Universidade da Beira Interior
Departamento de Informática

Projeto #2 - Sistema Solar 2D/3D

Elaborado por:
Diogo Santos - a45842
Luís Santos - a30646
Nelson Fortuna - a45429
Tiago Barreiros - a46118

Orientador:
Professor Doutor Abel Gomes

09 de janeiro de 2023, Covilhã

Agradecimentos

Primeiramente, gostaríamos de agradecer ao nosso orientador Prof. Doutor Abel Gomes por ser sempre justo e fiel às suas ideias e aquando a proposta deste tema.

A sua orientação foi crucial no desenvolvimento deste projeto, bem como os métodos e as estratégias lecionadas em aula, a forma didática e prática como foram facultadas, foi muito importante para o nosso desenvolvimento profissional e pessoal, pois sentimos que melhorámos como pessoas e como grupo de trabalho.

Resumo

Este projeto visa criar um sistema solar virtual usando tecnologias tridimensionais. Tentámos criar uma experiência realista, imersiva e interativa do nosso sistema solar, permitindo que os utilizadores explorem e experimentem as maravilhas do universo. Adicionalmente, contará com conteúdo educacional sobre os diferentes planetas e outras características do sistema solar. Assim, esperamos proporcionar uma experiência única e emocionante para os professores e outros alunos.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Palavras-Chave

3D, Computação Gráfica, Modelos Geométricos, OpenGL, Sistema Solar.

Acrónimos

CG Computação Gráfica

EI Engenharia Informática

IDE *Integrated Development Environment*

UBI Universidade da Beira Interior

UC Unidade Curricular

Capítulo

1

Introdução

1.1 Âmbito e Enquadramento

O presente relatório foi elaborado por alunos do 3.° ano da Licenciatura de Engenharia Informática (EI) da Universidade da Beira Interior (UBI). Este foi desenvolvido no contexto da Unidade Curricular (UC) Computação Gráfica (CG) da UBI, através de pesquisas pessoais e material pedagógico providenciado pelo Professor da própria UC.

1.2 Motivação

A Computação Gráfica é uma área de estudo que permite a criação de imagens e animações de alta qualidade, tornando possível a representação de objetos e cenários de maneira realista e detalhada. O Sistema Solar é um tema fascinante e o fato de poder visualizá-lo em 3D amplia ainda mais a nossa compreensão e conhecimento sobre este conjunto de corpos celestes. Além disso, a criação de um sistema solar em 3D também pode ser utilizada como ferramenta educacional, possibilitando que as pessoas tenham uma representação mais clara e concreta de como o nosso sistema solar funciona. Portanto, a realização deste projeto de Computação Gráfica sobre o Sistema Solar em 3D é uma oportunidade única de explorarmos ainda mais este fascinante tema e de contribuirmos para o conhecimento e a compreensão das pessoas sobre o nosso sistema solar.

1.3 Objetivos

Neste projeto pretende-se implementar um sistema solar interativo tanto em 2D como em 3D. Para isso, iremos utilizar a linguagem C++, bem como as bibliotecas OpenGL, GLFW, GLAD e GLM. Características da aplicação gráfica:

- Modelar o sol, os planetas e os seus respetivos satélites (lua no caso da terra);
- Menus que permitam obter informação sobre os elementos do sistema solar (por exemplo, planetas);
- Texturizar os planetas para aumentar o realismo;
- A câmara pode mover-se em qualquer posição do sistema solar, sendo que este movimento afeta a luz que incide sobre os elementos do sistema solar;
- Utilização do teclado para fazer zoom ao sistema solar e para alterar a vista;
- Iluminação através de um foco de luz proveniente do sol;
- Calcular as sombras que os planetas e os satélites poderão originar entre si.

1.4 Abordagem

Para abordar o tema do Sistema Solar neste projeto de Computação Gráfica, propomos seguir os seguintes passos:

- Realizar uma pesquisa aprofundada sobre o Sistema Solar, incluindo informações sobre as características de cada planeta, suas orbitas e movimentos, entre outros detalhes;
- Escolher um software de Computação Gráfica que seja adequado para a criação de modelos 3D e animações;
- Começar a criação dos modelos dos planetas, utilizando as informações obtidas na pesquisa para garantira precisão dos mesmos;
- Aplicar texturas e materiais realistas nos modelos, de maneira a torná-los o mais parecidos possível com os corpos celestes reais;

- Implementar animações animações que mostrem o movimento dos planetas ao redor do Sol e as rotações de cada um deles;
- Testar o sistema solar em 3D criado para garantir sua qualidade e funcionamento;
- Redigir o relatório, apresentando os resultados da pesquisa e a metodologia utilizada para criar o sistema solar.

1.5 Organização do Documento

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

1. O primeiro capítulo – **Introdução** – refere o projeto que nos foi proposto, o seu âmbito e Enquadramento, a Motivação para a sua realização, os objetivos e a respetiva organização do documento.
2. O segundo capítulo – **Estado da Arte, Desenvolvimento e Aplicações** – descreve as ferramentas utilizadas na realização deste projeto;
3. O terceiro capítulo – **Ferramentas e Funcionalidades** – abordada o Histórico de funcionalidades do projeto;
4. O quarto capítulo – **Desenvolvimento** – apresenta as dependências que existem no programa e os detalhes de implementação;
5. O quinto capítulo – **Testes e Validação** – explicita todos os Testes e Aprimoramentos realizados ao programa;
6. O sexto capítulo – **Engenharia de Software** – explica como são estruturados os dados e o ciclo de vida da aplicação;
7. O sétimo capítulo – **Conclusões e Trabalho Futuro** – como o nome indica, apresenta uma breve conclusão.

Capítulo

2

Estado da Arte e Trabalhos Relacionados

2.1 Introdução

Este capítulo apresenta as aplicações e ferramentas usadas no desenvolvimento deste projeto, os seus detalhes de implementação e uma breve descrição da plataforma Visual Studio. O capítulo divide-se do seguinte modo:

- a secção 2.2, **Overleaf**, introdução à plataforma de escrita;
- a secção 2.3, **Trello**, breve descrição do *website*;
- a secção 2.4, **Meios de Comunicação**, apresentada em duas subsecções, subsecção 2.4.1 - **Discord** e subsecção 2.4.2 - ***Google Meet***;
- a secção 2.5, **GitHub**, caracteriza fielmente a plataforma;
- a secção 2.6, **Visual Studio**, define as caraterísticas da principal ferramenta usada ao longo do projeto;
- a secção 2.7, **Trabalhos Relacionados**, esclarece o trabalho fornecido pelo professor orientador.

2.2 Overleaf

O Overleaf é uma plataforma de colaboração *online* para escrita de documentos científicos, que inclui recursos para facilitar a criação de artigos, relatórios, apresentações e muito mais.

É especialmente útil para escritores científicos, mas igualmente para nós, estudantes, pois permite-nos trabalhar no mesmo documento, juntos, em tempo real, compartilhar ideias e receber *feedback*. Também oferece integração com várias bases de dados e repositórios de código, o que facilita a incorporação de Citações, Referências e *Linkagem*.

Foi essencial na escrita e embelezamento do Relatório Final, pois possui características que se destacam, entre as quais:

- Capacidade de auto-completar funções;
- Compila o trabalho várias vezes, com o objetivo de estar sempre atualizado e não se perder dados;
- Na eventualidade de existirem erros, indica os mesmos.

Figura 2.1: Representação do logótipo do Overleaf.

2.3 Trello

O Trello é uma ferramenta de gestão de projetos que permite que as equipas criem quadros virtuais para organizar e priorizar as tarefas. Os quadros são divididos em listas, e cada lista pode conter cartões, usados para representar tarefas individuais ou etapas do projeto. Os cartões podem ser arrastados e soltos entre as listas para indicar o progresso e as mudanças de prioridade.

Para o nosso projeto esta ferramenta teve uma presença crucial, permitiu-nos resolver problemas rapidamente e tomar decisões de forma mais eficiente. Em termos de Costumização, o Trello permite que as equipas personalizem os quadros e os cartões com etiquetas, lembretes e outros recursos para ajudar a gerir o projeto consoante as suas necessidades específicas.

Em suma, e não menos importante, esta ferramenta é gratuita, com opções de assinatura *premium* disponíveis para aqueles que precisam de recursos adicionais. Neste caso foi uma vantagem para nós porque podemos usufruir de uma aplicação tão revolucionária sem ter de pagar nada.

Podem consultar a nossa área de trabalho através do link: Gestão de Projeto.

Figura 2.2: Representação do logótipo do Trello.

2.4 Meios de Comunicação

2.4.1 Discord

O Discord é uma aplicação de mensagens e voz *online* que é bastante utilizado por grupos de amigos, equipas de trabalho e outras comunidades. O Discord permite que os utilizadores criem canais de texto e de voz para conversar e comunicarem uns com os outros. Este também oferece uma série de ferramentas de moderação e opções de privacidade para ajudar os administradores da comunidade a manterem o controlo sobre o conteúdo e o comportamento dos utilizadores. Além disso, o Discord tem uma ampla variedade de integrações com outras plataformas e serviços, como jogos, música e serviços de *stream*, o que o torna um recurso valioso para qualquer comunidade. Também permite algo que é muito importante que é a partilha de ecrã.

Esta ferramenta foi usada para comunicarmos tanto por voz, como por texto, também tem a vantagem de não ter limites de utilização, ou seja, podemos comunicar por voz o tempo que quisermos sem pagar nada por isso. O Discord oferece várias opções de organização, incluindo canais de texto e de voz, *tags* de cargo e grupos de utilizadores, o que permite que os membros da equipa de trabalho acessem facilmente as informações e tudo o que seja relevante para o projeto.

Figura 2.3: Representação do logótipo do Discord.

2.4.2 *Google Meet*

O *Google Meet* é uma plataforma de videochamadas *online* desenvolvida pela *Google*. Este permite que as pessoas se reúnam virtualmente por chamadas de voz, independentemente de onde estejam. É fácil de usar e oferece uma série de recursos para ajudar as pessoas a comunicarem entre si e colaborarem eficientemente durante as reuniões.

Permite ainda que as chamadas sejam gravadas e depois compartilhadas com os membros do grupo de trabalho, o que pode ser útil para a revisão posterior ou para pessoas que não puderam participar na reunião ao vivo terem acesso a tudo o que foi dito e combinado. Foi em alguns casos a alternativa ao *Discord*.

Figura 2.4: Representação do logótipo do *Google Meet*.

2.5 GitHub

O GitHub é um serviço de gestão de código-fonte e colaboração *online*. Serve para armazenar e gerir projetos de código aberto e privado.

Permite que os utilizadores criem repositórios de código, façam o seu controlo de versões, adicionem colaboradores aos projetos e façam o rastreamento de problemas e tarefas. Também fornece ferramentas para revisão de código, integração contínua e integração com outras ferramentas de desenvolvimento de *software*.

Além disso, o GitHub é popular para desenvolvedores compartilharem e descobrirem novos projetos de código aberto, uma plataforma comunitária para colaboração e desenvolvimento de *software* em equipa.

Figura 2.5: Representação do logótipo do *GitHub*.

2.6 Visual Studio

É um *Integrated Development Environment* (IDE) que oferece uma ampla variedade de recursos e ferramentas para ajudar os desenvolvedores a criar, depurar e publicar aplicativos de maneira rápida e eficiente.

Em suma, algumas das suas principais características incluem:

- Suporte a várias linguagens de programação;
- Depuração e teste;
- possui ferramentas de *design* para ajudar os desenvolvedores a criar *interfaces* atraentes e intuitivas.

Figura 2.6: Representação do logótipo do Visual Studio.

2.7 Trabalhos Relacionados

Este trabalho, está diretamente relacionado com o projeto fornecido pelo professor orientador.

Nome do Projeto: solarSystem

Autor: Yonghui Rao

Implementação:

1. Cada um dos objetos é desenhado como uma esfera usando as texturas de mosaico, com seu próprio raio e cor;

2. Os planetas giram em torno da estrela(sol), cada um tem o seu próprio caminho e a sua própria velocidade;

3. A barra de espaços pode ser usada para pausar e retomar a animação;

4. A cena é iluminada por uma fonte de luz localizada no centro da estrela. A própria estrela é iluminada por uma luz ambiente brilhante;

5. Quando o *mouse* está sobre um planeta, ele fica com um destaque brilhante.

ID planetas:

Branco $\Rightarrow 0(sol)$

Amarelo $\Rightarrow 1(Mercúrio)$

Rosa $\Rightarrow 2(Vénus)$

Azul $\Rightarrow 3(Terra)$

verde $\Rightarrow 4(Marte)$

vermelho $\Rightarrow 5(Júpiter)$

Teclas:

Espaço $\Rightarrow Stop$

Capítulo

3

Tecnologias e Ferramentas Utilizadas

3.1 Introdução

Neste capítulo vão ser descritas e enumeradas as várias ferramentas utilizadas no decorrer do nosso trabalho. Tendo sido essenciais no desenvolvimento do código que possibilitou a realização do nosso Sistema Solar.

3.2 OpenGL

Open Graphics Library (OpenGL) consiste numa interface de programação de aplicações (API) multiplataforma e multilingue para renderização de gráficos vetoriais 2D e 3D. A API é normalmente usada para interagir com uma unidade de processamento gráfico (GPU) para obter renderização acelerada por hardware.

OpenGL é amplamente utilizado em Computer Aided Design (CAD), realidade virtual, visualização científica e jogos virtuais, onde é particularmente útil para desenvolvedores de jogos, uma vez que beneficiam da renderização acelerada por hardware.

O OpenGL fornece aos desenvolvedores um conjunto de comandos para renderizar uma variedade de objetos gráficos, como pontos, linhas, triângulos e quadrados, bem como efeitos gráficos avançados, como texturas, iluminação e sombras. Ele também inclui suporte para shaders, que são pequenos programas que permitem aos desenvolvedores personalizar como seus objetos gráficos são renderizados.

O OpenGL torna mais fácil para os desenvolvedores criar gráficos 3D acelerados por hardware de alta qualidade de forma rápida e fácil, tornando-o uma escolha popular para desenvolvedores de jogos.

3.3 GLFW

A Graphics Library Framework (GLFW) trata-se de uma biblioteca multiplataforma gratuita, onde o código é open-source. A GLFW fornece uma API simples para criar e gerir janelas, contextos (OpenGL) e superfícies, recebendo inputs e eventos. A GLFW é escrita em C e suporta os sistemas operativos Windows, MacOS e Linux.

A GLFW pode ser usada em várias aplicações, como jogos, realidade virtual e muito mais.

3.4 GLM

A OpenGL Mathematics (GLM) é uma biblioteca matemática escrita em C++ e foi projetada para ser muito eficiente e fácil de usar para programação de gráficos 3D. A GLM fornece uma variedade de objetos e funções matemáticas que podem ser usadas para criar e manipular objetos tridimensionais. Inclui funções para operações de vetores e matrizes, entre muitas outras. A GLM foi projetada para ser compatível com a API OpenGL, portanto, pode ser usada em qualquer aplicação com base em OpenGL. A GLM é open-source e está disponível para ser utilizada por qualquer pessoa.

3.5 Glad

A Glad é uma biblioteca importante para os desenvolvedores, pois permite que eles usem o OpenGL em várias plataformas. Com a Glad, os desenvolvedores podem criar contextos OpenGL, gerir ponteiros de funções OpenGL, carregar funções OpenGL e gerir estados para OpenGL. Estes aspetos são muito úteis, visto a OpenGL ser uma biblioteca independente de plataforma, tornando-se mais fácil escrever código que funcione transversalmente nas diversas plataformas.

3.6 Assimp

A Open Asset Import Library (Assimp) é uma biblioteca com o fim de importar e exportar modelos 3D numa ampla variedade de formatos de arquivo. Ela suporta mais de 35 formatos de arquivo diferentes, incluindo formatos populares como FBX, OBJ, COLLADA e 3DS. Ela também oferece suporte a arquivos usados em videojogos e aplicações de realidade virtual. A Assimp fornece uma API com acesso aos dados no modelo e lida com o carregamento e a análise do arquivo, permitindo que os desenvolvedores acedam de forma rápida e fácil os dados de que precisam. A Assimp é escrito em C++ Portable, sendo multiplataforma pode ser utilizada em várias plataformas. A Assimp é open-source e de uso gratuito, o que a torna a escolha ideal para desenvolvedores que procuram uma maneira fácil de trabalhar com modelos 3D.

3.7 FreeType

FreeType consiste num software open-source desenvolvido por David Turner, Robert Wilhelm e Werner Lemberg. É uma biblioteca que permite que as aplicações renderizem texto em fontes bitmap e fontes vetoriais. O FreeType é usado por muitos sistemas operativos e aplicações para renderizar fontes na tela. Também é usado por editores de fontes, como FontForge e FontLab. O FreeType é escrito em C e foi portado para várias plataformas, incluindo Windows, Mac OS e Linux.

Capítulo

4

Etapas de Desenvolvimento

4.1 Introdução

No presente capítulo, vamos descrever as etapas de desenvolvimento de um programa em OpenGL para visualizar o sistema solar em 3D. Apresentaremos a divisão de tarefas entre os membros da equipe. Em seguida, apresentaremos as principais etapas do desenvolvimento do programa, mostrando de seguida imagens.

4.2 Divisão de Tarefas

Para este trabalho, foi feita a seguinte divisão de tarefas:

- Criação de Planetas - Nelson Fortuna e Luis Santos
- Criação de Saturno - Diogo Santos
- SkyBox - Diogo Santos
- Orbitas - Luis Santos
- FreeType (biblioteca de renderização de texto) - Tiago Barreiros
- Transição da Câmara - Tiago Barreiros

4.3 Procedimento Tomado

No desenvolvimento projeto foi necessário ter em atenção alguns procedimentos:

1. Inicialmente é necessário conhecer o básico das bibliotecas OpenGL e como cada uma delas funciona. Isso inclui como desenhar formas geométricas, como aplicar transformações geométricas e como iluminar o cenário;

2. Definir o tamanho e a escala do sistema solar. Isso inclui determinar o tamanho dos planetas em relação ao Sol e a distância entre eles;

3. Desenhar os planetas. Isso pode ser feito usando formas geométricas básicas, como esferas, ou usando modelos 3D pré-existentes (no caso de Saturno);

4. Adicionar texturas aos planetas para dar mais realismo. Isso pode ser feito usando imagens de textura, facultado no link presente no enunciado do projeto;

5. Adicionar a Lua ao planeta Terra. Isso envolve o mesmo processo usado para desenhar os planetas em si;

6. Adicionar movimento ao cenário, usando rotações e translações para simular o movimento dos planetas em torno do Sol e o movimento da Lua em torno da Terra;

7. Adicionar iluminação e sombras ao cenário. Isso pode ser feito usando luzes recorrendo ao *Phong shading*;

8. Permitir a câmara tenha várias posições e adicionar a ampliação e redução da mesma;

9. Adicionar algum texto informativo para tornar o cenário mais interativo;

10. Testar e depurar o sistema solar para garantir que ele funcione corretamente. Isso pode incluir ajustes na escala, na iluminação e nos movimentos dos planetas.

4.4 Imagens Finais

A seguir, estão as imagens do resultado final do nosso trabalho sobre o sistema solar em OpenGL.

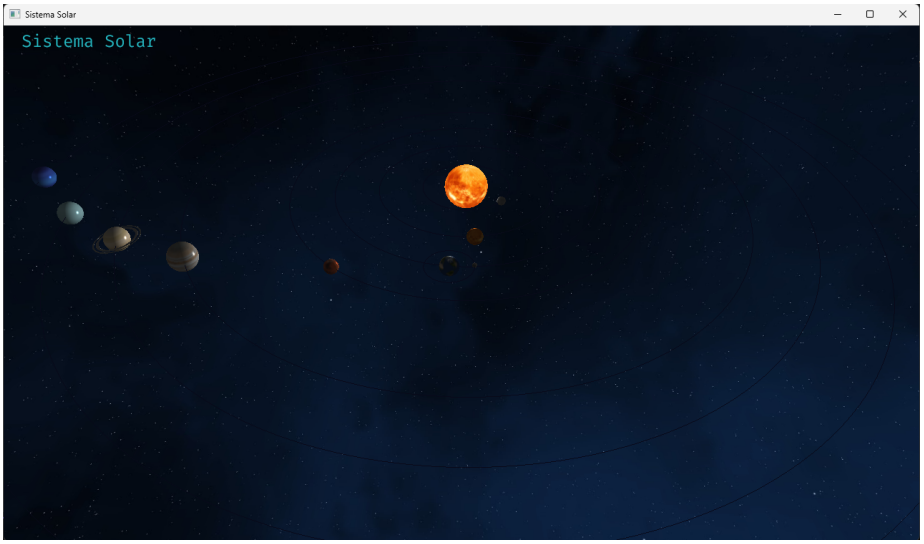

Figura 4.1: Sistema solar

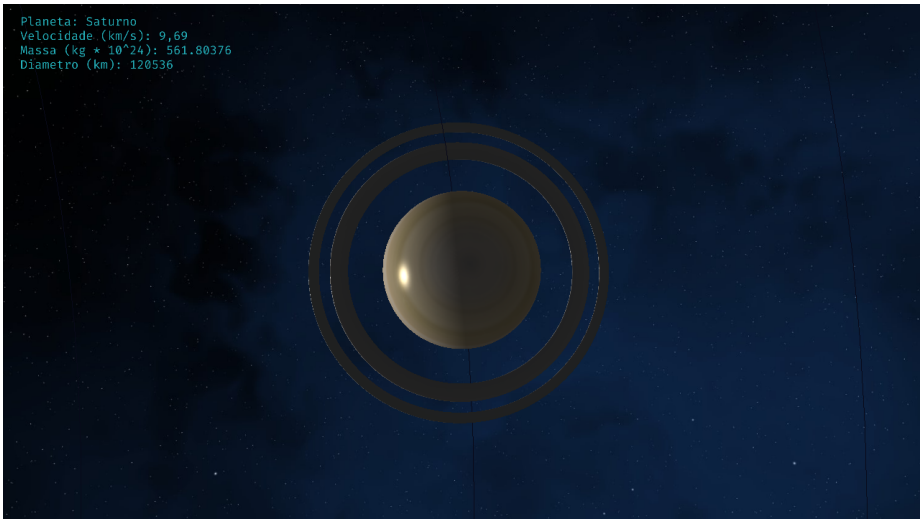

Figura 4.2: Saturno visto de cima

Capítulo

5

Desenvolvimento e Implementação

5.1 Introdução

Garantir a qualidade gráfica é importante, pois melhora a experiência do utilizador, evita problemas e erros, aumenta a segurança, a confiabilidade e maior retorno do investimento

Este capítulo representa, fielmente, o Ambiente desenvolvido. O capítulo divide-se do seguinte modo:

- a secção 5.2, **Dependências**, explica as dependências do projeto;
- a secção 5.3, **Detalhes de Implementação**, caracteriza e descreve cada ambiente gráfico desenvolvido;

5.2 Dependências

O programa apresenta algumas dependências como os *import´s* utilizados:

- `include <glad/glad.h>` - gerência funções de ponteiros para o OpenGL, neste caso mais usado para luz;
- `include <GLFW/glfw3.h>` - é usada para criar janelas, contextos e receber inputs, eventos;
- `include <glm/glm.hpp>` -biblioteca do OpenGL para as funções matemáticas;
- `include <iostream>` - implementa recursos de entrada/saída baseados em fluxo;

- `include <vector>` - é uma biblioteca que contem um tipo de array que pode mudar o seu tamanho, conforme a utilização do programa;
- `include <math.h>` - fornece funções e valores matemáticos.
- `include <stb_image.h>` - inclui as funções necessárias para ir buscar uma imagem e usar esta para as texturas;
- `include <ft2build.h>` - é através desta biblioteca que fazemos a escrita de palavra no ecrã;
- `include <shader_m.h>` - usado para fazer a leitura dos ficheiros shaders, criando assim um objeto Shader;
- `include <camera.h>` - fornece um objeto Camera, o qual contém já as funcionalidades da câmara andar pelo espaço;
- `include <model.h>` - é usado para fazer a leitura dos ficheiros .obj, neste caso só é usado no planeta Saturno.

5.3 Detalhes de Implementação

5.3.1 Planetas

Para criar os planetas, foi utilizada uma esfera como base para quase todos os planetas. Em seguida, as texturas dos planetas foram carregadas em variáveis. Quando foi necessário criar um planeta específico, a esfera base foi obtida e a textura do planeta correspondente foi aplicada. Depois, foi criada uma matriz de modelo para o planeta. A posição do planeta foi calculada e a matriz de modelo foi transladada para essa posição. O planeta foi então rotacionado em torno de seu próprio eixo (eixo Y) e a escala da esfera base foi ajustada para se adequar ao tamanho do planeta. Por fim, a nova matriz de modelo foi passada para os *shader* para que o planeta fosse desenhado na tela.

Para criar os satélites,no caso da lua, foi utilizado o modelo da Terra como base. Em seguida, foi aplicada uma translação ao redor da Terra para posicionar o satélite em sua órbita.

Saturno foi o único planeta criado de forma diferente, pois em vez de utilizar uma esfera como base foi usado um arquivo .obj com as características de Saturno, isto é com os anéis.. O procedimento para criar Saturno foi, em essência, o mesmo usado para criar os outros planetas, com a diferença de que foi utilizado um modelo diferente como base e a textura de Saturno foi

aplicada. A matriz de modelo foi então transformada como necessário para posicionar Saturno em sua órbita e para fazê-lo girar em torno de seu próprio eixo, e a matriz resultante foi passada para os *shader* para que Saturno fosse desenhado na tela.

```
GLuint vao = SolidSphere(planet_raio, 32, 32);
unsigned int terra_tex = loadTexture(terra_path);
// ...
// ######## Terra
//Aqui especificamos que VAO queremos que esteja ativo para o
    render do respetivo planeta
glBindVertexArray(vao);

//especificamos que textura queremos que fique ativa
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, terra_tex);

//Shader ativado para o desenho da esfera
sphereShader.use();

glm::mat4 model = glm::mat4(1.0f);
glm::mat4 moon;
double xx = sin(glfwGetTime() * plan_velocidade * 0.75f) *
    planet_raio * 3.0f * 2.0f;
double zz = cos(glfwGetTime() * plan_velocidade * 0.75f) *
    planet_raio * 3.0f * 2.0f;
posicaoPlanetas[2] = glm::vec3(xx, 0, zz);

model = glm::translate(model, glm::vec3(xx, 0.0f, zz));
moon = model;
model = glm::rotate(model, (float)glfwGetTime() + 2, glm::vec3
    (0.0f, 3.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.4, 0.4, 0.4));
sphereShader.setMat4("model", model);

glDrawElements(GL_TRIANGLES, numIndicies, GL_UNSIGNED_INT,
    BUFFER_OFFSET(0));
```

Excerto de Código 5.1: Excerto de codigo do planeta terra

5.3.2 *SkyBox*

Para criar uma *skybox* em OpenGL, primeiramente criamos um cubo gigante e carregar as imagens para as texturas das faces do cubo, depois criamos uma estrutura de dados para armazenar as informações da *skybox*, fazer o upload dos vértices do cubo para o VBO de seguida fundamos uma estrutura de dados para armazenar a textura da *skybox* e fazer o upload das imagens

para a textura da *skybox*, criamos um programa de *shader* específico para a *skybox*, desenhamos o cubo usando o programa de *shader* da *skybox* e a textura da *skybox*, e atualizamos a transformação de visualização e desenhar o cubo novamente quando o observador se move para criar a ilusão de um espaço infinito.

```
// vertices do cubo
  float skyboxVertices[] = {...};
  // skybox buffers
    unsigned int skyboxVAO, skyboxVBO;
    glGenVertexArrays(1, &skyboxVAO);
    glGenBuffers(1, &skyboxVBO);
    glBindVertexArray(skyboxVAO);
    glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &
        skyboxVertices, GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (
        void*)0);
// ... Load Texturas
    vector <string> faces{...};
  // Load do cubemap para colocar as texturas
  unsigned int cubemapTexture = loadCubemap(faces);
    //ativamos aqui o shader que vamos utilizar
    skyboxShader.use();
    skyboxShader.setInt("skybox", 0);
// ...
     glBindVertexArray(skyboxVAO);
        glDepthFunc(GL_LEQUAL);
        skyboxShader.use();
        view = glm::mat4(glm::mat3(camera.GetViewMatrix()));
        skyboxShader.setMat4("view", view);
        skyboxShader.setMat4("projection", projection);
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
        glDrawArrays(GL_TRIANGLES, 0, 36);
        glBindVertexArray(0);
        glDepthFunc(GL_LESS); // set depth function back to default
```

Excerto de Código 5.2: Excerto de código da criação da skybox

5.3.3 Orbitas

Primeiramente, criamos o vetor com os valores das órbitas que serão armazenados no *buffer*. Depois, durante o laço `while`, buscamos os valores armazenados no *buffer*, percorremos os 8 planetas e fazemos a translação da órbita para a posição do planeta. Em seguida, enviamos os dados para os *shaders* e

desenhamos na tela. Depois, para a órbita da Lua, fazemos a translação para que a Terra seja o centro da órbita e enviamos os dados para os *shaders* para desenhar na tela.

```
// ... Desenhar no ecr
     glBindVertexArray(VAO_t);
glm::mat4 modelorb = glm::mat4(1.0f);
    for (int i = 1; i < 9; i++)
    {
        modelorb = glm::mat4(1);
  // iniciar no meio para fazer o circulo conforme o meio
  modelorb = glm::translate(modelorb, posicao_centro);
        if (i <= 4) {
            modelorb = glm::scale(modelorb, glm::vec3(i * 2.f, i *
                2.f, i * 2.f));
        }
        else {
            modelorb = glm::scale(modelorb, glm::vec3(i * 2.5f, i *
                2.5f, i * 2.5f));
        }
        sphereShader.setMat4("model", modelorb);
        glDrawArrays(GL_LINE_LOOP, 0, (GLsizei)orbVert.size() / 3);
    }
    modelorb = glm::mat4(1.0f);
    modelorb = glm::translate(modelorb, posicaoPlanetas[2]);
    modelorb = glm::scale(modelorb, glm::vec3(0.5f * 2.f, 0, 0.5f *
        2.f));
    sphereShader.setMat4("model", modelorb);
    glDrawArrays(GL_LINE_LOOP, 0, (GLsizei)orbVert.size() / 3);
```

Excerto de Código 5.3: Excerto de código das orbitas

5.3.4 Letras

Utilizamos a biblioteca `FreeType` para escrever informações na tela. Primeiro, carregamos a fonte e os 128 primeiros caracteres da tabela ASCII. Depois, geramos uma textura para cada carácter e armazenamos os dados relevantes em uma estrutura de caracteres adicionada ao mapa de caracteres. Usamos uma função para renderizar cada carácter, calcular as dimensões do quadrado com a métrica do carácter e gerar um conjunto de 6 vértices para atualizar o VBO. Finalmente, escrevemos a string na tela com as texturas de cada carácter.

```
 void renderText(Shader &s, std::string text, GLfloat x, GLfloat y,
     GLfloat scale, glm::vec3 color)
{
  s.use();
    glUniform3f(glGetUniformLocation(s.ID, "textColor"), color.x, color.
        y, color.z);
    glActiveTexture(GL_TEXTURE0);
    glBindVertexArray(VAO);
    std::string::const_iterator c;
    for (c = text.begin(); c != text.end(); c++)    {
        Character ch = Characters[*c];
        GLfloat xpos = x + ch.Bearing.x * scale;
        GLfloat ypos = y - (ch.Size.y - ch.Bearing.y) * scale;
        GLfloat w = ch.Size.x * scale;
        GLfloat h = ch.Size.y * scale;
        GLfloat vertices[6][4] = {
            { xpos,     ypos + h,   0.0, 0.0 },
            { xpos,     ypos,       0.0, 1.0 },
            { xpos + w, ypos,       1.0, 1.0 },

            { xpos,     ypos + h,   0.0, 0.0 },
            { xpos + w, ypos,       1.0, 1.0 },
            { xpos + w, ypos + h,   1.0, 0.0 }
        };
        glBindTexture(GL_TEXTURE_2D, ch.TextureID);
        glBindBuffer(GL_ARRAY_BUFFER, VBO);
        glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(vertices), vertices);
        glBindBuffer(GL_ARRAY_BUFFER, 0);
        glDrawArrays(GL_TRIANGLES, 0, 6);
        x += (ch.Advance >> 6) * scale; }
    glBindVertexArray(0);
    glBindTexture(GL_TEXTURE_2D, 0);}
```

Excerto de Código 5.4: Excerto de código da renderização da string

5.3.5 Transições de Câmara

Para facilitar as implementações da câmara, usamos um objeto da biblioteca `camera.h`, utilizada anteriormente nos trabalhos práticos. A câmara tem as seguintes funcionalidades:

- *Zoom in*/ *Zoom out* com teclado ou *scroll* do rato;
- Movimentação no espaço;
- Fixação em cada planeta;

```
    void processInput(GLFWwindow* window)
{
    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);
    // zoom in com o z e zoom out com o x
    if (glfwGetKey(window, GLFW_KEY_X) == GLFW_PRESS)
    {float Zoom = camera.Zoom;
    if (Zoom > 45.0f)
      camera.Zoom = 45.0f;
        else
      camera.Zoom += 0.2f;
    }if (glfwGetKey(window, GLFW_KEY_Z) == GLFW_PRESS){
    float Zoom = camera.Zoom;
        if (Zoom < 1.0f)
      camera.Zoom = 1.0f;
    else
      camera.Zoom -= 0.2f;
    }if (glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS)
  {
    planeta = 1;
        Info.Nome = "Mercurio";
        Info.Velocidade = "47,87";
        Info.Massa = "0.32868";
        Info.Diametro = "4880";
  }}
```

Excerto de Código 5.5: Excerto de código usado para processar inputs

5.4 Conclusão

Durante a implementação dos planetas, um dos principais desafios foi adicionar as bibliotecas extras, como a `FreeType` e o modelo `.obj` de Saturno, pois elas exigiram um conhecimento mais aprofundado da linguagem de programação e da estrutura do código. Além disso, a adição de letras e de objeto para colocar a textura de Saturno requereu um esforço extra para entender o funcionamento dessas ferramentas e integrá-las ao programa de forma coesa. No entanto, esses esforços foram recompensados com o sucesso da implementação, que foi capaz de gerar planetas e satélites de forma realista e interativa na tela.

Capítulo

6

Conclusões e Trabalho Futuro

6.1 Conclusões Principais

Com a realização deste projeto foi possível criar uma representação em 3D do Sistema Solar de forma realista, detalhada e educativa. Através da utilização de técnicas de Computação Gráfica e da realização de uma pesquisa aprofundada sobre o Sistema Solar, foi possível criar modelos 3D precisos dos planetas, da lua e do Sol. Além disso, a adição de texturas e materiais realistas e a implementação de animações que mostram o movimento dos planetas contribuíram para tornar a representação ainda mais realista e detalhada. Os recursos interativos, como as informações sobre cada elemento do sistema solar, também foram importantes para tornar o sistema solar uma ferramenta educacional eficaz. Em resumo, o sistema solar criado neste projeto é uma excelente ferramenta para aumentar o conhecimento e a compreensão das pessoas sobre o nosso sistema solar.

6.2 Trabalho Futuro

É importante lembrar que o conhecimento sobre o Sistema Solar está sempre evoluindo e novas descobertas são feitas regularmente. Portanto, é possível que haja ainda mais detalhes e informações que possam ser adicionados à representação criada neste projeto. Além disso, existem outras áreas relacionadas com a Computação Gráfica que podem ser exploradas no futuro, como a criação de visualizações em realidade virtual ou aumentada, ou a utilização de técnicas de *machine learning* para gerar imagens mais realistas automaticamente. Em resumo, há muitas possibilidades de concretização de projetos futuros relacionadas com o tema. É importante continuar a pesquisar e a

explorar novas técnicas e tecnologias para poder criar representações ainda mais detalhadas e realistas do nosso sistema solar.

Bibliografia

[1] Scikit-learn
https://learnopengl.com/

[2] Medium
https://medium.com/@keynekassapa13/creating-the-solar-system-opengl-and-c

[3] OpenGL
https://www.opengl.org/

[4] w3schools
https://www.w3schools.com/cpp/default.asp

[5] Abel Gomes
https://www.di.ubi.pt/~agomes/cg/