Real-Time Computer Graphics— (Alien Breed

Student Name: **Yazeed Abdulhai – st20193167**

Introduction

I was required to produce a remake demo for Alien Breed, this project tasked me to create the models, texture and render them properly using OpenGL. However, I created my objects using Blender and baked their textures as well.

In the code snippets below, I will try to demonstrate briefly how did I develop and produce my rendered 3D models, textures, interaction, lighting, and transparency.

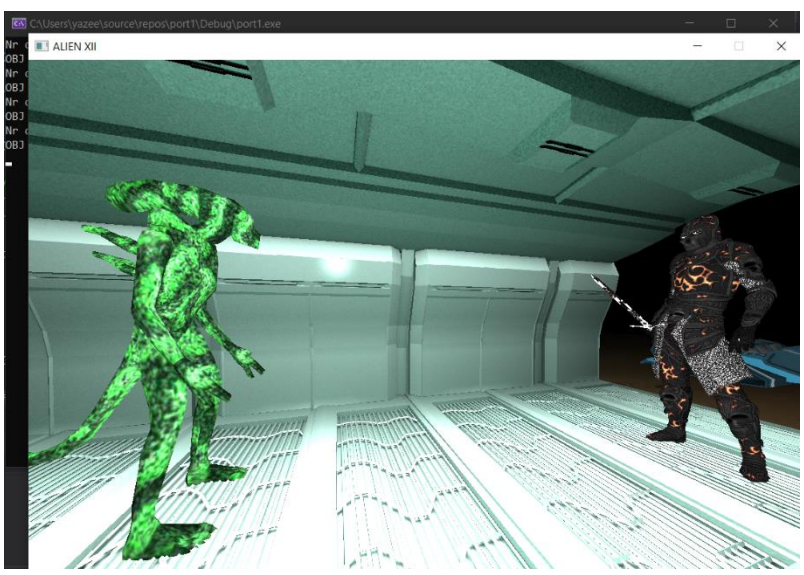
Model Rendering

Rendering a 3D model required a piece of code to identify the sequence of the model data in an OBJ file. Each object has its own vertices, vertex normal, vertex textures and faces. Nevertheless, some objects required some scaling and rotation using the model matrices as the knight was facing the opposite side and the alien was too big.

Each object in the scene gets imported and rendered with its own position, diffuse, and specular textures. In addition to that I have added some functions to be accessed by the model to translate, scale, or rotate it. The way the code was built gave me the ability to apply changes on any object in the scene.

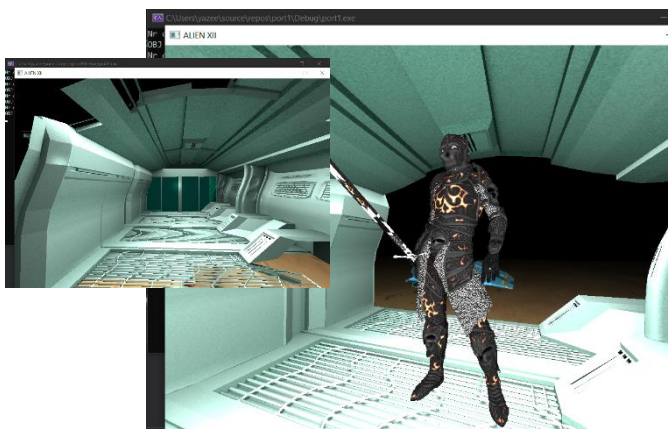
```
//Knight model
this->models.push_back(new Model(
    glm::vec3(-2.5f, -2.f, 24.f),
    this->materials[0],
    this->textures[TEX_CHARACTER],
    this->textures[TEX_CHARACTER_SPEC],
    "Objects/knight.obj"
)
);
this->models[1]->rotate(glm::vec3(0.f, -180.f, 0.f)); //Rotating the Knight to look in the opposite side
```

```
void updateModelMatrix()
{
    this->ModelMatrix = glm::mat4(1.f);
    this->ModelMatrix = glm::translate(this->ModelMatrix, this->origin);
    this->ModelMatrix = glm::rotate(this->ModelMatrix, glm::radians(this->rotation.x), glm::vec3(1.f, 0.f, 0.f));
    this->ModelMatrix = glm::rotate(this->ModelMatrix, glm::radians(this->rotation.y), glm::vec3(0.f, 1.f, 0.f));
    this->ModelMatrix = glm::rotate(this->ModelMatrix, glm::radians(this->rotation.z), glm::vec3(0.f, 0.f, 1.f));
    this->ModelMatrix = glm::translate(this->ModelMatrix, this->position - this->origin);
    this->ModelMatrix = glm::scale(this->ModelMatrix, this->scale);
}
```



Texture Mapping

While the object is being rendered, we go through the object's VBO that retains its texture coordinates, where it can be easily attached to the model as it got baked on that object using blender. The rendered textures can be seen on the Knight, hallway, and the spaceship.



Lighting

Three different types of lighting can be used to light a game scene up. However, I have used several point light sources to light my scene up. Two of my light sources are shown in the images on the right side. The way you light up your scene can be controlled by specifying values for the diffuse and specular lights. However, those values are sent to the shader to do some mathematical operations and return the final shader color at the end of the processing.

Interaction

In game, the simplest way to represent a player through interaction is by keyboard presses and mouse movement. The most popular way for PC games to handle player movements is by using the WASD keyset. When the player presses a key, the event handler picks it up and calls the updateKeyboardInput(). In case ***W*** has been pressed, the spaceship is going to move forward (in-screen) along the negative z-axis with a specified value from its current position. Nevertheless. our camera is following the spaceship (player) by applying the same movement method. Some camera movements were also applied based on the angle of the mouse movements.

```
void Game::updateMouseInput()
{
    glfwGetCursorPos(this->window, &this->mouseX, &this->mouseY);

    if (this->firstMouse)
    {
        this->lastMouseX = this->mouseX;
        this->lastMouseY = this->mouseY;
        this->firstMouse = false;
    }

    //Calculate offset
    this->mouseOffsetX = this->mouseX - this->lastMouseX;
    this->mouseOffsetY = this->lastMouseY - this->mouseY;

    //Set last X and Y as the current one
    this->lastMouseX = this->mouseX;
    this->lastMouseY = this->mouseY;
```

```
void Game::updateKeyboardInput()
{
    //Program
    if (glfwGetKey(this->window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(this->window, GLFW_TRUE);

    //Camera
    if (glfwGetKey(this->window, GLFW_KEY_W) == GLFW_PRESS)
    {
        this->camera.move(this->dt, FORWARD);
        this->pointLights[0]->setPosition((this->camera.getPosition()) + glm::vec3(0.f, 0.f, -20.f));
        /*Moving the lights with the camera
        and the added vec3 is to give the illusion of our spaceship having front lights like a car*/

        this->models[4]->move(glm::vec3(0.f, 0.f, -0.021f)); //Moving the spaceship with the camera
    }
    if (glfwGetKey(this->window, GLFW_KEY_S) == GLFW_PRESS)
    {
        this->camera.move(this->dt, BACKWARD);
        this->pointLights[0]->setPosition((this->camera.getPosition()) + glm::vec3(0.f, 0.f, -20.f));
        this->models[4]->move(glm::vec3(0.f, 0.f, 0.021f));
    }
    if (glfwGetKey(this->window, GLFW_KEY_A) == GLFW_PRESS)
    {
        this->camera.move(this->dt, LEFT);
        this->pointLights[0]->setPosition((this->camera.getPosition()) + glm::vec3(0.f, 0.f, -20.f));
        this->models[4]->move(glm::vec3(-0.021f, 0.f, 0.f));
    }
    if (glfwGetKey(this->window, GLFW_KEY_D) == GLFW_PRESS)
    {
        this->camera.move(this->dt, RIGHT);
        this->pointLights[0]->setPosition((this->camera.getPosition()) + glm::vec3(0.f, 0.f, -20.f));
        this->models[4]->move(glm::vec3(0.021f, 0.f, 0.f));
    }
}
```