Sistema Solar

Relatório de Projeto da Unidade Curricular
Computação Gráfica

2 de Junho de 2019

Edgar Daniel Nº35329
Pedro Batista Nº35746
Raquel Guerra Nº37484

Sinopse

Neste documento está descrito o projeto sobre o Sistema Solar realizado no âmbito da cadeira de Computação Gráfica, da licenciatura de Engenharia Informática da Universidade da Beira Interior. Neste projeto foi desenvolvido um Sistema Solar usando OpenGl.

Edgar Daniel Nº35329
Pedro Batista Nº35746
Raquel Guerra Nº37484
Covilhã, Portugal
2 de Junho de 2019

Conteúdo

Lista de Figuras

Acrónimos

GLAD *GL/GLES/EGL/GLX/WGL Loader-Generator*

GLFW *Graphics Library Framework*

GLM *OpenGL Mathematics*

NASA *The National Aeronautics and Space Administration*

OpenGL *Open Graphics Library*

Capítulo 1

Motivação

1.1 Enquadramento

Este projeto está a ser realizado no âmbito da cadeira de Computação Gráfica (UC 11569) de maneira a aprofundar os conhecimentos das bibliotecas *Open Graphics Library* (OpenGL), *Graphics Library Framework* (GLFW), *GL/GLES/EGL/GLX/WGL Loader-Generator* (GLAD) e *OpenGL Mathematics* (GLM).

1.2 Objetivos

Este projeto consiste na elaboração de uma representação do sistema solar. Para efetuar o mesmo iremos usar as bibliotecas referidas em 1.1. Esta representação deverá incluir as seguintes características:

- Modelar o sol, os planetas e os seus respetivos satélites (lua no caso da terra);
- Texturizar os planetas par aumentar o realismo;
- Menus que permitam obter informação sobre os elementos do sistema solar (por exemplo, planetas);
- Iluminação através de um foco de luz proveniente do sol;
- Calcular sombras que os planetas e os satélites poderão originar entre si;
- A câmara pode mover-se para 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.

Capítulo 2

Tecnologias Utilizadas

2.1 Introdução

Neste capitulo iremos descrever as tecnologias utilizadas de maneira a obter os objectivos propostos.

2.2 Tecnologias

Neste trabalho utilizamos o OpenGL 4.3 caso estejamos a correr numa plataforma não apple e 4.1 se estivermos a correr numa plataforma Apple. A linguagem utilizada para desenvolver em OpenGl é C++, utilizando as bibliotecas *GL/GLES/EGL/-GLX/WGL Loader-Generator* (GLAD),*Irrklang*, *Open Graphics Library* (OpenGL), *Graphics Library Framework* (GLFW) e *OpenGL Mathematics* (GLM).

Adicionalmente utilizámos o código fornecido pelo [5], e as bibliotecas respetivas.

Utilizamos também o IDE *Visual Studio 2019* e o software *blender* para a criação das esferas.

Capítulo 3

Etapas de Desenvolvimento

No inicio deste projeto tentamos renderizar um objeto criado na aplicação (Blender), este processo foi realizado com sucesso mas de seguida tivemos algumas dificuldades com a aplicação das texturas no objeto. Para a resolução deste desafio foi estipulada uma pesquisa para um exemplo prático onde já existisse objetos esféricos implementados e que poderiam ser úteis. Conseguimos encontrar um repositório, que está referenciado na nossa bibliografia [2], que ajudou com a reestruturação da nossa aplicação.

Depois desta fase quisermos replicar a esfera para criar os vários objetos do sistema solar, sendo cada um deles composto por texturas diferentes. Para as texturas fizemos o download das imagens que estavam presentes no recurso fornecido pelo professor para que assim fosse possível dar um aspeto mais realista e atrativo ao nosso sistema solar. Para a distancia dos planetas realizamos o cálculo de uma trajetória circular para que se pudessem movimentar em torno do sol.

Para um aspeto mais realista foram definidos os materiais para cada um dos objetos em conjunto com os coeficientes de refletividade.

Como ponto de luz tentamos implementar, sem sucesso, um ponto fixo com o objetivo de iluminar toda a cena.

Uma vez que já tínhamos todos os objetos definidos e com devidas posições e trajetórias, decoramos a cena com ajuda de uma skybox, ou seja, implementação de uma esfera com textura cintilante.

Para criar uma interação mais dinâmica com o utilizador foram incorporados funcionalidades adjacentes com ajuda de controlos para que a câmara se pudesse mover para qualquer ponto do sistema solar.

De seguida foi implementado um modo de exibição 2D, de maneira a dar uma perspetiva *top-down* do sistema solar.

Como movimento rotacional, foram implementadas funções tais como *glm::rotate*.

O modo menu foi desenhado com o intuito de dar informação detalhada relativa ao planeta ou sol selecionado pelo utilizador.

Como funcionalidade extra incorporamos uma cintura de asteróides que circula entre os planetas Marte e Júpiter.

Capítulo 4

Descrição do funcionamento do Software

4.1 Renderização dos Objetos

Para renderizarmos os planetas e asteróides começamos por desenhar um modelo no *blender*, lá criamos uma esfera , como a representada em 4.1, que depois exportamos como um objeto e importámos usando uma biblioteca do [5].

```
//loading do objeto (planeta em geral)
venus = ObjMesh::load("../media/objects/venus.obj", true);

//porcao do codigo aproveitado do CookBook
std::unique_ptr<ObjMesh> mesh(new ObjMesh());

    ObjMeshData meshData;
    meshData.load(fileName, mesh->bbox);
    // Generate normals
    meshData.generateNormalsIfNeeded();
    // Generate tangents?
    if( genTangents ) meshData.generateTangents();
    // Convert to GL format
    GlMeshData glMesh;
    meshData.toGlMesh(glMesh);
    if( center ) glMesh.center(mesh->bbox);
    // Load into VAO
    mesh->initBuffers(
            &(glMesh.faces), &glMesh.points, &glMesh.normals,
            glMesh.texCoords.empty() ? nullptr : (&glMesh.texCoords),
            glMesh.tangents.empty() ? nullptr : (&glMesh.tangents)
    );
    cout << "Loaded mesh from: " << fileName
         << " vertices = " << (glMesh.points.size() / 3)
         << " triangles = " << (glMesh.faces.size() / 3) << endl;
    return mesh;
```

Excerto de Código 4.1: *Load* de um objeto

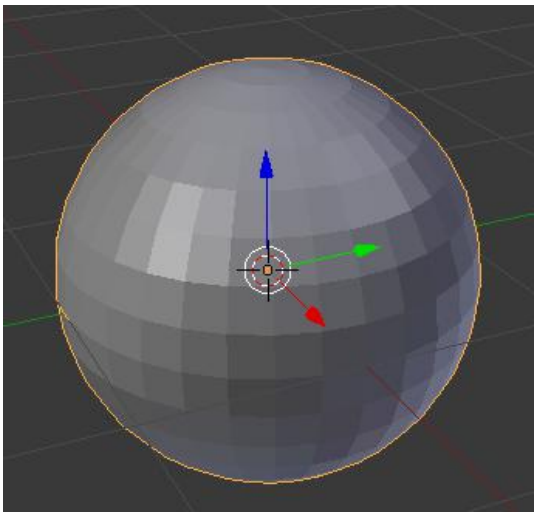

Figura 4.1: Esfera criada no blender

Devido a problemas no uso de texturas nos objetos exportados, maioritariamente devido à falta de prática dos membros com o software mencionado, decidimos que o melhor seria procurar objetos pré-feitos online, estes encontrados em [2].

Na totalidade do projecto usamos apenas três objetos, um para os planetas, luas e estrelas gerais, outro para Saturno (devido aos anéis) e finalmente um para os asteróides, com uma superfície com ar mais rochoso.

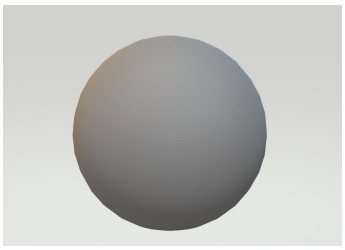

Figura 4.2: Esfera Planeta

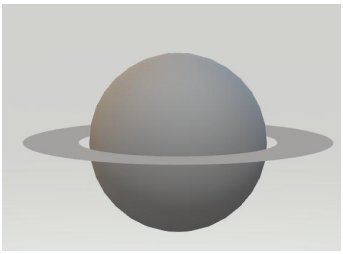

Figura 4.3: Esfera Saturno

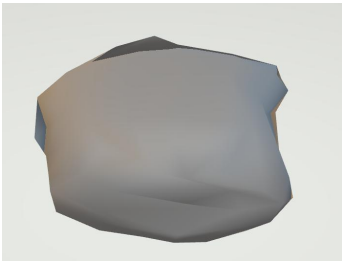

Figura 4.4: Objeto asteróide

O 4.2 é usado para todos os planetas (à excepção de Saturno), para o Sol e para a Lua, o 4.3 é usado para Saturno e o 4.4 é usado para os trinta e dois asteróides presentes na cintura.

4.2 Replicação e posicionamento dos Objetos

Para posicionar os objetos na sua localização inicial, criamos uma estrutura de dados, e dividimos os objetos em cinco categorias.

- Os planetas;
- O sol;
- Os asteróides;
- As luas ou satélites naturais, que apesar de estar preparado para várias, apenas temos uma;
- O *background*.

Cada uma destas categorias contém uma estrutura associada, dependente das necessidades de cada uma delas, com isto em mente, temos então uma estrutura similar entre os planetas, os asteróides e as luas. Seguidamente temos o sol com uma estrutura mais pequena pois não necessita de movimento de translação, e finalizando temos o *background* com a estrutura mais pequena das cinco, pois não necessita de rotação. O presente nas estruturas mais pequenas encontra-se também presente nas maiores.

Com isto em mente, comecemos por explicitar os elementos da estrutura representativos da localização:

- A *Location*, esta que é usada para determinar a localização inicial do centro, este presente em todas as categorias;
- O *radius*, esta usada para calcular o quanto o objeto inicial vai escalar, de maneira a ter um tamanho mais apropriado ao restante sistema solar, no caso do *background*, irá escalar para valores bastante elevados, para que a câmara esteja presente no interior do objeto, este também presente em todas as categorias;
- A *Default Location*, esta usada para retornar os planetas e as luas à sua posição original, útil no uso do menu, mais explicitado em 4.12 .

```
//variavel responsavel pela localizacao (background)
float SceneTexture::bkLocation[1][3]{ {0.0f, 0.0f , 0.0f} };
//variavel responsavel pelo scale (background)
float SceneTexture::bkRadius = 150.0f;
//variavel responsavel pela localizacao default (planetas)
float SceneTexture::planetDefaultlocations[8][3] = {...}
```

Excerto de Código 4.2: Estrutura de Localizacao e Posicionamento

4.3 Movimento de Translação

Como referido em 4.2, temos categorias com uma estrutura correspondente, essas estruturas contêm variáveis de apoio extra para calcular o movimento de translação dos planetas, dos asteróides e das luas, sendo essas as seguintes.

- A **velocidade**, utilizada para calcular a velocidade de translação de um planeta;
- O **ângulo**, usado para calcular o angulo de translação a que os objetos se encontram do centro, podendo calcular assim a rota circular à volta do sol.
- O **centro de translação**, usado pela categoria das luas para saber qual o planeta à volta do qual roda.

```
  //caso o centro de translacao nao seja o sol
moonTransCenter[0][0] = planetlocations[2][0];
moonTransCenter[0][1] = planetlocations[2][1];
moonTransCenter[0][2] = planetlocations[2][2];

  //definicao da velocidade de translacao
moonAngle[0] += moonSpeed[0];
  //caso tenha completado a translacao
while (moonAngle[0] > 360.0)
  moonAngle[0] -= 360.0;
  //calculo de um movimento circular, atualizando a posicao atual
float tempAngle = (moonAngle[0] / 180.0f) * 3.14159f;
moonlocations[0][0] = moonTransCenter[0][0] + sin(tempAngle) *
    moonDistance[0];
moonlocations[0][2] = moonTransCenter[0][2] + cos(tempAngle) *
    moonDistance[0];
```

Excerto de Código 4.3: Cálculo do movimento de translação

Utilizando estas variáveis auxiliares, conseguimos facilmente calcular o movimento de translação dos planetas com a função demonstrada em 4.3, tendo assim algo semelhante a 4.5 em aparência.

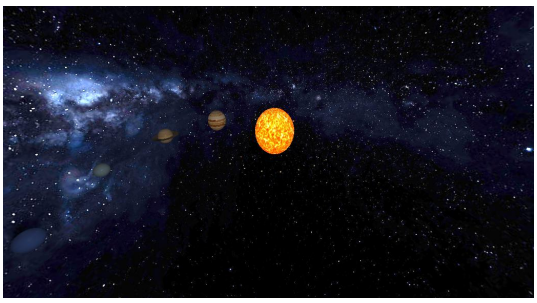

Figura 4.5: Sistema Solar *Overview*

4.4 Aplicação de Texturas

Para aplicarmos as texturas, começamos por escolher as texturas mais adequadas para o nosso programa, começamos por ir ao site oficial da *The National Aeronautics and Space Administration* (NASA), mas considerámos que as texturas fornecidas eram demasiado pesadas para o nosso programa, tornando-o mais lento desnecessariamente, por isso todas as texturas à excepção da textura utilizada na *skybox* foram retiradas do [2], sendo a da *skybox* retirada de [4].

Para as carregarmos e aplicarmos nos nossos objetos, usamos uma versão adaptada do código fornecido em [5], adaptando de maneira a tornar o carregamento mais leve, e tornando mais fléxivel a mudança de textura.

```
    //Carregamento de uma textura (Sol)
    texSun = Texture::loadPixels("../media/texturas/2k_sun.jpg", sSun,
        tSun);
    //Executado na inicializacao
  glGenTextures(1, &textL);

  //Load Texture
  glBindTexture(GL_TEXTURE_2D, textL);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height,0, GL_RGBA,
        GL_UNSIGNED_BYTE, data);

    //Aplicacao da textura antes da renderizacao do objeto
    textureLoad(texUranus, sUranus, tUranus);
  venus->render();
```

Excerto de Código 4.4: Carregamento e aplicação das texturas

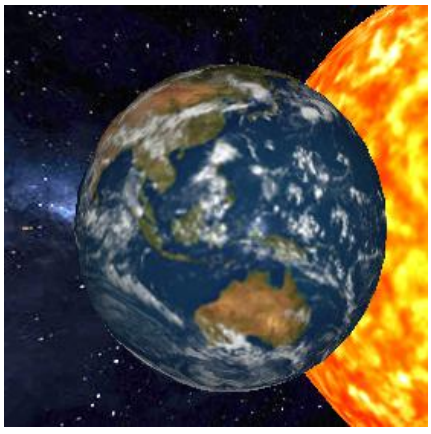

Figura 4.6: Terra Texturizada

Temos assim presente o aspecto de um planeta texturizado usando estes métodos 4.6.

4.5 Definição da Reflectividade dos Objetos

Tal como referido em 4.2 temos várias variáveis auxiliares para certos efeitos, para a reflexão do material temos mais uma.

- A *luz*, uma matriz de x por nove, sendo x o número de elementos pertencentes à categoria, em linha da matriz, as três primeiras colunas representam a refletividade da luz difusa do objeto, as próximas três a refletividade da luz especular e as restantes representam a refletividade da luz de ambiente.

Utilizamos estes valores para dar uma reflexão apropriada à distância a que cada objeto se encontra do Sol, dando um ar mais realista ao Sistema Solar, como podemos observar na figura 4.7.

```
    //Envio dos valores de reflexao para o shader
  prog.setUniform("Material.Kd", sLi[0], sLi[1], sLi[2]);
  prog.setUniform("Material.Ks", sLi[3], sLi[4], sLi[5]);
  prog.setUniform("Material.Ka", sLi[6], sLi[7], sLi[8]);
  prog.setUniform("Material.Shininess", 50.0f);
```

Excerto de Código 4.5: Envio dos valores de reflexão para o shader

Figura 4.7: Reflexão

4.6 Ponto de Luz

Temos presente um ponto de luz, este que irá emitir sempre a mesma intensidade de luz, tirando para a *skybox* (esta difere para dar um ar mais vivido ao *background*.

```
    //Envio dos valores da luz para o shader
  prog.setUniform("Light.Position", glm::vec4(.0f, .0f, .0f, 1.0f));
  prog.setUniform("Light.Intensity", glm::vec3(1.f, 1.f, 1.f));
```

Excerto de Código 4.6: Definição do ponto de luz e intensidade da mesma

4.7 *Skybox*

De maneira a fazermos um background mais convincente e mais realista do que um simples fundo preto, decidimos criar uma *skybox*, esta consiste de uma esfera de proporções bastante mais elevadas do que os restantes objetos, com a câmara posicionada no interior, texturizada com uma textura representativa do horizonte do espaço.

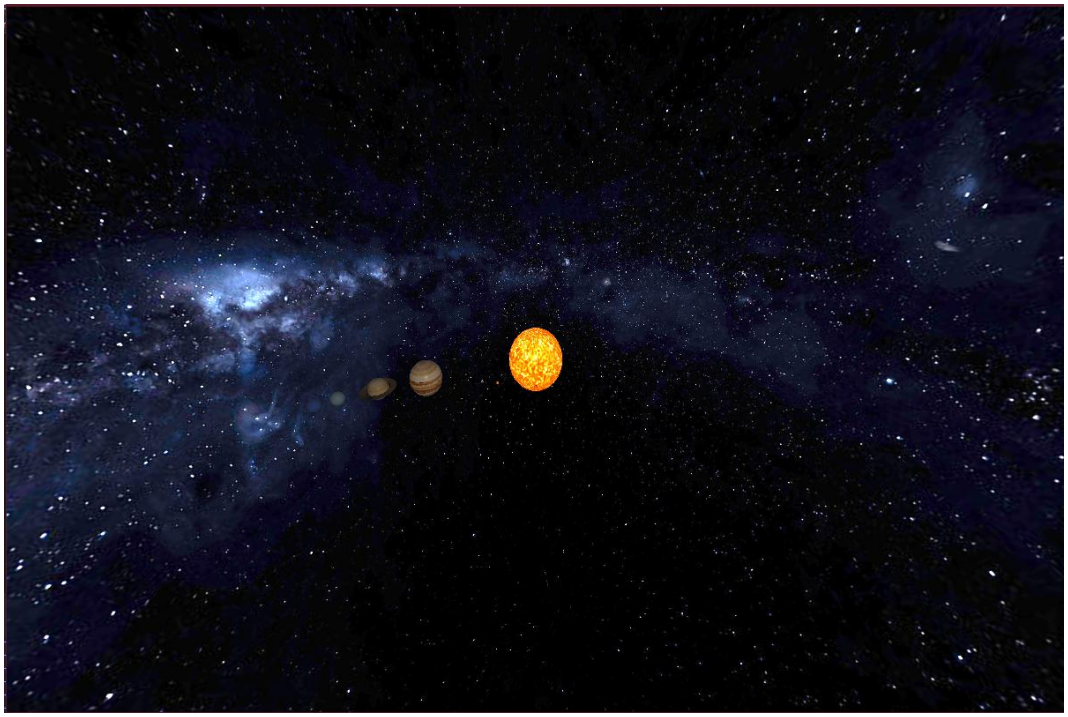

Figura 4.8: *skybox*

4.8 Câmara

A nossa projecção encontra-se em 16:9, com um *field of view* bastante elevado, desta maneira conseguimos ter uma visão bastante abrangente do Sistema Solar, a câmara é movível até certos limites através dos controlos especificados em 4.9.

```
//Definicao da Projecao
projection = glm::perspective(glm::radians(100.0f), 16.0f / 9.0f,
    0.1f, 10000.0f);
```

Excerto de Código 4.7: Definição da Projeção

Tirando o modo de menu, o utilizador encontra-se sempre a olhar para o Sol, tendo assim sempre uma visão centrada do Sistema Solar, como observado por exemplo em 4.8, podemos fazer *zoom* em todos os eixos, até um certo limite, para evitar

sair da *skybox* tendo assim uma certa liberdade de movimento. No modo menu a câmara encontra-se apontada para o planeta seleccionado.

4.9 Controlos

Temos vários controlos, sendo eles, fazer *zoom*, pausar o movimento e rotação dos planetas, entrar no modo de 2D ou de menu, sair dos mesmos, selecção do planeta dentro do modo de menu (4.9), para mostrar todos os comandos disponíveis no momento e ativação de um *easter egg*.

Estes controlos foram conseguidos atribuindo funções a botões variados, sendo estes:

- W e S , usados para fazer zoom no eixo dos x (limitado entre 500 e 2900);
- A e D , usados para fazer zoom no eixo dos y (limitado entre -1800 e 1800);
- Q e E , usados para fazer zoom no eixo dos z (limitado entre -1800 e 1800);
- M e N , usados para entrar e sair do modo de menu respectivamente;
- V , para entrar e sair do modo 2d;
- Setas direita e esquerda, usada para mudar de planeta no modo de menu;
- H , usado para apresentar os comandos disponíveis na linha de comandos;
- CapsLock , usado para ativar o easter egg.

```
      if (key == GLFW_KEY_W && action == GLFW_REPEAT) {
    if (camx > 500) {
      camx -= 100.0f;
    }
  }
```

Excerto de Código 4.8: Exemplo de atribuição de um comando

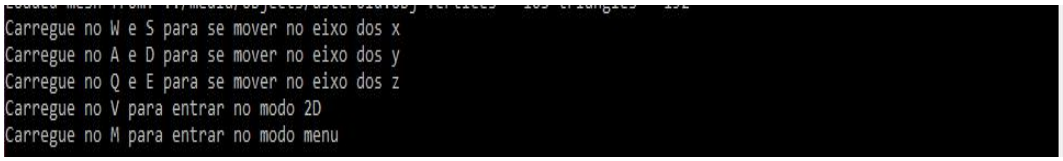

Figura 4.9: Comandos

4.10 Modo 2D

Temos implementado no nosso sistema solar uma visão 2d, ao ativarmos este iremos mudar para uma visão *top down* (4.10) do sistema solar, podendo ver assim de uma maneira mais geral a atividade do mesmo.

Tal como descrito em 4.9, este menu pode ser ativado e desativado usando a tecla "V", usamos uma variável de controlo para saber se o modo se encontra ativado ou não.

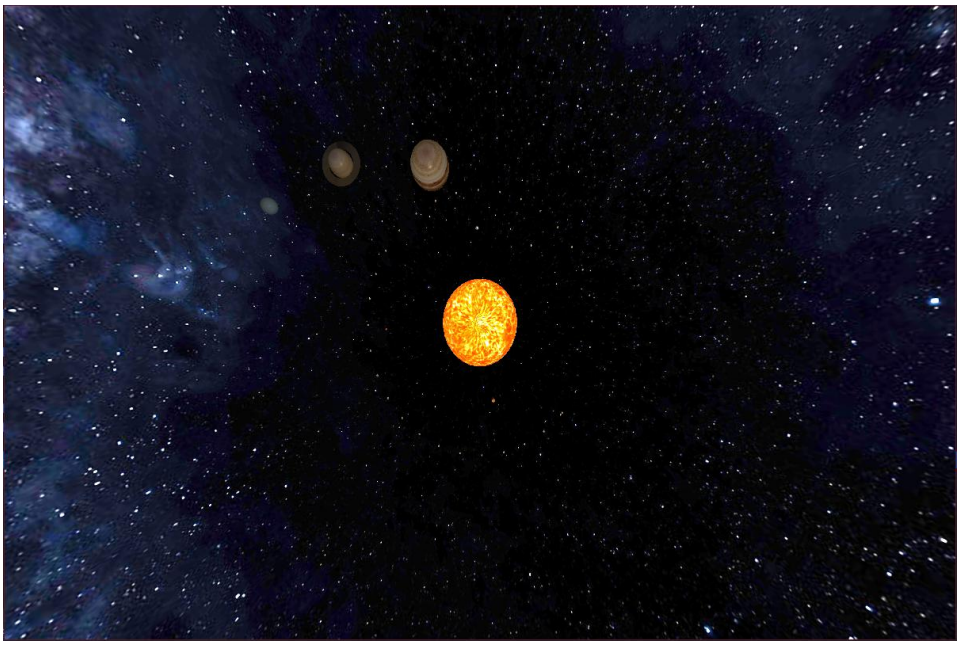

Figura 4.10: Visão 2D

4.11 Movimento rotacional

Os planetas, estrelas e luas além do movimento de translação à volta dos respetivos centros, também executam um movimento de rotação em voltas de eles mesmos, isto é atingindo facilmente rodando os objetos correspondentes pelo eixo dos y, como demonstrado em .

```
  //Movimento de translacao
model = glm::translate(model, glm::vec3(sLoc[0], sLoc[1],sLoc[2]));
//Escala
scale = glm::scale(glm::mat4(1.0f), glm::vec3(sR, sR, sR));
//Rotacao
model = glm::rotate(model, glm::radians<float>(t * rotation),vec3(0,
    1, 0));
```

4.12 Modo Menu

No modo menu, colocamos os planetas na sua posição inicial, e com a rotação em pausa, ao fazermos isto temos assim uma linha horizontal contendo todos os planetas (4.11) e um painel com informação relativa ao planeta selecionado.

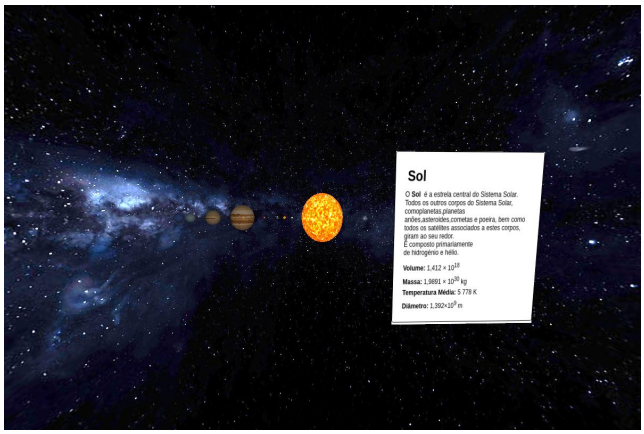

Figura 4.11: Modo menu zoomed out

Ao ativarmos este modo, é feito um *zoom* automático de maneira a ter uma melhor e mais centrada visualização do modelo (4.12).

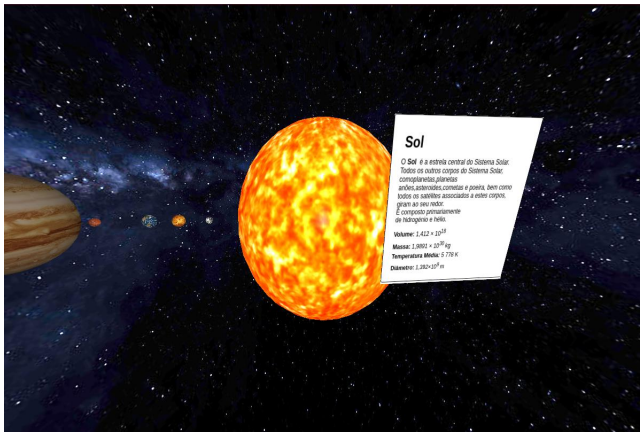

Figura 4.12: Posição inicial do modo menu

```
    if (key == GLFW_KEY_M && action == GLFW_PRESS) {
    paused = true;
    isMenu = true;
    //Zoom
    camx = 800.00;
    camy = 200.00;
    camz = 50.00;
    selected = -1;
    //Posicionamento dos planetas no local inicial
    for (int i = 0; i < 8; i++) {
      for (int j = 0; j < 3; j++) {
        planetlocations[i][j] = planetDefaultlocations[i][j];
      }
    }
  }
```

Excerto de Código 4.9: Ativacao do menu

Podemos também alternar entre cada planeta usando as setas correspondentes, como se encontra referido com maids detalhe em 4.9, ao alternarmos , além da informação presente no painel, irá também ser apresentada informação sobre o planeta selecionado na linha de comandos, sendo esta o nome, raio e massa.

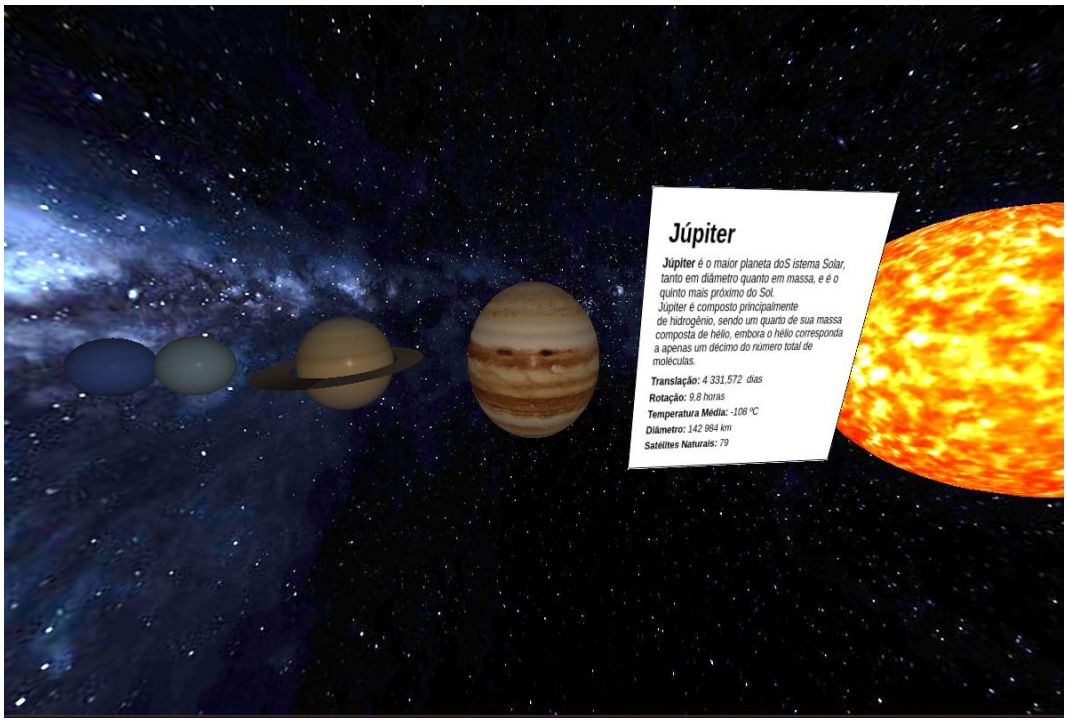

Figura 4.13: Modo Menu Jupiter

4.13 Cintura de Asteróides

Para darmos um pouco mais de realismo ao nosso Sistema Solar decidimos adicionar a cintura de asteróides ao mesmo, utilizando um objeto diferente para os renderizar.

```
    for (int i = 0; i < 32; i++)
  {
    asteroidAngle[i] += asteroidSpeed[i];
    while (asteroidAngle[i] > 360.0)
      asteroidAngle[i] -= 360.0;
    float tempAngle = (asteroidAngle[i] / 180.0f) * 3.14159f;
    asteroidLocations[i][0] = sin(tempAngle) * asteroidDistance[i];
    asteroidLocations[i][2] = cos(tempAngle) * asteroidDistance[i];
  }
```

Excerto de Código 4.10: Calculo do posicionamento dos asteróides

Utilizamos trinta e dois asteróides, tendo assim uma quantidade suficiente para dar a ilusão da cintura. Tendo em conta como a posição atual dos objetos é calculada, ao usarmos ângulos diferentes de inicialização , e ao termos a mesma velocidade de translação em cada, temos assim uma cintura de asteróides (4.14).

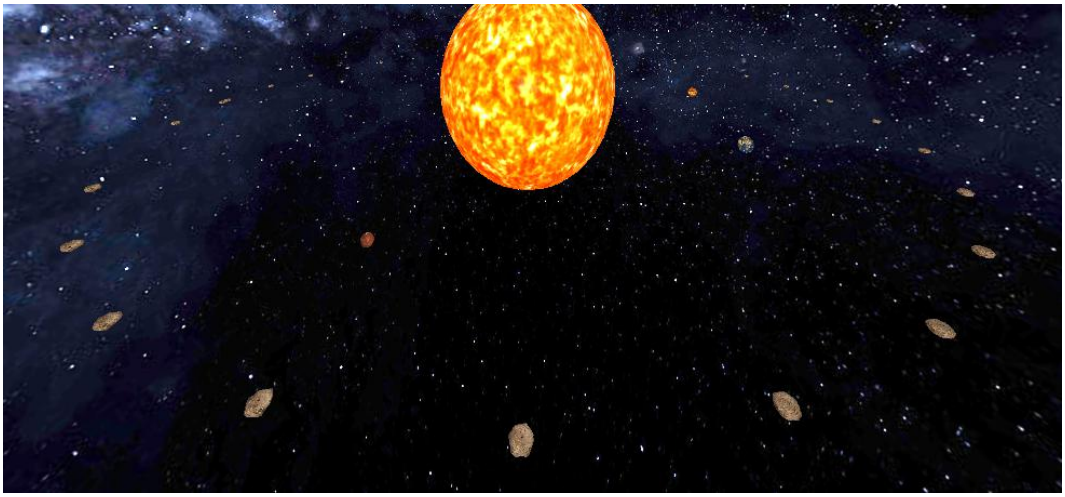

Figura 4.14: Cintura de Asteroides

4.14 Som

Para fazer o som, utilizamos a biblioteca *Irrklang*, desta maneira conseguimos ter uma música de fundo bem como alguns sons de transição.

Para a implementarmos seguimos o tutorial explicado em [1], e através deste introduzimos uma musica de fundo, retirada do jogo "Mass Effect", intitulada "Uncharted Worlds"([3], esta a ser tocada em loop durante a execução do programa.

```
#include <irrKlang.h>
using namespace irrklang;
//Criacao do gestor de som
ISoundEngine* SoundEngine = createIrrKlangDevice();

//Ativacao da musica de fundo antes do main loop
SoundEngine->play2D("background.mp3", GL_TRUE);

glfwSetKeyCallback(window, SceneTexture::keyfunc);

// Enter the main loop
mainLoop(window, scene);
```

Excerto de Código 4.11: Introdução da música de fundo

Capítulo 5

Trabalho Futuro

Tendo em conta que não conseguimos concluir todos os objetivos propostos no enunciado, a parte inicial do trabalho futuro seria completar os mesmos. Para começar iríamos tentar melhorar a iluminação, tendo assim um ponto de luz a sair do Sol. Seguidamente acrescentaríamos sombras aos nossos objetos, finalizando assim o que faltava da iluminação.

Como *features* adicionais, pensaríamos que seria interessante renderizar as trajectórias descritas pelos planetas no nosso sistema, poderíamos também fazer uma otimização no cálculo das rotas bem como no carregamento das texturas, e uma maneira de seguir os planetas a qualquer momento.

Capítulo 6

Considerações Finais

Neste capitulo iremos descrever a estrutura do documento bem como a importância dos capitulos anteriores.

1. O primeiro capítulo – **Motivação** – apresenta o projeto os seus objetivos e a respectiva organização do documento, através deste capitulo conseguimos dar uma ideia geral do que o nosso projecto trata e do que foi requerido, podendo assim ser usado como ponto de referencia para discutir os objetivos alcançados e falhados.

2. O segundo capítulo – **Tecnologias Utilizadas** – descreve as tecnologias utilizadas durante o desenvolvimento deste projeto, desta maneira temos informação do que utilizamos para alcançar os objetivos propostos e contextualizar o projecto temporalmente com as *release versions* das mesmas.

3. O terceiro capítulo – **Etapas de Desenvolvimento** – descreve as diferentes etapas que seguimos até à conclusão do mesmo. Através deste podemos ter uma noção mais detalhada dos passos seguidos até ao produto final e os objectivos que foram tornadas como prioridade.

4. O quarto capítulo – **Descrição do funcionamento do Software** – descreve a maneira como o nosso software funciona, desta maneira podemos observar como foram feitas algumas funcionalidades bem como as utilizar, funcionando como um tutorial e um manual de utilizador.

5. O quinto capítulo – **Trabalhos Futuros** – descreve o trabalho ainda por concluir, incluindo objectivos não alcançados e *features* extra que gostaríamos de desenvolver, desta maneira conseguimos dar uma reflexão do que falhamos e não conseguimos obter devido a restrições temporais e outras.

6. O sexto capítulo – **Considerações Finais** – descreve o documento, incluindo uma pequena reflexão sobre cada capitulo e a sua utilidade.

7. O sétimo capítulo – **Bibliografia** – apresenta a bibliografia consultada durante o desenvolvimento do projecto.

Bibliografia

[1] AMBIERA. https://learnopengl.com/in-practice/2d-game/audio.

[2] https://github.com/slobachev. https://github.com/slobachev/solar-system/tree/master/object.

[3] Sam Hulick. Uncharted worlds.

[4] INOVE. https://www.solarsystemscope.com/textures/.

[5] David Wolff. *OpenGL 4.0 Shading Language Cookbook*. Packt Publishing, 2010.