ENVIRONMENT MAPPING - REFLECTION

Shizhe Bao (sxb150731)

Computer Graphics | 28th April 2017

INTRODUCTION

In this project, we aim at implementing a popular technique which makes use of Spheremaps is skyboxes which provides an illusion of a 3d background in a scenery by encasing the viewing camera and environment mapping which makes use of reflective properties in a scenery. For this we require six texture images, each corresponding to a face of the skybox, representing the surrounding environment; direction and reflection vectors to implement the reflective properties in the skybox. Reason to use the Spheremap technique is that it's an inexpensive implementation of environment maps which also allows us to reuse the environment map for any given viewing direction which is demonstrated by implementing reflection of the skybox on a sphere model.

AIM

The following are the goals of the project:

- **Implementing a Skybox using Sphere mapping**
  The skybox implementation using sphere-mapping will be done by using six texture images which will be mapped on a large sphere encasing the viewing camera, to give the perception of being present in a specific environment.

- **Implementing reflection on a model**
  The texture of the sphere map is reflected on the sphere face which is facing it.

- **Rotating the camera around the skybox**
  The camera can be rotated around the skybox and the 360-degree view of the skybox is visible to us.

- **Translating the object around the Spheremap**
  The object can be translated anywhere around the Spheremap. As the object move around the skybox, it reflects the texture of the Spheremap that it is facing.

PROCESS DESCRIPTION

In this section, I'll explain how we went about with the implementation of our project, which will be followed by screenshots of the final output in the Results section.

There are three main sections in my main.cpp.

Section-1

```
Model objModel;
if (!objModel.loadModel("sphere.obj")) //load sphere.obj
{
	glfwTerminate();
	std::system("pause");
	return -1;
}
```

Section-2

```
// 6 faces of box
std::vector<const char*> faces;
faces.push_back("sky_rt.jpg");
faces.push_back("sky_lf.jpg");
faces.push_back("sky_up.jpg");
faces.push_back("sky_dn.jpg");
faces.push_back("sky_bk.jpg");
faces.push_back("sky_ft.jpg");
SkyBox skybox;
skybox.init(faces);
```

Section-3

```
Shader shader("scene.vertex", "scene.frag"); // Reflection
Shader skyBoxShader("skybox.vertex", "skybox.frag"); // Background
```

Draw Scene

```
//Draw scene as normal
shader.use();
glm::mat4 projection = glm::perspective(camera.mouse_zoom, (GLfloat)(WINDOW_WIDTH) / WINDOW_HEIGHT, 0.1f, 100.0f);
glUniformMatrix4fv(glGetUniformLocation(shader.programId, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
glm::mat4 view = camera.getViewMatrix();
glUniformMatrix4fv(glGetUniformLocation(shader.programId, "view"), 1, GL_FALSE, glm::value_ptr(view));
glm::mat4 model;
glUniformMatrix4fv(glGetUniformLocation(shader.programId, "model"), 1, GL_FALSE, glm::value_ptr(model));
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_CUBE_MAP, skybox.getTextId());
glUniform1i(glGetUniformLocation(shader.programId, "envText"), 0);
glUniform3f(glGetUniformLocation(shader.programId, "cameraPos"), camera.position.x, camera.position.y, camera.position.z);
objModel.draw(shader);
```

Draw Skybox

```
skyBoxShader.use();
view = glm::mat4(glm::mat3(camera.getViewMatrix()));
glUniformMatrix4fv(glGetUniformLocation(skyBoxShader.programId, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(glGetUniformLocation(skyBoxShader.programId, "view"), 1, GL_FALSE, glm::value_ptr(view));
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_CUBE_MAP, skybox.getTextId());
glUniform1i(glGetUniformLocation(skyBoxShader.programId, "skybox"), 0);
skybox.draw(skyBoxShader);
```

Besides, there are three main actions I can do with my project.

Action1: mouse_move_callback

```
void mouse_move_callback(GLFWwindow* window, double xpos, double ypos)
{
    if (firstMouseMove)
    {
        lastX = xpos;
        lastY = ypos;
        firstMouseMove = false;
    }

    GLfloat xoffset = xpos - lastX;
    GLfloat yoffset = lastY - ypos;

    lastX = xpos;
    lastY = ypos;

    camera.handleMouseMove(xoffset, yoffset);
}
```

Action2: mouse_scroll_callback

```
void mouse_scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
    camera.handleMouseScroll(yoffset);
}
```

Action3: key_callback

```
if (keyPressedStatus[GLFW_KEY_W])
    camera.handleKeyPress(FORWARD, deltaTime);
if (keyPressedStatus[GLFW_KEY_S])
    camera.handleKeyPress(BACKWARD, deltaTime);
if (keyPressedStatus[GLFW_KEY_A])
    camera.handleKeyPress(LEFT, deltaTime);
if (keyPressedStatus[GLFW_KEY_D])
    camera.handleKeyPress(RIGHT, deltaTime);
if (keyPressedStatus[GLFW_KEY_O])
    camera.handleKeyPress(UP, deltaTime);
if (keyPressedStatus[GLFW_KEY_P])
    camera.handleKeyPress(DOWN, deltaTime);
```

Vertex Shader and Fragment Shader

Scene.vertex

```
#version 330 core

layout(location = 0) in vec3 position;
layout(location = 1) in vec2 textCoord;
layout(location = 2) in vec3 normal;


uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;

out vec3 FragNormal;
out vec3 FragPos;

void main()
{
	gl_Position = projection * view * model * vec4(position, 1.0);
	FragPos = vec3(model * vec4(position, 1.0));
	mat3 normalMatrix = mat3(transpose(inverse(model)));
	FragNormal = normalMatrix * normal;
}
```

Scene.frag

```
#version 330 core

in vec3 FragNormal;
in vec3 FragPos;

uniform samplerSphere envText;
uniform vec3 cameraPos;

out vec4 color;


void main()
{
	vec3 viewDir = normalize(FragPos - cameraPos);
	vec3 reflectDir = reflect(viewDir, normalize(FragNormal));
	color = texture(envText, reflectDir);
}
```

Skybox.vertex

```
#version 330 core

layout(location = 0) in vec3 position;


uniform mat4 projection;
uniform mat4 view;

out vec3 TextCoord;

void main()
{
	vec4 pos = projection * view * vec4(position, 1.0);
	gl_Position = pos.xyww;
	TextCoord = position;
}
```

Skybox.frag

```
#version 330 core

in vec3 TextCoord;
uniform samplerSphere  skybox;

out vec4 color;


void main()
{
	color = texture(skybox, TextCoord);
}
```

Reflection

Reflection is the property of an object (or its part) to reflect its surrounding. For example, a mirror. Reflection vector is calculated using GLSL's built-in reflect function. The resulting vector R is then used as a direction vector to sample the Spheremap returning a color value of the environment. The resulting effect is that the object seems to reflect the skybox. First, I calculate the camera direction vector I and use it to calculate the reflection vector R which is then used to sample from the skybox Spheremap.

RESULT