.0f),
				glm::vec3(0.0f, 0.0f, 1.0f));

	glm::mat3 normal = glm::transpose(glm::inverse(glm::mat3(view*model)));

	viewPerspective = projection * view;

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glUseProgram(program);
	modelLoc = glGetUniformLocation(program,"model");
	viewLoc = glGetUniformLocation(program,"viewPerspective");
	glUniformMatrix4fv(viewLoc, 1, 0, glm::value_ptr(viewPerspective));
	normalLoc = glGetUniformLocation(program,"normalMat");
	glUniformMatrix3fv(normalLoc, 1, 0, glm::value_ptr(normal));
```

```
        colourLoc = glGetUniformLocation(program,"colour");

        glBindVertexArray(triangleVAO);

        glUniform4f(colourLoc, 1.0, 0.0, 0.0, 1.0);
        glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
        glUniformMatrix3fv(normalLoc, 1, 0, glm::value_ptr(normal));
        glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_SHORT, NULL);

        model = glm::translate(model, glm::vec3(2.0, 2.0, 0.0));
        normal = glm::transpose(glm::inverse(glm::mat3(view*model)));
        glUniform4f(colourLoc, 0.0, 1.0, 0.0, 1.0);
        glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
        glUniformMatrix3fv(normalLoc, 1, 0, glm::value_ptr(normal));
        glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_SHORT, NULL);

        model = glm::translate(model, glm::vec3(-4.0, 2.0, 0.0));
        normal = glm::transpose(glm::inverse(glm::mat3(view*model)));
        glUniform4f(colourLoc, 0.0, 0.0, 1.0, 1.0);
        glUniformMatrix4fv(modelLoc, 1, 0, glm::value_ptr(model));
        glUniformMatrix3fv(normalLoc, 1, 0, glm::value_ptr(normal));
        glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_SHORT, NULL);

        glutSwapBuffers();

}
```

Note how we display the cubes, the first time we display it in the original position. The second time we translate it by (2, 2, 0), and the third time we translate it by (-2, 4, 0), the two translations are combined to give the final position.

We now make a change to the keyboardFunc procedure. We add two more commands, ‘p’ for perspective projection and ‘o’ for orthographic projection. The code that we need to add to this procedure is:

```
        case 'p':
               projection = glm::perspective(45.0f, 1.0f, 1.0f, 100.0f);
               break;
        case 'o':
               projection = glm::ortho(-5.0f, 5.0f, -5.0f, 5.0f, 1.0f, 100.0f);
               break;
```

The ortho procedure produces an orthographic projection matrix. The size parameters to this procedure are the minimum x value, the maximum x value, the minimum y value, the maximum y value, the minimum z value and the maximum z value.

The shader code is similar to what we have used before. The code for the vertex shader is:

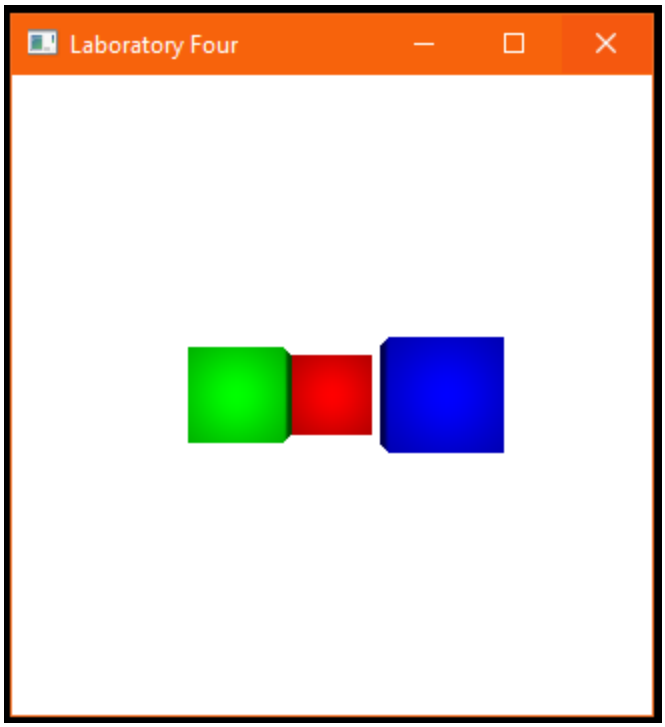
```
in vec4 vPosition;
in vec3 vNormal;
```

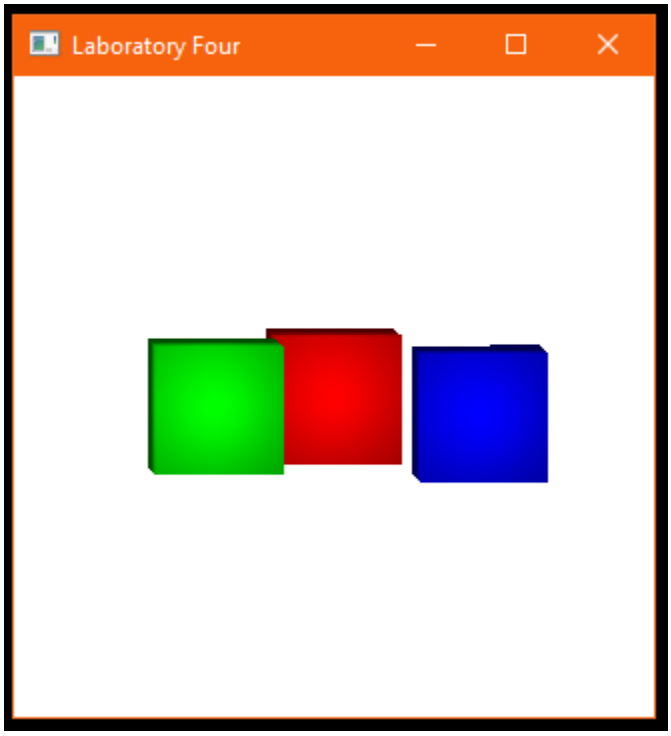
```
out vec3 normal;
uniform mat4 model;
uniform mat4 viewPerspective;
uniform mat3 normalMat;

void main() {

	gl_Position = viewPerspective * model * vPosition;
	normal = normalMat * vNormal;

}
```

The code for the fragment shader is:

```
in vec3 normal;
uniform vec4 colour;

void main() {

	vec3 N;
	vec3 L = vec3(0.0, 0.0, 1.0);
	float diffuse;

	N = normalize(normal);
	diffuse = dot(N,L);
	if(diffuse < 0.0) {
		diffuse = 0.0;
	}

	gl_FragColor = min(0.3*colour + 0.7*diffuse*colour, vec4(1.0));
	gl_FragColor.a = colour.a;
}
```

You now have everything that you need. You many need to change the viewing parameters slightly to see all three cubes. The results from your project should like similar to this (figure 1 is for perspective projection and figure 2 is for orthographic projection):


Laboratory Four


Laboratory Four

Other modifications

Add a key stroke command that allows you to interactively change the field of view (the first parameter to perspective) for the perspective projection. Could you do something similar for the orthographic projection? You should also try displaying the three cubes as a simple wireframe to better understand the difference between the two types of projections.

Laboratory Report

Submit your source code and a screenshot of your final result in the Blackboard dropbox.