Report

2019110023 LeshanWang

Overview

This project creates a graphical application that demonstrates the ability to render visual scenes using OpenGL, with several objects, shaders, and light sources. The report follows a structure which demonstrates and explains each achieved functionality in format of images, words, and segments of codes. Below shows the scene created:

The scene is constructed with eight objects: four pyramids with textures applied individually, three cubes as light sources with different colors, and one plain with texture set as the floor.

Objects

All objects implemented in the scene are constructed in code, the construction follows similar logic:

first set the coordinate, color, texture coordinate and normal of each vertex, and set indices for the vertices order (floor as example).

```
Vertex vertices1[] =
{
	Vertex{glm::vec3(-15.0f, 0.0f,  15.0f), glm::vec3(0.0f, 1.0f, 0.0f),
glm::vec3(1.0f, 1.0f, 1.0f), glm::vec2(0.0f, 0.0f)},
	Vertex{glm::vec3(-15.0f, 0.0f, -15.0f), glm::vec3(0.0f, 1.0f, 0.0f),
glm::vec3(1.0f, 1.0f, 1.0f), glm::vec2(0.0f, 1.0f)},
```

```
	Vertex{glm::vec3(15.0f, 0.0f, -15.0f), glm::vec3(0.0f, 1.0f, 0.0f),
glm::vec3(1.0f, 1.0f, 1.0f), glm::vec2(1.0f, 1.0f)},
	Vertex{glm::vec3(15.0f, 0.0f,  15.0f), glm::vec3(0.0f, 1.0f, 0.0f),
glm::vec3(1.0f, 1.0f, 1.0f), glm::vec2(1.0f, 0.0f)}

GLuint indices1[] =
{
	0, 1, 2,
	0, 2, 3
};
```

Then generate shader object using shader files. The shader file imports coordinates, colors, texture colors and normal of the object, also the light color, light position and camera position from main function. Within it calculates shader based on these information and light patterns. In this project there are three types of lights: direct light, spotlight and point light. The shader file calculates each pattern separately and the output can be set as any combination of the three.

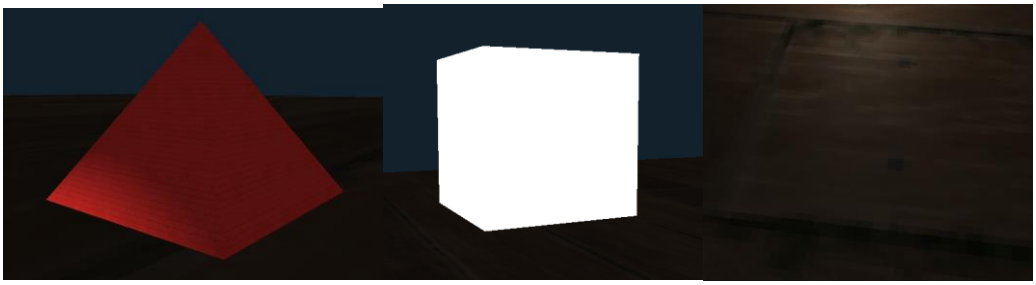
```
Shader shaderProgram("main/default.vert", "main/default.frag");
	std::vector <Vertex> vertsP(vertices, vertices + sizeof(vertices) /
sizeof(Vertex));
	std::vector <GLuint> indP(indices, indices + sizeof(indices) /
sizeof(GLuint));
	std::vector <Texture> texP(textures0, textures0 + sizeof(textures0) /
sizeof(Texture));
	Mesh pyramid(vertsP, indP, texP);
```

After creating shaders, Mesh.cpp will generate vertex buffer object and element buffer object and link them to vertex array object, which basically represents the object model. Then bind with texture and draw the actual object.

```
VAO.Bind();
	VBO VBO(vertices);
	EBO EBO(indices);
	VAO.LinkAttrib(VBO, 0, 3, GL_FLOAT, sizeof(Vertex), (void*)0);
	VAO.LinkAttrib(VBO, 1, 3, GL_FLOAT, sizeof(Vertex), (void*)(3 *
sizeof(float)));
	VAO.LinkAttrib(VBO, 2, 3, GL_FLOAT, sizeof(Vertex), (void*)(6 *
sizeof(float)));
	VAO.LinkAttrib(VBO, 3, 2, GL_FLOAT, sizeof(Vertex), (void*)(9 *
sizeof(float)));

...

glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_INT, 0);
```



Perspective Projection

The project has implemented a prospective projection that adapts to window size when resized.

Camera

The application implements a first-person style 3D camera which a user can navigate the scene using mouse and keyboard. When holding the left mouse key, user can look around; by pressing W, S, A, D, the user can move forward, backward, left, and right; when pressing E or Q, the user can move upward or downward. When holding LEFT CTRL or LEFT SHIFT with W, S, A, D, E, Q, the user can move with speed up or slow down. The camera uses glfwGetKey and glfwGetMouseButton to get keyboard and mouse input. These signals are further transferred to speed and direction.

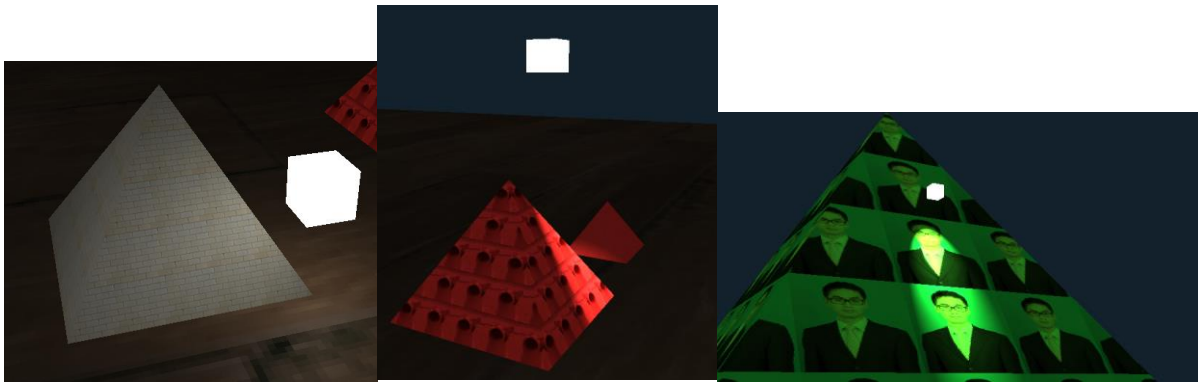
Point Light

The application includes three point lights, each has its own light color to be distinguished from others. The colors and coordinates of each light source are grouped into calculation when drawing object meshes, which means the light sources are not hard coded in shader files. The object will dynamically calculate its lighting properties according to the real-time position of the light source.

```
shaderProgramPlank.Activate();
      glUniformMatrix4fv(glGetUniformLocation(shaderProgramPlank.ID,
"model"), 1, GL_FALSE, glm::value_ptr(objectModel));
```

```
        glUniform4f(glGetUniformLocation(shaderProgramPlank.ID, "lightColor"),
lightColor.x, lightColor.y, lightColor.z, lightColor.w);
        glUniform3f(glGetUniformLocation(shaderProgramPlank.ID, "lightPos"),
lightPos.x, lightPos.y, lightPos.z);
```

The first source emits white light, the second source emits red light, and the third source emits green light.



Diffuse and Ambient Shading

After calculating the intensity of light with respect to distance, the ambient and diffuse lighting are calculated accordingly. They are finally calculated with texture properties and intensity, and output to the meshes.

```
vec3 lightVec = lightPos - crntPos;
float dist = length(lightVec);
float a = 3.0;
float b = 0.7;
float inten = 1.0f / (a * dist * dist + b * dist + 1.0f);

// ambient lighting
float ambient = 0.20f;
// diffuse lighting
vec3 normal = normalize(Normal);
vec3 lightDirection = normalize(lightVec);
float diffuse = max(dot(normal, lightDirection), 0.0f);

...

return (texture(tex0, texCoord) * (diffuse * inten + ambient) + texture(tex1,
texCoord).r * specular * inten) * lightColor;
```

Blinn-Phong lighting model

All three light sources implemented the Blinn-Phong lighting model.

```
float specular = 0.0f;
	if (diffuse != 0.0f)
	{
		float specularLight = 0.50f;
		vec3 viewDirection = normalize(camPos - crntPos);
		vec3 halfwayVec = normalize(viewDirection + lightDirection);
		float specAmount = pow(max(dot(normal, halfwayVec), 0.0f), 16);
		specular = specAmount * specularLight;
	};
	float angle = dot(vec3(0.0f, -1.0f, 0.0f), -lightDirection);
	float inten = clamp((angle - outerCone) / (innerCone - outerCone),
0.0f, 1.0f);

	return (texture(tex0, texCoord) * (diffuse * inten + ambient) +
texture(tex1, texCoord).r * specular * inten) * lightColor;
```

Texture Mapping

Five objects are mapped with textures.

```
	Texture textures1[]
	{
		Texture((texPath + "Chongshou.png").c_str(), "diffuse", 0,
GL_RGBA, GL_UNSIGNED_BYTE)
	};
```

The texture is read from local file, then used to create mesh of object.

```
	Shader shaderProgram4("main/default.vert", "main/defaultS.frag");

	std::vector <Vertex> vertsP4(vertices, vertices + sizeof(vertices) /
sizeof(Vertex));
	std::vector <GLuint> indP4(indices, indices + sizeof(indices) /
sizeof(GLuint));
	std::vector <Texture> texP4(textures1, textures1 + sizeof(textures1) /
sizeof(Texture));
	// Create pyramid mesh
	Mesh pyramid4(vertsP4, indP4, texP4);
```