Universidade da Beira Interior
Departamento de Informática

2020/2021: *SISTEMA SOLAR*

Elaborado por:

Cristiano Santos, nº 43464
Bruno Monteiro, nº 43994
Alexandre Monteiro, nº 44149

Unidade Curricular: Computação Gráfica

Docentes da Cadeira:

Professor Doutor Abel João Padrão Gomes

Professor Nuno Filipe Alexandre Carapito

9 de janeiro de 2021

Agradecimentos

Queriamos começar por agradecer ao Professor Doutor Abel Gomes por toda a matéria lecionada e pela prosposta de projeto que nos permitiu reforçar os conhecimentos pedagógicos da Unidade Curricular.

Gostávamos também de agradecer ao Professor Nuno Carapito por toda a disponibilidade e ajuda prestada.

Conteúdo

Lista de Figuras

Acrónimos

OpenGL Open Graphics Library
GLEW OpenGL Extension Wrangler Library
GLFW Graphics Library Framework
API Application Programming Interface
GLM Generalized Linear Model
IDE Integrated Development Environment
UBI Universidade da Beira Interior

Capítulo

1

Introdução

1.1 Enquadramento

Neste trabalho foi nos dada a escolha de um projecto de entre 5 temas disponíveis. O tema escolhido pelo grupo foi o modelo do Sitema Solar. O trabalho foi realizado em C++ com recusro às bibliotecas Open Graphics Library (OpenGL), Graphics Library Framework (GLFW), OpenGL Extension Wrangler Library (GLEW) e Generalized Linear Model (GLM).

Este relatório foi feito no contexto da unidade curricular de Computação Gráfica da Universidade da Beira Interior (UBI). Foi na UBI que desenvolvemos todo o trabalho.

1.1.1 Enquadramento Histórico

O Sistema Solar foi formado à cerca de 4,5 mil milhões de anos a partir de uma densa nuvem de gás interestelar e poeira.

Quando essa nuvem de poeira entrou em colapso, formou uma nebulosa solar (um disco giratório de material). Vários aglomerados chocaram e formaram objetos cada vez maiores que se juntaram e deram origem aos planetas, planetas anões e luas.

Oficialmente o Sistema Solar tem 8 planetas e 150 luas conhecidas. Mercúrio, Vénus, Terra e Marte são planetas rochosos, Júpiter e Saturno são gigantes gasosos e Urano e Neptuno são gigantes de gelo.

1.2 Motivação

A Computação Gráfica (CG) é uma área da Ciência da Computação que se dedica ao estudo e desenvolvimento de técnicas e algoritmos para a síntese de imagens através do computador. Atualmente, é uma das áreas de maior expansão e importância que propicia o desenvolvimento de trabalhos multi-disciplinares.

Assim o projeto que se apresenta ajudará a aprofundar o conhecimento nesta área cada vez mais importante e adicionalmente aprofundar os conhecimentos aprendidos na Unidade Curricular

1.3 Objetivos

O projecto consiste na realização de um modelo interativo do Sistema Solar, em 2D e 3D, onde projetamos os astros constituintes do mesmo e os seus movimentos (rotações e translações) a uma escala que consideramos adequada, de modo a transmitir ao utilizador uma ideia correta e concreta da realidade.

Caraterísticas da aplicação Gráfica:

1. Modelar o sol, os planetas e os seus respetivos satélites;
2. Texturizar os planetas par aumentar o realismo;
3. Menus que permitam obter informação sobre os elementos do Sistema Solar;
4. A câmera pode mover-se para qualquer posição do Sistema Solar, sendo que este movimento afecta a luz que inside sobre os elementos do Sistema Solar.
5. Utilização do teclado para fazer zoom ao sistema solar e para alterar a vista.
6. Iluminação através de um foco de luz proveniente do Sol.
7. Calcular sombras que os planetas e os satélites poderão originar entre si.

1.4 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** – 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 mais importantes no âmbito deste projeto, bem como as tecnologias utilizadas durante do desenvolvimento do projeto.

3. O terceiro capítulo – **Etapas de Desenvolvimento** – Neste capítulo são apresentadas quais as etapas tidas em conta no desenevolvimento do projeto e qual o membro do grupo que as realizou.

4. O quarto capítulo – **Descrição do funcionamento do Software** – Nesta etapa explicamos ao pormenor como funcionam as funcões que integram o código, qual o seu intuito e de que forma se relacionam.

5. O quinto capítulo – **Considerações Finais** – Neste capítulo elaborámos uma conclusão sobre o projeto e o conhecimento retido com a sua elaboração.

6. O sexto capítulo – **Referências bibliográficas** – Lista das fontes e referências bibliográficas utilizadas.

Capítulo

2

Tecnologias Utilizadas

2.1 Introdução

Neste capítulo descrevemos os conceitos mais importantes no âmbito deste projeto, bem como as tecnologias utilizadas durante do desenvolvimento da projeto.

2.2 Conceitos importantes

Alguns dos conceitos mais importantes:

1. **Básicos de Geometria** – Conhecimmentos teóricos sobre vetores, matrizes e algebra;

2. **Transformações Geométricas** – Noções básicas de transformações de matrizes (matrizes de tranformação, rotação e *scaling*);

3. **Windows and Viewports** – Uma viewport define em coordenadas normalizadas uma área retangular no display onde a imagem dos dados aparece. Uma window define a área retangular em coordenadas.

4. **Shading and Ilumination** – O shading é a implementação do modelo de iluminação nos pixeis e superfícies poligonais dos objetos gráficos. Por sua vez, a Ilumination é a forma como o shading é aplicado ns diferentes objetos.

5. **Color** – Atribuição de diferentes cores da escala RGB (em OpenGL, entre 0 e 1) aos diversos pixeis e objetos.

2.3 Tecnologias Utilizadas

O C++ é uma linguagem de programação e de uso geral. É uma das linguagens mais populares desde 1990 e foi criada por Bjarne Stroustrup.

O OpenGL é uma Application Programming Interface (API) livre utilizada na computação gráfica para o desenvolvimento de aplicações gráficas, ambientes 3D, jogos, etc.

O GLM generaliza a regressão linear permitindo que este seja relacionado à variável de resposta por meio de uma função de ligação, permitindo que a magnitude da variância de cada medição seja uma função do seu valor previsto.

O GLFW é uma biblioteca que pode ser usada com OpenGL. Esta permite que programadores possam criar e manusear janelas e contextos OpenGL. Permite também interagir com joysticks, rato e teclado.

O GLEW é uma biblioteca de carregamento de extensões C/C++ de código *open-source*. Este fornece mecanismos eficientes para determinar quais as extensões OpenGL que são suportadas na plantaforma alvo.

O Visual Studio Integrated Development Environment (IDE) é uma plataforma de lançamento criativa que se pode usar para alterar, depurar, construir código e publicar um aplicação.

Capítulo

3

Etapas de desenvolvimmento

3.1 Introdução

Neste capítulo são apresentadas quais as etapas tidas em conta no desenevolvimento do projeto e qual o membro do grupo que as realizou.

3.2 Parte técnica

1. **Modelação** – Número de objetos que iremos precisar para fazer o sistema solar definir um tamanho para cada planeta e lua. Pensar como iremos realizar a trajetória dos planetas (1);
2. **Blender** – Fazer no Blender planetas e luas (2);
3. **Interação** – Rato responsável pelo movimento da câmera. Teclado(WASD) responsável pelo movimento pela cena (3);
4. **Iluminação** – Pesquisa do melhor tipo de iluminação a usar e sua aplicação (iluminação flat) (4);
5. **Texturização** – Pesquisar e aplicar as texturas dos planetas e luas (5).

3.3 Funcionalidades Extras

1. **Implementação do Menu** – Constituido pelas opções FreeWalk e "Planetas", sendo esta última, uma opção que permite ao utilizador escolher um planeta e ver um pouco das suas caraterísticas (6);
2. **Background** – Aplicação de um background do universo ao modelo (7).

3.4 Relatório

1. **Pesquisa** – Pesquisa sobre o funcionamento e aplicação do LaTeX(8);
2. **Motivação do trabalho e tecnologias utilizadas** (9);
3. **Explicação do Código** (10);
4. **Conclusão e considerações finais** – Breve conclusão sobre o projeto e pequena sintese sobre os conhecimentos consolidados e/ou adquiridos (11);
5. **Finalização do relatório** (12).

3.5 Atribuição de Tarefas

- **Cristiano Santos** – resolução dos tópicos 1, 3, 5, 6, 7, 8, 9, 10, 12;
- **Bruno Monteiro** – resolução dos tópicos 1, 2, 4, 5, 6, 8, 9, 10, 12;
- **Alexandre Monteiro** – resolução dos tópicos 1, 4, 5, 6, 8, 10, 11, 12.

Capítulo

4

Descrição do funcionamento do Software

4.1 Introdução

Nesta etapa explicámos ao pormenor como funcionam as funcões que integram o código, qual o seu intuito e de que forma se relacionam.

No desenvolvimento do projeto, foi necessário organizar o código em diferentes ficheiros de modo a torná-lo mais organizado e percetível para quem o interpreta. Sendo assim, temos os segunites ficheiros: *main.cpp*, *draw.hpp* e o *transferDataToGPU.hpp*. Por sua vez os *shadders* utilizados foram os seguintes: *TransformVertexShader.vertexshader* e o *TextureFragmentShader.fragmentshader*.

4.2 Funcionamento Geral

Quando o programa é iniciado, o utilizador através das arrow keys up and down, pode selecionar uma das 3 opções disponíveis: "Free Walk", "Planets"e "Quit". Carregando no enter entra numa das opções.

Na opção "Free Walk"o utilizador pode andar livremente pela cena, usando as teclas "WASD"e o rato. Clicando no "R"tem uma perspetiva real do Sistema Solar e clicando no "F"os planetas ficam mais próximos uns dos outros e mais estagnados para permitir ao utilizador ver os planetas de forma mais pormenorizada. Clicando no "Q"volta ao Menu Inicial.

Na opção "Planets"o utilizador pode usar as arrow keys up and down para trocar entre a história dos diferentes planetas. Tal como no menu anterior, pode usar as teclas "WASD"e o rato para se mover na cena. Clicando no "Q"volta ao Menu Inicial.

Na opção "Quit"o programa é encerrado.

4.3 *Main.cpp*

Iniciamos este ficheiro com a importação das diferentes bibliotecas necessárias:

```
// Include standard headers
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <vector>

// Include GLEW
#include <GL/glew.h>

// Include GLFW
#include <GLFW/glfw3.h>
GLFWwindow* window;

// Include GLM
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>
using namespace glm;

#include "common/shader.hpp"
#include "common/texture.hpp"
#include "common/controls.hpp"
#include "common/objloader.hpp"
#include "common/shader.cpp"
#include "common/texture.cpp"

#include "common/controls.cpp"
#include "common/objloader.cpp"

#include "transferDataToGPU.hpp"
#include "draw.hpp"
```

Excerto de Código 4.1: *Includes* das bibliotecas usadas.

Criámos uma *typedef struct INFO*, usada para cada um dos seguintes objetos: Background, Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Ring, Asteroid, Moon, Menu1, Menu2, Menu3, MercuryHistory, VenusHistory, EarthHistory, MarsHistory, JupiterHistory, SaturnHistory, UranusHistory, NeptuneHistory, AsteroidHistory, MoonHistory, SunHistory.

```
typedef struct {
  GLuint VertexArrayID;
```

```
  GLuint MatrixID;
  GLuint Texture;
  GLuint TextureID;
  GLuint uvbuffer;
  GLuint vertexbuffer;
}INFO;
```

Excerto de Código 4.2: *typedef struct INFO*.

Criámos uma *typedef struct VECTORS*, usada para cada um dos seguintes objetos: Sphere, Back, VRing, VAsteroid, Menu;

```
typedef struct {
  std::vector<glm::vec3> vertices;
  std::vector<glm::vec2> uvs;
  std::vector<glm::vec3> normals;
}VECTORS;
```

Excerto de Código 4.3: *typedef struct VECTORS*.

Criámos uma *typedef struct TRANSFORMATIONS*, usada para cada um dos seguintes objetos: transfBackground, transfSun, transfMercury, transfVenus, transfEarth, transfMars, transfJupiter, transfSaturn, transfUranus, transfNeptune, transfRing, transfAsteroid, transfMoon, transfList, transfHistory, randomAsteroid[100];

```
typedef struct {
  glm::mat4 self_rotation;
  glm::mat4 center_rotation;
  glm::mat4 translation;
  glm::mat4 scaling;
  float xPosition;
  float yPosition;
  float zPosition;
}TRANSFORMATIONS;
```

Excerto de Código 4.4: *typedef struct TRANSFORMATIONS*.

Criámos a *typedef struct planetRotation*, que foi necessária para a criação da *typedef struct PLANETSROTATIONS*;

```
typedef struct {
  float center_Rotation;
  float self_Rotation;
}planetRotation;
```

Excerto de Código 4.5: *typedef struct planetRotation*.

Criámos uma *typedef struct PLANETSROTATIONS*, usada para cada um dos seguintes objetos: UsingRotation;

```
typedef struct {
  planetRotation sun;
```

```
  planetRotation mercury;
  planetRotation venus;
  planetRotation earth;
  planetRotation mars;
  planetRotation jupiter;
  planetRotation saturn;
  planetRotation uranus;
  planetRotation neptune;
  planetRotation moon;
  planetRotation asteroidBelt;
}PLANETSROTATIONS;
```

Excerto de Código 4.6: *typedef struct PLANETSROTATIONS*.

Criámos uma *typedef struct DIMENSIONS*, usada para cada um dos seguintes objetos: Real, Fake, Using;

```
typedef struct {
  float sun;
  float mercury;
  float venus;
  float earth;
  float mars;
  float jupiter;
  float saturn;
  float uranus;
  float neptune;
  float moon;
  float asteroidBelt;
}DIMENSIONS;
```

Excerto de Código 4.7: *typedef struct DIMENSIONS*.

Após a criação das *typedef structs*, referenciamos as funções criadas e utilizadas. São elas as seguintes:

- *transferDataToGPU()*, contida no *transferDataToGPU.hpp*;
- *draw()*, *setMVP()*, contida no *draw.hpp*;
- *resetPosPlanetsList()*;
- *selectingMenu1()*;
- *selectingMenu3()*;
- *declareDimensions()*;
- *incrementRotations()*;
- *CelestialBodiesPosition()*;

- *deletebuffers()*;

4.3.1 Funções Implementadas

4.3.1.1 *transferDataToGPU()*

Começámos por passar como parâmetros a struct INFO, Vectors, a variavel associada aos shaders, o nome da textura e do objeto.

```
glGenVertexArrays(1, (&a->VertexArrayID));
glBindVertexArray((a->VertexArrayID));
```

Excerto de Código 4.8: *VertexArray* funções.

Nestas duas funções vamos buscar o número de Arrays que existem e atribuimo-los ao VertexArrayID da nossa struct INFO.

```
a->MatrixID = glGetUniformLocation(programID, "MVP");

a->Texture = loadDDS(textureName);

a->TextureID = glGetUniformLocation(programID, "myTextureSampler");

bool res = loadOBJ(objectName, Sphere->vertices, Sphere->uvs, Sphere->
    normals);
```

Excerto de Código 4.9: *VertexArray* funções.

A primeira função vai utilizar o *MatrixID* para lhe atribuir a localizaçao da matriz MVP dos *shaders*. A segunda vai atribuir a variável textura à nossa textura do que nos é passada em parametro para uma função de carregar texturas (loadDDS(*texturename()*). A terceira textura, no fundo, faz o mesmo que a priemira função mas aplicada à textura. A quarta e ultima, vai atribuir à nossa *struct VECTORS* os vértices, os UVs e as normais do objecto.

```
glGenBuffers(1, &a->vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, a->vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, (&Sphere->vertices)->size() * sizeof(glm::
    vec3), &Sphere->vertices[0][0], GL_STATIC_DRAW);

glGenBuffers(1, &a->uvbuffer);
glBindBuffer(GL_ARRAY_BUFFER, a->uvbuffer);
glBufferData(GL_ARRAY_BUFFER, (&Sphere->uvs)->size() * sizeof(glm::vec2)
    , &Sphere->uvs[0][0], GL_STATIC_DRAW);
```

Excerto de Código 4.10: *Buffers* funções.

4.3.1.2 *draw(), setMVP()*

```
glm::mat4 setMVP(TRANSFORMATIONS b) {
  // Compute the MVP matrix from keyboard and mouse input
  computeMatricesFromInputs();
  glm::mat4 ProjectionMatrix = getProjectionMatrix();
  glm::mat4 ViewMatrix = getViewMatrix();
  glm::mat4 ModelMatrix = glm::mat4(1.0);
  glm::mat4 MVP = ProjectionMatrix * ViewMatrix * ModelMatrix * b.
      center_rotation * b.translation * b.self_rotation * b.scaling;
  return MVP;
}
```

Excerto de Código 4.11: *setMVP()*.

Esta função vai receber como parâmetro a *struct Transformations*, que vai ser usada para aplicar à matriz MVP (*computeMatricesFromInputs()*). A matriz MVP foi calculada da seguinte forma : *MVP = ProjectionMatrix * ViewMatrix * ModelMatrix * b.center_rotation * b.translation * b.self_rotation * b.scaling*.

```
void draw(INFO *a, TRANSFORMATIONS b, GLuint programID, GLuint primitive
    , std::vector<glm::vec3>* verticesName) {
  // Use our shader

  glUseProgram(programID);
  glm::mat4 MVP= setMVP(b);
  // Send our transformation to the currently bound shader,
  // in the "MVP" uniform
  glUniformMatrix4fv(a->MatrixID, 1, GL_FALSE, &MVP[0][0]);

  // Bind our texture in Texture Unit 0
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, a->Texture);
  // Set our "myTextureSampler" sampler to use Texture Unit 0
  glUniform1i(a->TextureID, 0);

  // 1rst attribute buffer : vertices
  glEnableVertexAttribArray(0);
  glBindBuffer(GL_ARRAY_BUFFER, a->vertexbuffer);
  glVertexAttribPointer(
    0,                  // attribute
    3,                  // size
    GL_FLOAT,           // type
    GL_FALSE,           // normalized?
    0,                  // stride
    (void*)0            // array buffer offset
  );

  // 2nd attribute buffer : UVs
  glEnableVertexAttribArray(1);
  glBindBuffer(GL_ARRAY_BUFFER, a->uvbuffer);
```

```
  glVertexAttribPointer(
    1,                                  // attribute
    2,                                  // size
    GL_FLOAT,                           // type
    GL_FALSE,                           // normalized?
    0,                                  // stride
    (void*)0                            // array buffer offset
  );

  // Draw the triangle !
  glDrawArrays(primitive, 0, verticesName->size());

  glDisableVertexAttribArray(0);
  glDisableVertexAttribArray(1);
}
```

Excerto de Código 4.12: *draw()*.

A função *draw()* recebe como parâmetros as structs INFO e TRANSFORMATIONS, o *programID* (link para os shaders), a *primitive* e o *verticesName* (vetor com os vertices). A *glActiveTexture* e a *glBindTexture* são usadas para carregar as texturas.

A primeira ocorrência destas funções, *glEnableVertexAttribArray*, *glBindBuffer*, *glVertexAttribPointer* envia os vértices deste objeto para os shaders. A segunda, envia para os shaders os UVs, que por si são as coordenadas das texturas.

Posteriormente ele vai desenhar de acordo com a primitiva passada como parametro, desde o índice 0 até ao tamanho do vetor.

4.3.1.3 *resetPosPlanetsList()*

Esta função vai dar *reset* à posição da camâra quando o utilizador alterna entre planetas no Menu *Planets*

```
void resetPosPlanetsList() {
  position = glm::vec3(145.78f, 2.13f, 39.61f);
  horizontalAngle = -1.55f;
  verticalAngle= -0.03f;
```

Excerto de Código 4.13: *resetPosPlanetsList()*.

4.3.1.4 *selectingMenu1()*

Esta função é responsável pela escolha da opção do Menu Inicial.

```
  if (glfwGetKey(window, GLFW_KEY_ENTER) == GLFW_PRESS && glfwGetKey(
      window, GLFW_KEY_ENTER) == GLFW_RELEASE)
  {
```

```
    if (selectedMenu1 == 1)
      currentMenu = 2;
    else if (selectedMenu1 == 2) {
      resetPosPlanetsList();
      currentMenu = 3;
    }

    else if (selectedMenu1 == 3)
      currentMenu = 4;
  }
```

Excerto de Código 4.14: *Pequeno excerto da função selectingMenu1().*

4.3.1.5 *selectingMenu3()*

Esta função é a responsável pela escolha da história dos planetas que irão aparecer.

```
draw(&Background, transfBackground, programID, GL_TRIANGLES, &Back.
    vertices);
  if (glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_PRESS && glfwGetKey(
      window, GLFW_KEY_DOWN) == GLFW_RELEASE)
  {
    resetPosPlanetsList();
    if (selectedMenu3 == 11)
      selectedMenu3 = 1;
    else
      selectedMenu3++;
  }
  if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_PRESS && glfwGetKey(window
      , GLFW_KEY_UP) == GLFW_RELEASE)
  {
    resetPosPlanetsList();
    if (selectedMenu3 == 1)
      selectedMenu3 = 11;
    else
      selectedMenu3--;
  }
```

Excerto de Código 4.15: *Pequeno excerto da função selectingMenu3().*

4.3.1.6 *declareDimensions()*

Nesta função nós declaramos a escala real dos planetas e uma escala *Fake*, sendo esta última usada para visualizar melhor os planetas. Atribuimos um valor *random* ã posição inicial dos planetas em torno do Sol.

```
Real.sun = 0.0f * AU;
Real.mercury = 0.28f * AU;
Real.venus = 0.367f * AU;

(...)

Fake.sun = 0.0f * AU;
Fake.mercury = 0.28f * AU;
Fake.venus = 0.367f * AU;

(...)

UsingRotation.sun.center_Rotation = (float)(rand() % 365);
UsingRotation.mercury.center_Rotation = (float)(rand() % 365);
UsingRotation.venus.center_Rotation = (float)(rand() % 365);
```

Excerto de Código 4.16: *Pequeno excerto da função declareDimensions().*

4.3.1.7 *incrementRotations()*

Esta função é a responsável pela escala de rotação e translação dso planetas.

```
UsingRotation.sun.center_Rotation += 0.0f;
UsingRotation.mercury.center_Rotation += 4.20f * rotationScale;
UsingRotation.venus.center_Rotation += 1.43f * rotationScale;

(...)

UsingRotation.sun.self_Rotation += 0.0115f;
UsingRotation.mercury.self_Rotation += 0.017f;
UsingRotation.venus.self_Rotation -= 0.004f;
```

Excerto de Código 4.17: *Pequeno excerto da função incrementRotations().*

4.3.1.8 *CelestialBodiesPosition()*

Esta função define a translação em torno do Sol, a rotação sobre si mesma e o tamanho do planeta(*scaling*).

```
//Sol
  transfSun.center_rotation = glm::rotate(glm::mat4(1.0), glm::radians(
      UsingRotation.sun.center_Rotation), glm::vec3(0, 1, 0));
  transfSun.self_rotation = glm::rotate(glm::mat4(1.0), glm::radians(
      UsingRotation.sun.self_Rotation), glm::vec3(0, 1, 0));
  transfSun.translation = glm::translate(glm::mat4(1.0), glm::vec3(Using
      .sun, 0.0f, 0.0f));
  transfSun.scaling = glm::scale(glm::mat4(1.0), glm::vec3(19.0f));
```

```
//Mercury
  transfMercury.center_rotation = glm::rotate(glm::mat4(1.0), glm::
      radians(UsingRotation.mercury.center_Rotation), glm::vec3(0, 1, 0)
      );
  transfMercury.self_rotation = glm::rotate(glm::mat4(1.0), glm::radians
      (UsingRotation.mercury.self_Rotation), glm::vec3(0, 1, 0));
  transfMercury.translation = glm::translate(glm::mat4(1.0), glm::vec3(
      Using.mercury, 0.0f, 0.0f));
```

Excerto de Código 4.18: *Pequeno excerto da função CelestialBodiesPosition().*

4.3.1.9 *deletebuffers()*

Nesta função apagamos o *programID*, todas as texturas, *buffers*(contêm vértices dos objectos e UVs). Apagamos ainda os *VertexArrayID*'s.

```
glDeleteProgram(programID);

//Delete Textures
glDeleteTextures(1, &Background.Texture);
glDeleteTextures(1, &Sun.Texture);

//Delete buffers
glDeleteBuffers(1, &Background.vertexbuffer);
glDeleteBuffers(1, &Background.uvbuffer);

//Delete VertexArrays
glDeleteVertexArrays(1, &Background.VertexArrayID);
glDeleteVertexArrays(1, &Sun.VertexArrayID);
```

Excerto de Código 4.19: *Pequeno excerto da função deletebuffers().*

4.3.1.10 *Cintura de Asteróides*

Depois de declarmos o randomAsteroid[100] do tipo *strut* TRANSFORMATIONS, percorremos cada posição do Array e atribuímos uma posição para o asteróide correspondente e um tamanho variável. De forma a tornar isto o mais *random* possível gerámos uma nova *seed* em cada execução do programa.

```
    int j = 0;
    float i = 0;
    //Random seed
    time_t t;
    srand((unsigned) time(&t));
    for (i = 0, j = 0; i < 360; i += 3.6, j++)
    {
        //float cent_Rotation = (((float)(rand()) / RAND_MAX) * i) + i;
        float cent_Rotation = (rand() % 4) + i - 0.4;
```

```
        float transl = ((rand() % 10) + AU - 10);
        float scal = ((float)(rand()) / RAND_MAX) * 0.1f;

        randomAsteroid[j].self_rotation = glm::rotate(glm::mat4(1.0),
            glm::radians(1.0f), glm::vec3(0, 1, 0));
        randomAsteroid[j].center_rotation = glm::rotate(glm::mat4(1.0),
            glm::radians(cent_Rotation), glm::vec3(0, 1, 0));
        randomAsteroid[j].translation = glm::translate(glm::mat4(1.0),
            glm::vec3(transl, 0.0f, 0.0f));
        randomAsteroid[j].scaling = glm::scale(glm::mat4(1.0), glm::vec3
            (scal));
    }
```

Excerto de Código 4.20: *Excerto de código para a criação da cintura de Asteroides.*

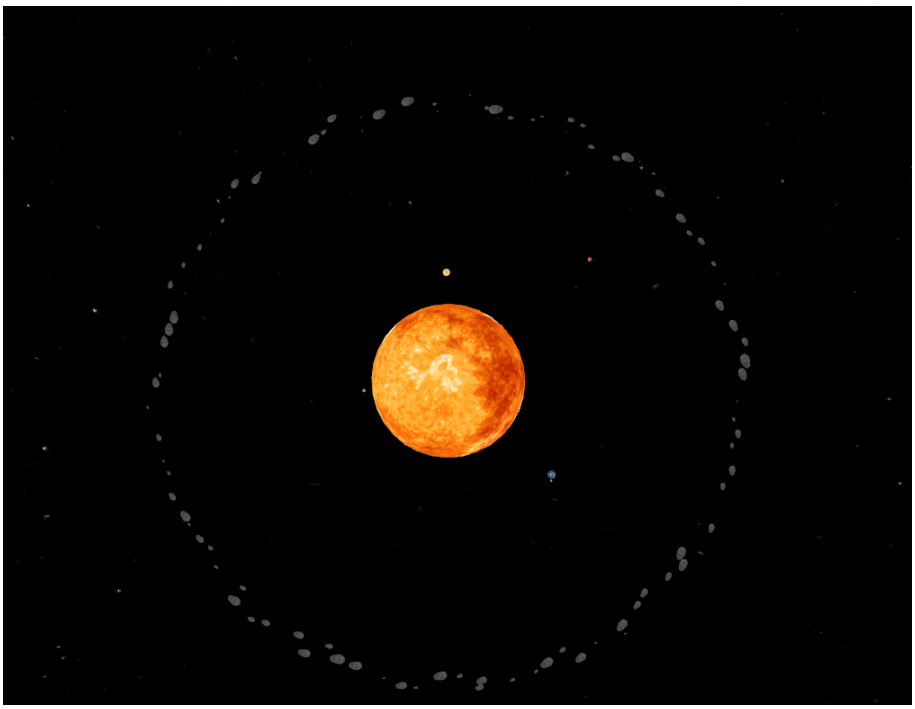

Figura 4.1: Cintura de Asteróides vista de cima

Figura 4.2: Menu Inicial

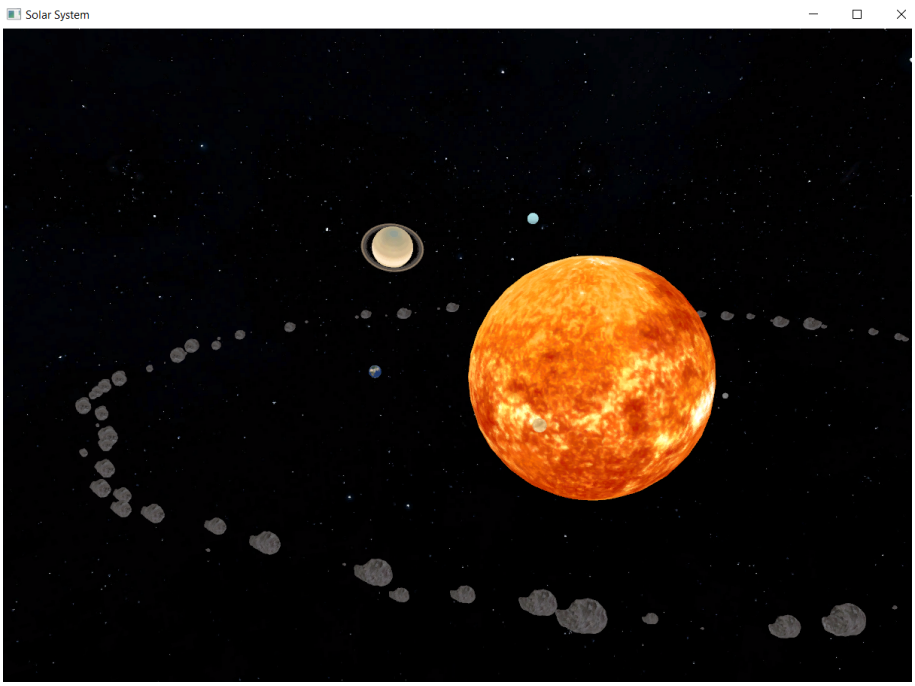

Figura 4.3: Perspetiva do Sistema Solar

Figura 4.4: História da Terra

4.4 *Shaders*

4.4.1 *VertexShader*

Os *shaders* recebem as posições dos vértices do objeto que são armazenadas no *vertexPosition_modelspace*. Recebem também as posições dos objetos no qual terá a textura, estas são guardadas no *vertexUV*.

O MVP é uma matriz 4x4 recebida do *draw()* através do: *glUniformMatrix4fv(a->MatrixID, 1, GL_FALSE, &MVP[0][0])*.

Primeiramente transformamos o *vertexPosition_modelspace* num vec4, uma vez que a matriz MVP é um mat4. Desta forma a multiplicação é possível para obter o *gl_Position*.

O UV recebe os *VertexUV* e envia-os para o *FragmentShader*.

```
#version 330 core

layout(location = 0) in vec3 vertexPosition_modelspace;
layout(location = 1) in vec2 vertexUV;

out vec2 UV;

uniform mat4 MVP;

void main(){

  gl_Position =  MVP * vec4(vertexPosition_modelspace,1);
```

```
  UV = vertexUV;
}
```

Excerto de Código 4.21: *VertexShader*

4.4.2 *FragmentShader*

O *FragmentShader* recebe os UV's vindos do *VertexShader* e a textura (*myTextureSampler*) recebida do *transferDataToGPU()* através da função:
a->TextureID = glGetUniformLocation(programID, "myTextureSampler").

Aplicamos a função *texture()* onde passamos como argumento a textura e os UV's. Esta tranformação é enviada como vec3 para o OpenGL.

Como só existe um único *out* o OpenGL já sabe qual é a variável das cores.

```
#version 330 core

in vec2 UV;

out vec3 color;

uniform sampler2D myTextureSampler;

void main(){

  color = texture( myTextureSampler, UV ).rgb;
}
```

Excerto de Código 4.22: *FragmentShader*

Capítulo

5

Conclusões e Trabalho Futuro

5.1 Conclusão

A concretização do projeto foi um processo demorado, mas simultanemante estimulante e desafiador para os alunos. A realização deste projeto constituiu-se uma mais valia na nossa formação e espelha a sistematização dos conhecimentos teórico-práticos adquiridos ao longo da Unidade Curricular.

Permitiu-nos expandir e consolidar os nossos conhecimentos referentes a C++ e ao OpenGL.

Devido à falta de tempo, não conseguimos implementar o sistema de iluminação como desejávamos, sendo algo a ter em conta em versões futuras do programa.

Um ponto a destacar foi a oportunidade de usar o Blender que facilitou a construção dos objetos. Foi conseguida a interação do sistema teclado e rato para com o utilizador. As texturas deram um realce mais estético ao nosso Sistema Solar. A implementação de uma funcionalidade no Menu que permite consultar a história de um planeta à escolha, permite ao utilizador aumentar os seus conhecimentos.

Posto isto e tendo o prazo sido cumprido, achamos que o nosso projeto foi concluido com sucesso.

5.2 Trabalho Futuro

No futuro pretendemos aplicar uma melhor iluminação e seria nossa intenção explorar formas de otimizar melhor o código. Seria de salientar a implementação de novas funcionalidades ao programa bem como melhorar a interação com o *user*.

Bibliografia

[1] *https://learnopengl.com/*

[2] *https://www.di.ubi.pt/ agomes/cg/*

[3] *https://www.solarsystemscope.com/textures/*

[4] *https://solarsystem.nasa.gov/*

[5] *https://www.education.com/science-fair/article/scale-model-planets-solar-system/*

[6] *https://www.exploratorium.edu/ronh/solar*$_{s}ystem/scale.pdf$