Universidade da Beira Interior
Departamento de Informática

2023: *Simulação do Sistema Solar em OpenGL*

Elaborado por:

Afonso Martins (45838)

Diogo Ferreira (46198)

Orientador:

Professor Doutor Abel J.P. Gomes

9 de janeiro de 2024

Agradecimentos

A conclusão deste trabalho, não seria possível sem a ajuda do professor Abel J.P. Gomes que nos ajudou com inúmeras dúvidas e deu muitas ideias para nos facilitar o desenvolvimento da aplicação. Além disso, foi o mesmo que nos ensinou e deixou cada vez mais curiosos acerca do que se pode fazer com o *OpenGL*

Conteúdo

Lista de Figuras

Lista de Excertos de Código

Acrónimos

CG	Computação Gráfica
SO	Sistema Operativo
SS	Sistema Solar
VM	*Virtual Machine*
UBI	Universidade da Beira Interior
WSL	Windows Subsystem for Linux

Capítulo

1

Introdução

1.1 Enquadramento

Este relatório foi feito no contexto da unidade curricular de Computação Gráfica (CG) da Universidade da Beira Interior (UBI) tal como o *software* ao qual se refere. Todo o projeto foi desenvolvido na UBI

1.2 Motivação

Este trabalho foi desenvolvido visando ser a avaliação final da cadeira de CG e tendo sido escolhido o desenvolvimento de um Sistema Solar (SS) pelos desenvolvedores.

1.3 Objetivos

O objetivo da aplicação é criar uma simulação do nosso SS o mais realista possível em *OpenGL*

1.4 Organização do Documento

De modo a refletir o trabalho feito, este documento encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta o projeto, a motivação para a sua escolha, o enquadramento para o mesmo, os seus objetivos e a respetiva organização do documento.

2. O segundo capítulo – **Tecnologias Utilizadas** – descreve os conceitos-chave no âmbito deste projeto, bem como as tecnologias utilizadas durante do desenvolvimento da aplicação.

3. O terceiro capítulo – **Etapas de Desenvolvimento** – descreve os passos e todas as etapas ultrapassadas para desenvolvimento deste mesmo projeto.

4. O quarto capítulo – **Funcionamento do *Software*** – descreve alguma da lógica por detrás da aplicação e como utilizar a mesma sem dificuldade.

5. O quinto e ultimo capítulo – **Conclusões e Trabalhos Futuros** – dá a conhecer os conhecimentos retirados deste projeto, e o que se conclui com o desenvolvimento do mesmo. Além disso, apresenta algumas das funcionalidades que os desenvolvedores da mesma pretendem adicionar.

Capítulo

2

Tecnologias Utilizadas

2.1 Introdução

Este capitulo dá uma breve descrição de cada uma das tecnologias utilizadas para o desenvolvimento do nosso SS e uma explicação do porquê de termos escolhido essa mesma tecnologia.

2.2 Tecnologias

As principais tecnologias utilizadas foram *OpenGL*, C++ e o Windows Subsystem for Linux (WSL) como principal fonte para correr o programa utilizando uma linha de compilação e execução específica.

2.2.1 *Blender*

O *Blender* é um programa de *design* 3D e foi usado para facilitar o código, visto que em vez de se ter de criar funções para criar esferas ou até mesmo anéis, tal como de Saturno, apenas se criou no *Blender* e usaram-se funções dentro do mesmo para se obter os ficheiros OBJ com as faces triangulares.

2.2.2 *OpenGL*

O *OpenGL* é a principal tecnologia, é a base de todo o projeto, visto que sem as funções do mesmo não seria possível tão facilmente renderizar os nossos planetas, ou simular luzes, ou mover no nosso ambiente.

2.2.3 C++

O C++ é a linguagem no qual tudo foi programado, uma linguagem de programação muito potente e com a junção ao *OpenGL* faz com que tudo o que queremos fazer no SS seja possível.

2.2.4 WSL

O WSL funciona sendo quase como uma simulação do *Linux* e facilitando o uso de comandos de compilação e execução do programa, o que deu muito jeito, por haver mais suporte para o *Linux* do que havia para o *Windows* e visto que os desenvolvedores usam ambos *Windows* era mais simples a utilização do WSL do que uma *Virtual Machine* (VM) ou *dual-boot* ou até mesmo trocar de Sistema Operativo (SO) para uma dos subsistemas Linux.

2.3 Conclusões

Concluímos este capitulo apercebendo que para um programa até simples como o nosso simulador de um SS são precisas várias tecnologias cada uma com o seu forte para depois no todo o ser algo capaz de fazer tudo o que o desenvolvedor pretender.

Capítulo

3

Etapas de Desenvolvimento

3.1 Introdução

Neste capitulo serão apresentadas as várias etapas do desenvolvimento com uma breve descrição do que se fez em cada uma das mesmas etapas

3.2 Etapas

As etapas de desenvolvimento não foram estruturadas, porém, os problemas principais que se teve de ultrapassar foram os seguintes:

- **Escolha da base do programa**
- **Design dos Planetas**
- ***Load* dos planetas**
- **Adição de Funcionalidades**

3.2.1 Escolha da base do Programa

O professor da cadeira de CG tinha disponibilizados vários esqueletos de código durante as aulas práticas [1] da cadeira, logo foi assumido como um bom ponto de partida escolher um desses esqueletos, sendo esse baseado num cubo que ilumina o ambiente todo tal como o sol do SS. Assim sendo, foi visto como sendo um bom ponto de partida.

3.2.2 Design dos Planetas

Para o *design* dos planetas foi utilizado o *Blender* visto que o mesmo permite reduzir a densidade de polígonos e em vez de se usar funções mais complexas para desenhar as esferas que são os planetas apenas se carregou os vértices do ficheiro OBJ. Outra razão da nossa escolha de utilizar o *Blender* foi para podermos ter Saturno num modelo apenas para simplificar a programação

3.2.3 *Load* dos planetas

Após o *design* dos planetas, os mesmos precisavam ser carregados para a cena. Para tal foi preciso encontrar uma função que lesse os ficheiros OBJ e tratasse de organizar os vértices e todos os outros componentes. Depois como usar esses dados que foi através de quase sempre os mesmos excertos de código que consistiam em guardar os vértices num *array* para desenhar os vértices, ativar a luz para os mesmos, desenhar os polígonos e mostrar no ecrã.

3.2.4 Adição de Funcionalidade

Após estar tudo resolvido em termos de ver os planetas e carregamento de imagem foram criadas funções para o utilizador poder interagir livremente com ambiente. São exemplos dessas funcionalidades as seguintes:

- Movimento utilizando o rato para mover a câmara e analisar o que rodeia o utilizador;
- Movimento espacial podendo aproximar dos planetas e flutuar pelo SS;
- Mover os planetas podendo acelerar a velocidade de rotação dos mesmos ou avagarar. No movimento, todos os planetas têm uma velocidade diferente para melhor simular a realidade espacial;
- Obter informação dos planetas ao aproximando e clicando aparecendo na linha de comando, neste o WSL.

3.3 Conclusões

Ao analisar este capitulo apercebemo-nos que um *software* mesmo não tendo etapas definidas muitas vezes acabam por ter desafios a ultrapassar os quais acabam por se tornar as mesmas etapas de desenvolvimento. Dentro das mesmas há muito desenvolvimento para que tudo corra o melhor possível e se obtenha uma aplicação sólida.

Capítulo

4

Funcionamento do Software

4.1 Introdução

Uma aplicação independentemente do nível de complexidade necessita sempre de algumas funções as quais serão explicadas nesta secção tal como utilizar este *software* para tirar o máximo proveito

4.2 Interação com o *software*

A aplicação não é um executável pré-compilado, logo a interação com a mesma reparte-se em duas partes, a compilação e execução da mesma e como utilizar todas as *features*

4.2.1 Compilação e execução

A compilação e execução do *software* é uma das partes principais para que o utilizador possa usufruir do programa, porém as mesmas podem ser repartidas em passos simples. Os passos são os seguintes:

1. Navegar até à página do *GitHub* e fazer *download* do ficheiro ZIP
2. Extrair o ficheiro ZIP para uma pasta
3. Abrir o WSL e navegar até a pasta para a qual foi extraido o ZIP
4. Executar o comando `g++ main.cpp glad/src/glad.c objloader.cpp -o rotating -I/usr/include/freetype2 -lglfw -lGLEW -lGL -lX11 -lpthread -lXrandr -ldl -lfreetype`

Pode ser necessário instalação de algumas bibliotecas no WSl

5. Se o passo anterior correr bem sem erros então é só executar o comando `./rotating`

4.2.2 Controles

Os controles desta aplicação foram o mais simplificado possível para todo o utilizador poder utilizar sem dificuldade. As descrições e devidos controles são os seguintes:

- Rotação dos planetas para um sentido ou para o outro - **Seta Esquerda** e **Seta Direita**
- Rotação dos planetas para um sentido ou para o outro - **Seta Esquerda** e **Seta Direita**
- Aceleração da rotação dos planetas e desaceleração - **E** e **Q**
- Movimentação espacial para a frente e para trás - **W** e **S**
- Movimentação espacial para a direita e para a esquerda - **D** e **A**
- Aumento da velocidade espacial e redução - **SHIFT** e **CTRL**
- Olhar em redor- **Rato**
- Obter informação de um planeta (só é possível quando extremamente próximo do mesmo)- **Tecla Esquerda do Rato**

4.3 Funcionamento

O *software* tem inúmeras funções que o permitem funcionar de forma simples. As principais partes da aplicação são a renderização e funcionalidades. A renderização tem por base tudo o que tenha a ver com a renderização dos planetas e da iluminação. As funcionalidades são todas as funcionalidades que permitem o utilizador interagir com o ambiente.

4.3.1 Renderização

Para a renderização dos planetas foi utilizada a biblioteca *objloader* visto que foram utilizados ficheiros OBJ. Assim sendo o principal era carregar para o *array* de desenho

```
//MERCURIO
    glGenVertexArrays(1, &MercurioVAO);
    glGenBuffers(1, &vertexbufferMercurio);

    glBindBuffer(GL_ARRAY_BUFFER, vertexbufferMercurio);
    glBufferData(GL_ARRAY_BUFFER, verticesMercurio.size() * sizeof(glm::
        vec3), &verticesMercurio[0], GL_STATIC_DRAW);
    glBindVertexArray(MercurioVAO);

    glGenBuffers(1, &colorbufferMercurio);
    glBindBuffer(GL_ARRAY_BUFFER, colorbufferMercurio);
    glBufferData(GL_ARRAY_BUFFER, verticesMercurio.size() * sizeof(glm::
        vec3), &verticesMercurio[0], GL_STATIC_DRAW);

    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
    glEnableVertexAttribArray(1);
```

Excerto de Código 4.1: Exemplo de carregamento de um planeta para os buffers de desenho

Após estar os vértices prontos a serem desenhados então entra no *loop* principal de desenho onde é colocada a luz que faz as contas de como aplicar as sombras. Além disso, também se define rotação localização e cores. Depois de tudo é mandado desenhar os polígonos e mostrar esses mesmos polígonos no ecrã

```
    // Render Mercurio
        lightingShader.use();
        lightingShader.setVec3("objectColor", 1.0f, 1.0f, 0.192f);  //
            Set color for Mercurio
        lightingShader.setVec3("lightColor", 1.0f, 1.0f, 1.0f);
        lightingShader.setVec3("lightPos", lightPos);

        // Set model matrix for Mercurio
        modelMercurio = glm::translate(glm::mat4(1.0f), mercurioPosition
            );
        modelMercurio = glm::rotate(modelMercurio, anguloMercurio, glm::
            vec3(0.0f, 1.0f, 0.0f));  // Rota  o em torno do eixo y
        modelMercurio = glm::scale(modelMercurio, glm::vec3(escala));
        lightingShader.setMat4("model", modelMercurio);

        // Bind Mercurio VAO and render
        glBindVertexArray(MercurioVAO);
        glDrawArrays(GL_TRIANGLES, 0, 960*3);
        glBindVertexArray(0);  // Unbind Mercurio VAO
```

Excerto de Código 4.2: Exemplo de renderização de um planeta

Quando todo o tratamento de imagem é completo então os *buffers* são eliminados para não ficarem em memória

```
glDeleteVertexArrays(1, &MercurioVAO);
glDeleteBuffers(1, &vertexbufferMercurio);
```

Excerto de Código 4.3: Limpeza dos buffers

4.3.2 Funcionalidades

As funcionalidades podem ser repartidas em 3 campos principais:

- **Movimentação dos Planetas**, onde o utilizador pode rodar os planetas em torno do sol ao seu comando simulando o SS;
- **Movimentação do utilizador**, onde o utilizador pode usar o teclado e rato para se aproximar dos planetas e andar no meio dos mesmos;
- **Obtenção de Informação**, onde o utilizador pode ao clicar nos planetas obter um set de informação acerca dos mesmos.

4.3.2.1 Movimentação dos Planetas

A movimentação dos planetas foi feita atualizando o campo *vec3* associada a cada um dos planetas. Quando se aguenta a tecla, então o ângulo relativamente ao inicial aumenta e então a posição é calculada tendo em conta que a rotação do planeta é circular.

```
if (glfwGetKey(window, GLFW_KEY_LEFT) == GLFW_PRESS) {
        anguloMercurio += MercurioRotationSpeed;
        }
if (glfwGetKey(window, GLFW_KEY_RIGHT) == GLFW_PRESS) {
        anguloMercurio -= MercurioRotationSpeed;
        }
 mercurioPosition = glm::vec3(glm::cos(anguloMercurio) * 5.8/escala,
     0.0f, glm::sin(anguloMercurio) * 5.8/escala);
```

Excerto de Código 4.4: Rotação de Mercurio

O cálculo da rotação é feita utilizando o círculo trigonométrico, visto que consoante o seno e o cosseno do ângulo atual a posição no espaço é multiplicada pelo valor do seno e cosseno do mesmo.

```
if (glfwGetKey(window, GLFW_KEY_E) == GLFW_PRESS){
        speedScale=speedScale+0.01f;
    }
    if ((glfwGetKey(window, GLFW_KEY_Q) == GLFW_PRESS) && speedScale
        >0.01f ){
```

```
        speedScale=speedScale-0.01f;
    }
```

Excerto de Código 4.5: Aumento e diminuição da velocidade de rotação de Mercurio

4.3.2.2 Movimentação do utilizador

A movimentação do utilizador é, na verdade, apenas a alteração da posição da câmara no espaço, sendo que a mesma se move no espaço e o utilizador pode olhar no seu rodando apenas utilizando o teclado e rato. O utilizador também pode acelerar a velocidade a que se move espacialmente.

```
    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
        camera.ProcessKeyboard(FORWARD, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
        camera.ProcessKeyboard(BACKWARD, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        camera.ProcessKeyboard(LEFT, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        camera.ProcessKeyboard(RIGHT, deltaTime);
    if (glfwGetKey(window, GLFW_KEY_E) == GLFW_PRESS){
        speedScale=speedScale+0.01f;
    }
    if (glfwGetKey(window, GLFW_KEY_LEFT_SHIFT) == GLFW_PRESS)
        camera.MovementSpeed+=0.5f;
    if ((glfwGetKey(window, GLFW_KEY_LEFT_CONTROL) == GLFW_PRESS) && (
        camera.MovementSpeed>0.5f))
        camera.MovementSpeed-=0.1f;
```

Excerto de Código 4.6: Movimentação espacial da camera

4.3.2.3 Obtenção de Informação

Para obter informação do planeta é necessário primeiro clicar no mesmo, mas isso seria simples num ambiente 2D em vez de 3D. Assim sendo é feita essa transformação sempre que se clica no rato, é calculada a projeção e se no lugar onde se clicou estiver o planeta então é impresso no WSL a informação que se pretende mostrar.

```
    if (glfwGetMouseButton(window, GLFW_MOUSE_BUTTON_LEFT) == GLFW_PRESS
        ) {

        // Verifique se algum planeta foi clicado
        if (checkPlanetClick(mercurioPosition, 0.1f, clickRayOrigin,
            clickRayDirection)) {
            // A   o quando Mercurio    clicado
```

```
            std::cout << "\n";
            std::cout << "Planeta: Merc rio\nDiametro: 4,879.4 KM\
                nMassa: 3.285     10^23 KG\nDura ao do dia: 88 d " <<
                std::endl;
        }
```

Excerto de Código 4.7: Obtenção de infromação de um planeta

4.4 Conclusões

Com este capítulo conseguimos perceber que muitos dos comandos que se utilizam têm de ser programados e apesar de terem alguma complexidade depois com a ajuda do *OpenGL*. Além disso, é possível concluir que num programa pré-compilado tal como o qual este relatório se refere é preciso fazer alguns passos importantes antes de se poder aproveitar para usufruir da aplicação, muito possivelmente para usos lúdicos.

Capítulo

5

Conclusões e Trabalho Futuro

5.1 Conclusões Principais

Com este trabalho é possível concluir que é possível criar ambientes tridimensionais, renderizar e criar ambientes tão realistas quanto pretendermos apenas utilizando C++ e não tendo de ser "escravos"das grandes tecnologias de renderização tridimensional tais como *Blender*, *Unreal Engine* ou *Unity* entre os demais. Assim sendo conseguimos criar programas muito mais leves, pois apenas está incorporado no mesmo todas as tecnologias que se necessita.

Apesar de tudo também é muito mais complexo visto que muito do que as outras tecnologias já têm pré-programado, no caso do OpenGL é preciso programar à mão.

Concluindo este *software* apesar de ser algo simples, deu para dar a conhecer as funcionalidades que o OpenGL tem para oferecer e melhor perceber tudo o que está por detrás dos grandes *softwares* gráficos.

5.2 Trabalho Futuro

Este *software* apesar de dado como terminado nunca está realmente, visto que se pode sempre adicionar algo mais para o tornar melhor. São exemplos dessas melhorias a adição de texturas de forma tornar os planetas mais realistas, algo que os desenvolvedores tentaram adicionar, porém, as mesmas desligavam as sombras e foi visto como prioridade as sombras. Outra melhoria que ficou para trabalho futuro é a adição de texto gráfico para que o utilizador não precise de desviar os olhos da aplicação focando-se apenas na mesma para, por exemplo, ler a informação ao clicar nos planetas. A melhorar

também se encontra a capacidade de clicar no planeta sem ser preciso estar tão perto.

5.3 Consideração Final

Este trabalho é dado por concluído, tendo aprendido muito graças ao senhor professor Abel Gomes, regente da cadeira de CG. Terminamos assim com uma imagem que tanta hora olhamos para ter a certeza que tudo funcionava dentro dos conformes apesar de nunca perfeito

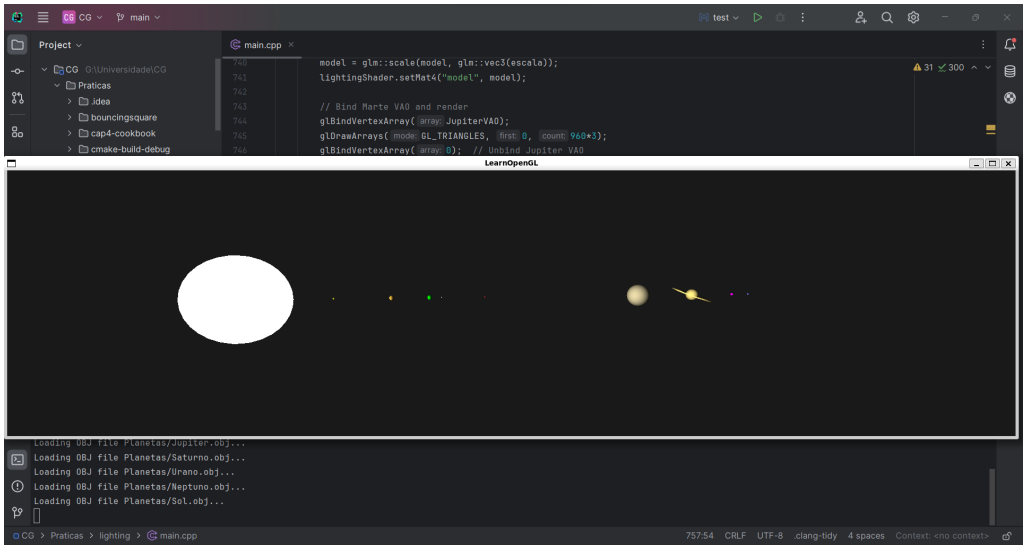

Figura 5.1: Imagem do SS na sua fase final

Glossário

Blender O Blender é um conjunto de ferramentas de software de computação gráfica 3D gratuito e de código aberto utilizado para criar filmes de animação, efeitos visuais, arte, modelos impressos em 3D, gráficos em movimento, aplicações 3D interactivas, realidade virtual e, anteriormente, jogos de vídeo.. iii, 3, 13

C++ É uma linguagem de programação de alto nível e de uso geral.. 3, 13

dual-boot Capacidade de um computador de inicializar em dois sistemas operacionais diferentes.. 4

Linux Sistema operacional de código aberto.. 4

OBJ Extensão de arquivo usada para armazenar modelos 3D.. 3, 6, 8

OpenGL O OpenGL é uma interface de programação de aplicações multilinguagem e multi-plataforma para renderização de gráficos vectoriais 2D e 3D.. iii, 1, 3, 12, 13

Unity Unity é um motor de jogo multiplataforma.. 13

Unreal Engine O Unreal Engine é uma série de motores de jogos 3D para computação gráfica.. 13

ZIP Formato de arquivo usado para compactar arquivos.. 7

Bibliografia

[1] Abel J. P. Gomes. Computer graphics labs lab.6, 2020. ILLUMINATION AND SHADING OF 3D SCENES.