Universidade da Beira Interior
Departamento de Informática

[Trabalho de Computação Gráfica]
Licenciatura em Engenharia Informática

Elaborado por:

Gustavo Pina 50072

23 de novembro de 2024

Índice

Capítulo

1

Introdução

O objetivo deste trabalho é, explorar o uso do formato obj ao carregar esse ficheiros para uma cena, movimentar um desses objetos(a nave), conseguir definir onde está o observador e finalmente conseguir aplicar iluminação a uma cena.

Capítulo

2

Problema

O problema principal é criar uma cena que carregue ficheiros obj e os apresente. Devemos também conseguir mover a nave entre dois pontos (os hangar) e ter esta cena devidamente iluminada.

Capítulo

3

Solução

Como solução para carregar os obj foi usada a livraria tiny obj loader com ela o carregamento dos dados dos obj para o programa foi relativamente simples. Em relação ao movimento foi usada uma simples função que deteta quando certas teclas são pressionadas e aplica uma transação à nave. Por fim a iluminação foi feita através do método de phong.

Capítulo

4

Implementação

4.1 Ficheiros OBJ

Para carregar um ficheiro obj começamos por criar as variáveis do tipo tiny obj loader que serão necessárias para chamar a função. O resto do trabalho é feito pela livraria e apenas resta fazer a extração dos vértices e normais.

```
    // Fun  o para carregar o modelo OBJ
bool loadOBJ(const char* path, std::vector<float>& out_vertices, std::
    vector<float>& out_normals)
{
    tinyobj::attrib_t attrib;
    std::vector<tinyobj::shape_t> shapes;
    std::vector<tinyobj::material_t> materials;
    std::string warn, err;

    bool ret = tinyobj::LoadObj(&attrib, &shapes, &materials, &warn, &
        err, path);
    if (!ret) {
        fprintf(stderr, "O Obj n o carregou bem: %s\n", path);
        return false;
    }

    // Extrair os v rtices e normais
    for (const auto& shape : shapes) {
        for (const auto& index : shape.mesh.indices) {
            out_vertices.push_back(attrib.vertices[3 * index.
                vertex_index + 0]);
            out_vertices.push_back(attrib.vertices[3 * index.
                vertex_index + 1]);
            out_vertices.push_back(attrib.vertices[3 * index.
                vertex_index + 2]);
```

```
            out_normals.push_back(attrib.normals[3 * index.normal_index
                + 0]);
            out_normals.push_back(attrib.normals[3 * index.normal_index
                + 1]);
            out_normals.push_back(attrib.normals[3 * index.normal_index
                + 2]);
        }
    }
    return true;
}
```

4.2 Movimentação

A função de movimentação é simples e apenas consiste em aplicar transações ou rotações dependendo das teclas que são pressionadas.

```
    //Funcao para o movimento da nave
void controloNave(GLFWwindow* window, int key, int scancode, int action,
     int mods) {
    if (action == GLFW_PRESS || action == GLFW_REPEAT) {
        if (key == GLFW_KEY_D) {
            modelX += modelSpeed;
        }
        if (key == GLFW_KEY_A) {
            modelX -= modelSpeed;
        }
        if (key == GLFW_KEY_W) {
            modelZ += modelSpeed;
        }
        if (key == GLFW_KEY_S) {
            modelZ -= modelSpeed;
        }
        if (key == GLFW_KEY_E) {
            modelRotationY += 1.0f;
        }
        if (key == GLFW_KEY_Q) {
            modelRotationY -= 1.0f;
        }
    }
}


//A posicao da nave depende dos valores que s o alterados na fun   o
    controloNave
glm::mat4 falconModel = glm::translate(glm::mat4(1.0f),
glm::vec3(modelX, 0.0f, modelZ));
    falconModel = glm::rotate(falconModel, glm::radians(modelRotationY),
```

```
        glm::vec3(0.0f, 1.0f, 0.0f));
```

4.3 Iluminação

Por fim a iluminação é feita com o modelo de phong onde calculamos a cor para cada ponto.

```
    #version 330 core

in vec3 fragmentColor;
in vec3 fragPosition_cameraSpace;
in vec3 fragNormal_cameraSpace;

out vec3 color;

uniform vec3 lightPosition_cameraSpace;
uniform vec3 lightColor;
uniform vec3 ambientColor;
uniform float shininess;
uniform float strength;

void main() {
    vec3 N = normalize(fragNormal_cameraSpace);

    vec3 L = normalize(lightPosition_cameraSpace -
        fragPosition_cameraSpace);

    vec3 ambient = ambientColor * fragmentColor;

    float diff = max(dot(N, L), 0.0);
    vec3 diffuse = lightColor * diff * fragmentColor;

    vec3 viewDir = normalize(-fragPosition_cameraSpace);
    vec3 reflectDir = reflect(-L, N);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
    vec3 specular = strength * spec * lightColor;

    color = ambient + diffuse + specular;
}
```

Capítulo

5

Conclusão

Em suma, com o uso do tiny obj loader conseguimos carregar e apresentar modelos bastante complexos sem os definir ponto a ponto no código como era feito anteriormente. Exploramos também a movimentação de objetos controlados pelo utilizador e o método de iluminação de phong para melhorar as nossas cenas.