UNIVERSIDADE DO MINHO
MIEI

COMPUTAÇÃO GRÁFICA

Segunda Fase
Transformações Geométricas

Grupo:
João Nunes - a82300
Shahzod Yusupov - a82617
Luís Braga - a82088
Luís Martins - a82298

Braga, Portugal
5 de Abril de 2019

Conteúdo

1 Introdução

No âmbito da unidade curricular de Computação Gráfica foi proposta, nesta segunda fase, que se implementasse um sistema capaz de desenhar um certo conjunto de figuras, feitas anteriormente na primeira fase do trabalho, à base de translações, rotações e escalonamento. Para tal, foi preciso reformular a maneira de olhar para o ficheiro XML a percorrer e, consequentemente, para o *parse* elaborado para a primeira versão de XML realizado. Tendo, por isso, sofrido uma grande reformulação na implementação do mesmo. Além disso, foi necessário também modificar a forma de desenhar depois deste ficheiro estar lido, pois agora teve-se que incluir as transformações geométricas referidas em cima.

O principal objetivo desta fase é que seja realizado um sistema solar estático através da leitura do ficheiro XML, tendo também, por base, a definição da escala para representar este mesmo sistema.

2 Estrutura do projeto

Esta segunda fase do projeto divide-se no *generator* e no *engine*, sendo que no generator geram-se os pontos, guardando-os num ficheiro, que mais tarde serão usados para desenhar as figuras no engine (através da leitura de ficheiros XML). Para o engine, foi necessário criar novas classes auxiliares que facilitassem a leitura de ficheiros, que serão explicadas na secção do engine. É de referir que se retiraram algumas funcionalidades do engine, uma vez que nesta fase do projeto já não fariam sentido existir.

2.1 Generator

É de referir que esta componente do projeto sofreu uma alteração, embora o principal foco nesta fase tenha sido a construção do ficheiro XML e a respetiva leitura efetuada, o grupo de trabalho decidiu também incluir o método necessário de modo a gerar os pontos para a construção de um torus, cujo intuito é aplicar no desenho da cintura de asteroides e kuiper, e no anel desenhado à volta de Saturno.

Contudo, as diferentes estruturações de modo a desenvolver os vértices para as figuras geométricas da fase anterior ainda se mantêm.

2.1.1 Nova primitiva geométrica

Torus

A elaboração do sistema solar trouxe ao grupo novas dificuldades, tais como a criação do anel de Saturno, a Cintura de Asteroides e ainda a Cintura de Kuiper. Para tal, o grupo elaborou uma figura que satisfizesse a composição destas figuras.

Um Torus é uma figura geométrica que tem um formato tipo "*donut*". Para gerar esta figura são necessários os seguintes parâmetros: raio da Coroa (raio de um anel do Torus), raio (distância do centro do Torus ao centro de cada anel), faces (número de faces de cada anel) e anéis (número de anéis que vai constituir o Torus).

Para a elaboração do Torus é crucial considerar que a sua estrutura baseia-se na distância até ao centro dos anéis e no raio dos anéis.

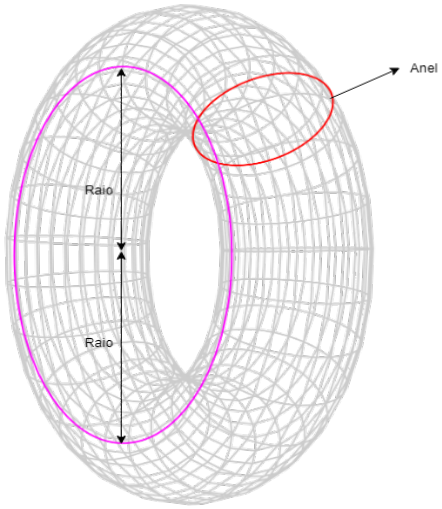


Figura 2.1: Torus desenhado com linhas.

O algoritmo de cálculo de pontos começa por selecionar um anel desenhando depois todas as faces deste. Podendo confirmar-se o que é um anel e o raio na figura acima e o que é uma face na figura abaixo.

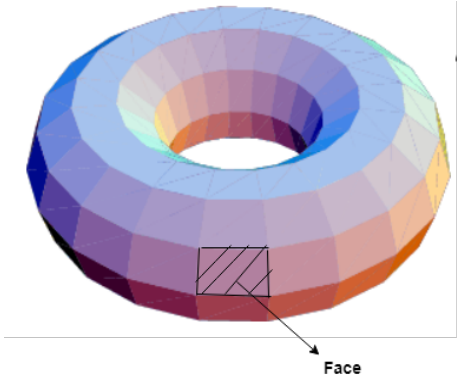


Figura 2.2: Demonstração do que é uma face.

É de notar que tanto o anel de Saturno como as "Cinturas"tiram partido deste algoritmo, sendo que o anel de Saturno é um Torus como está representado nas figuras (2.1 e 2.2), enquanto que as "Cinturas"são conseguidas, desenhando figuras (com o algoritmo da esfera, apesar de conterem poucos slices e stacks) em cada ponto que constitui o Torus. Obtendo-se assim uma nuvem de esferas.

2.2 Engine

De modo a ler a nova estrutura dos ficheiros XML, criaram-se as seguintes classes: *translation*, *rotation*, *scale*, *colour* e *group*.

No **engine.cpp** tiveram que se efetuar alterações na maneira como é lida no parser o ficheiro XML , tal como referido anteriormente, sendo que esta aplicação receberá um ficheiro XML com a estrutura que será referida posteriormente. Além disso, a maneira de como são renderizados pelo *GLUT* também sofreram alterações, visto que se tem que ter em conta que a ordem com que se faz translações, rotações e escalonamento são de extrema relevância. Por exemplo, primeiro executa-se uma translação, em seguida uma rotação e o escalanomento, sendo que por último se faz o desenho dos pontos e respetivos triângulos. Por outro lado, se, por exemplo, no caso da Terra que tem como satélite a Lua, faz-se as transformações geométricas (pela ordem anteriormente referida), não relativamente ao sol (como o caso do Terra) mas relativamente ao planeta indicado.

2.2.1 Classes

Para além das classes feitas na fase anterior, o grupo de trabalho decidiu também, face ao problema em mão, criar novas classes com diferentes intuitos. Foram criadas 3 classes para cada uma das transformações geométricas ou seja, o **Rotate**, **Scale** e **Translate**. Para além destas 3 novas classes, foram elaboradas também o **Colour**, de modo a facultar uma possível cor aos planetas (*RGB*) e por fim o **Group**, que armazena um conjunto de formas geométricas e posteriormente associa às respetivas transformações geométricas efetuadas sobre a forma.

É de referir também, que cada uma das classes foi escrita usando uma sintaxe *"orientada aos objetos"*, portanto cada uma das classes apresentadas posteriormente irá possuir os construtores, os gets e os sets.

Point

Na classe **Point**, está explicita a estrutura necessária para guardar um ponto, coordenadas (x,y,z), necessárias para constituir um triângulo.

```
#ifndef POINT_H__
#define POINT_H__

#include <string>

using namespace std;

class Point {
        float x;
        float y;
        float z;

        public:
                Point();
                Point(float, float, float);
                float getX();
                float getY();
                float getZ();
                void setX(float);
                void setY(float);
                void setZ(float);
                string to_String();
};

#endif
```

Figura 2.3: Declaração dos métodos e variáveis no point.h

Struct

Nesta classe, são guardados o conjunto de pontos necessários para representar as determinadas figuras. Contendo como tal a estrutura **vector<Point*> LP**, ou seja, uma lista de pontos (LP).

```
#ifndef STRUCT_H__
#define STRUCT_H__

#define _USE_MATH_DEFINES

#include <math.h>
#include <iostream>
#include <vector>
#include "Point.h"

using namespace std;

class Struct {
        vector<Point*> LP;

public:
        Struct();
        Struct(vector<Point*>);
        vector<Point*> getLP();
        void setLP(vector<Point*>);
        void genPlane(float);
        void genCylinder(float, float, int);
        void genSphere(float, int, int);
        void genCone(float , float , int , int );
        void genBox(float, float, float, int);

};

#endif
```

Figura 2.4: Declaração dos métodos e variáveis no struct.h

Rotate

Classe que guarda informação necessária para aplicar uma rotação, existindo para tal o vetor (x,y,z), sobre o qual a rotação será efetuada, e o ângulo (*angle*), correspondente o ângulo de rotação do vetor.

```
#ifndef __ROTATE_H__
#define __ROTATE_H__

class Rotate {
        float angle;
        float x;
        float y;
        float z;

public:
        Rotate();
        Rotate(float, float, float, float);
        float getAngle();
        float getX();
        float getY();
        float getZ();
        void setAngle(float);
        void setX(float);
        void setY(float);
        void setZ(float);
};

#endif
```

Figura 2.5: Declaração dos métodos e variáveis no rotate.h

Scale

Classe que guarda a informação necessária para redimensionar, sendo para tal necessárias três variáveis representativas das três novas dimensões relativas aos eixos (x,y,z).

```
#ifndef __SCALE_H__
#define __SCALE_H__

class Scale {
        float x;
        float y;
        float z;

public:
        Scale();
        Scale(float, float, float);
        float getX();
        float getY();
        float getZ();
        void setX(float);
        void setY(float);
        void setZ(float);
};

#endif
```

Figura 2.6: Declaração dos métodos e variáveis no scale.h

Translate

Classe que guarda a informação necessária para efetuar uma rotação, sendo então necessário, tal como nos anteriores, as variáveis de instância x,y,z que representam um vetor de translação em relação à posição original.

```
#ifndef __TRANSLATE_H__
#define __TRANSLATE_H__

class Translate {
        float x;
        float y;
        float z;

public:
        Translate();
        Translate(float, float, float);
        float getX();
        float getY();
        float getZ();
        void setX(float);
        void setY(float);
        void setZ(float);
};

#endif
```

Figura 2.7: Declaração dos métodos e variáveis no translate.h

Colour

Classe necessária para aplicar a cor para alterar a cor das figuras pretendidas. A cor é processada segundo o sistema de cores *RGB*, portanto foi necessária a declaração de três variáveis para cada cor, o vermelho, verde e azul.

```
#ifndef __COLOUR_H__
#define __COLOUR_H__

class Colour {
        float rr;
        float gg;
        float bb;

public:
        Colour();
        Colour(float, float, float);
        float getRR();
        float getGG();
        float getBB();
        void setRR(float);
        void setGG(float);
        void setBB(float);
};

#endif
```

Figura 2.8: Declaração dos métodos e variáveis no colour.h

Group

O Group é uma classe cujo intuito é armazenar toda a informação referente a um grupo. A classe foi criada com o intuito de aquando na leitura dos ficheiros *XML*, em cada grupo lido e posteriormente interpretado, irá corresponder à estrutura Group.

O Group irá ter informação relativa ao vetor de pontos a desenhar do dado grupo **LP**, o **childs** que é uma lista que contêm os filhos do Group, importante para estabelecer uma hierarquia no ficheiro de configuração. De seguida possui também a informação relativa às transformações geométricas, desde a rotação, **rotation**, a translação, **translation**, e a escala **scale**. É também guardada informação referente à cor, **colour**.

A estrutura de dados Group, será vital para o armazenamento dos dados retirados a partir do parse, que será apresentado ulteriormente

```
#ifndef __GROUP_H__
#define __GROUP_H__

#include <vector>
#include "rotate.h"
#include "translate.h"
#include "scale.h"
#include "colour.h"
#include "../../../src/headers/Point.h"

using namespace std;

class Group {
        Rotate* rotation;
        Translate* translation;
        Scale* scale;
        Colour* colour;
        vector<Point*> LP;
        vector<Group*> childs;

public:
        Group();
        Rotate* getRotation();
        Translate* getTranslation();
        Scale* getScale();
        Colour* Group::getColour();
        vector<Point*> getLP();
        vector<Group*> getChilds();
        void setRotation(Rotate*);
        void setTranslation(Translate*);
        void setScale(Scale*);
        void setColour(Colour*);
        void setLP(vector<Point*>);
        void addChild(Group*);

};

#endif
```

Figura 2.9: Declaração dos métodos e variáveis no group.h

3 Processo de apresentação do sistema solar

Tendo explicado anteriormente todas as classes necessárias para a base do projeto, de seguida, apresentam-se os vários passos até apresentar o cenário final ao utilizador.

3.1 Processo de leitura e interpretação

Todo o processo de leitura é feito pelo engine, sendo que este se encontra no **parser**, onde o processo é inicializado quando um ficheiro *XML* é fornecido à aplicação do engine.

O ficheiro *XML* é explorado recursivamente recorrendo para tal à função **parseGroup(Group* g, XMLElement* e)**, que a cada chamada recursiva onde sabe exactamente o elemento XML que se encontra a explorar (e), e o grupo (g) onde a informação recolhida, fruto do parse, será armazenada.

Na primeira chamada desta função, temos que é dado como input à função o primeiro filho do ficheiro *XML*, que será, se tudo estiver correto, um group. De seguida, é testado se o elemento que se está a ler corresponde às transformações geométricas, rotate, scale ou translate ou colour, sendo que nesse caso serão encaminhados para as funções respetivas para efetuar o parse de cada um desses elementos. De seguida, o resultado do parse de cada um dos elementos anteriormente referidos, é guardado na estrutura do group atual.

```
        XMLElement* trans = e->FirstChildElement("translate");
        if (trans != nullptr) parseTranslate(g, trans);

        XMLElement* rotate = e->FirstChildElement("rotate");
        if (rotate != nullptr) parseRotate(g, rotate);

        XMLElement* scale = e->FirstChildElement("scale");
        if (scale != nullptr) parseScale(g, scale);

        XMLElement* colour = e->FirstChildElement("colour");
    if (colour != nullptr) parseColour(g, colour);
```

Figura 3.1: Extrato de código do parseGroup relativo ao parse das transformações e cor.

Caso o elemento a ser interpretado não seja uma transformação geométrica ou cor, então só poderá ser uma lista de modelos ou um grupo filho. No primeiro caso, é chamada a função auxiliar **readFile** com o nome do modelo interpretado. O nome do modelo do ficheiro interpretado terá apenas de acabar com a extensão *".3d"*. De seguida o ficheiro, com esse nome, é aberto, sendo depois efetuado um *ciclo for* com o número de vértices presentes no ficheiro, sendo então

percorridas as linhas todas do ficheiro, efetuando a sua transformação em floats. No final da invocação da função **readFile** teremos então a lista com os pontos da figura interpretada, sendo esta adicionada à estrutura group.

```
            XMLElement* models = e->FirstChildElement("models");
            if (models != nullptr) {
                    models = models->FirstChildElement("model");
                    while(models!=nullptr) {
                    g->setLP((readFile(models->Attribute("file"))));
            models = models->NextSiblingElement();
            }
        }
```

Figura 3.2: Extrato de código do parseGroup relativo ao parse do modelo.

```
vector<Point*> readFile(string name) {
        vector<string> coord;
        string buffer;
        string linha;
        int index = 0;

        vector<Point*> LP;
        ifstream file(name);
        if (file.is_open()) {
                getline(file, linha);
                int nvertices = atoi(linha.c_str());
                int i;
                for (i = 0; i < nvertices; i++) {
                        getline(file, linha);
                        stringstream ss(linha);
                        while (ss >> buffer) coord.push_back(buffer);

                        float x = stof(coord[index]);
                        float y = stof(coord[index + 1]);
                        float z = stof(coord[index + 2]);
                        LP.push_back(new Point(x, y, z));
                    index += 3;
                }
        }
        else { cout << "Nao foi possível abrir o ficheiro 3d
        (possivelmente não terá gerado os pontos da figura
        a desenhar com o 'generator')" << endl; }

        return LP;
}
```

Figura 3.3: Extrato de código da função readFile.

Caso seja um grupo filho, é instanciada uma nova estrutura de dados Group, sendo esta adicionada à lista de groups associadas ao group pai. Todo este comportamento é feito com o

intuito de estabelecer uma hierarquia, de seguida é chamada a função **parseGroup** novamente, recursivamente.

```
XMLElement* childs = e->FirstChildElement("group");
        if (childs!=nullptr) {
                while (childs != nullptr) {
                        nr_group++;
                        Group* child = new Group();
                        g->addChild(child);
                        if (childs) parseGroup(child, childs);
childs = childs->NextSiblingElement();
            }
    }
```

Figura 3.4: Extracto de código da função readFile relativo ao parse do group.

Resta referir portanto, que a estrutura do ficheiro XML responsável pelo ficheiro de configuração do sistema solar, está presente no anexo do relatório.

3.2 Processo de renderização

No que toca à renderização, uma vez alcançado o primeiro group, todos os restantes grupos irão ser o seu grupo filho, pelo que o processo de renderização começa no primeiro group.

A função cujo intuito é a renderização, é a **renderScene**, cuja principal alteração da primeira fase é **drawGroups** que é chamado recursivamente no corpo da função.

```
void renderScene(void) {

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glLoadIdentity();

        Point* camPos = camera.getCamPosition();
        Point* lookP = camera.getLookPoint();
        Point* titl = camera.getTitl();

        gluLookAt(camPos->getX(), camPos->getY(), camPos->getZ(),
                lookP->getX(), lookP->getY(), lookP->getZ(),
                titl->getX(), titl->getY(), titl->getZ());

        glPolygonMode(GL_FRONT_AND_BACK, linha);
        glFrontFace(face);

        if(nr_groups > 0) drawGroups();

        glutSwapBuffers();
}
```

Figura 3.5: Extrato de código da função renderScene.

A função **drawGroups** irá por sua vez, também chamar recursivamente a função **drawGroup** que recebe como argumento um *Group**. Nesta função uma vez que a matriz de transformação será alterado, é necessário primeiramente guardar o seu estado, de seguida efetuar as transformações necessárias, sendo elas translate, rotate, scale ou até mesmo aplicar cor, que o grupo decidiu criar de modo a aproximar o modelo do sistema solar à realidade, e só depois é que o estado é reposto.

Tendo isto em conta, e uma vez aplicadas as transformações e cor (caso tenham), segundo os parâmetros guardados na estrutura Group, só nesta instância é que se está pronto para começar a desenhar o sistema solar, bastando percorrer a lista de pontos na estrutura de dados Group e desenhar sequencialmente os triângulos.

A função no final, é chamada recursivamente para cada um dos filhos do Group, de modo a repetir o processo anterior na totalidade da estrutura de dados.

```
void drawGroups() {
        int size = LG.size();
        for (int i = 0; i < size; i++) {
                drawGroup(LG[i]);
        }
}

void drawGroup(Group* g){
        Translate* trans = g->getTranslation();
        Rotate* rotat = g->getRotation();
        Scale* scale = g->getScale();
        Colour* colour = g->getColour();

        vector<Group*> childs = g->getChilds();
        int size = childs.size();

        glColor3f(colour->getRR(), colour->getGG(), colour->getBB());
        glPushMatrix();

        glTranslatef(trans->getX(), trans->getY(), trans->getZ());
        glRotatef(rotat->getAngle(),rotat->getX(), rotat->getY(), rotat->getZ());
        glScalef(scale->getX(), scale->getY(), scale->getZ());

                drawFigure(g->getLP());
                for (int i = 0; i < size; i++) {
                        drawGroup(childs[i]);
                }
        glPopMatrix();
}
```

Figura 3.6: Extratos de código da função drawGroups e drawGroup.

4 Escala do sistema solar

De modo a aproximar o sistema solar da realidade, foi elaborada uma escala tendo sempre em mente as restrições e imposições de modo a ficar percetível ao utilizador quando este tentasse correr a aplicação.

Por outro lado, ao tentar estabelecer uma escala realista, houve dificuldade uma vez que os últimos quatro planetas encontram-se a uma larga distância do sol, bem como os satélites de cada um dos planetas uma vez que estes são pequenos, tornando-se num processo difícil de conciliar a realidade com a escala estabelecida pelo grupo.

De seguida, encontra-se as informações em que o grupo se baseou para construir a escala, bem como a escala estabelecida pelo grupo de trabalho:

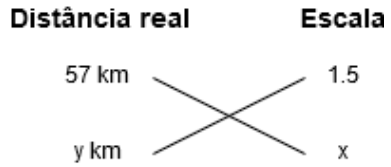


Figura 4.1: Escala base utilizada para descobrir a distância dos planetas em relação ao sol

Pelo que a fórmula utilizada para calcular a escala para os quatro primeiros planetas, tendo sempre por base a distância em relação ao centro do sistema solar, ou seja o sol, utilizando a regra de três simples anterior é:

$$x = \frac{1.5 * y}{57}$$

Onde neste caso, o y é a distância real, e o x é a distância, aproximada, utilizada pelo grupo de trabalho na construção do sistema solar.

É de referir também que cada um dos valores reais utilizados pelo grupo foi retirado de um papel científico disponível na bibliografia do trabalho.

Para os últimos quatro planetas, o grupo estabeleceu como base a distância ao Júpiter de 12 unidades em relação ao sol, uma vez que existe a cintura de asteroides entre Marte e Júpiter. Tendo isto em conta, após algumas tentativas foi estabelecida a seguinte fórmula geral para o cálculo da escala dos três últimos planetas, ou seja, Saturno, Úrano e Neptuno:

$$x_i = x_{i-1} + \frac{d_i - d_{i-1}}{200} \ , \ para \ i \in \{2, 3, 4\}$$

Onde o i representa os 4 últimos planetas ou seja, $i = 1$ temos que este é Júpiter, com o $i = 2$ Saturno e assim sucessivamente. O x_i é a escala utilizada pelo grupo de modo a estabelecer o *translate* de uma maneira realista, enquanto que o d_i é a distância real.

Por exemplo, para Saturno ou seja $i = 2$, temos a seguinte igualdade:

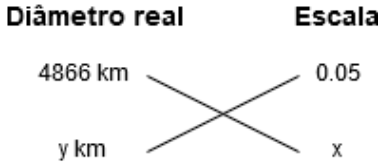
$$x_2 = x_1 + \frac{d_2 - d_1}{200}$$

Onde, substituindo as variáveis pelos seus valores correspondentes:

$$x_2 = 12 + \frac{1437 - 780}{200} = 15.285$$

Portanto, o valor utilizado pelo grupo para estabelecer o *translate* de Saturno será de 15.285.

No que toca ao diâmetro dos planetas, para os primeiros quatros planetas, foi utilizado como base o valor de 0.05 que corresponde a Mercúrio na escala estabelecida pelo grupo, onde temos que novamente, recorrendo a uma regra de três simples:



Onde se deriva a seguinte fórmula geral, tendo por base o diâmetro de Mercúrio para os primeiros quatro planetas:

$$x = \frac{0.05 * y}{4866}$$

Onde, x é a escala utilizada pelo grupo e y é o diâmetro real do planeta, os diâmetros reais utilizados pelo grupo de trabalho, também podem ser consultados através de um *link* disponível na bibliografia.

Para os quatro últimos planetas, o grupo teve por base o planeta Neptuno, onde foi determinado que 0.3 seria a escala do diâmetro deste planeta, sendo que 45432 é o seu diâmetro real. De seguida, os restantes três últimos planetas, Júpiter, Úrano e Neptuno têm a escala calculada tendo por base o diâmetro de Neptuno.

$$x = (\frac{0.3 * y}{45432})/1.5$$

A variável x é escala do planeta, e o y é o diâmetro real do planeta em questão. O 1.5 na fórmula supracitada foi adicionado de forma a aproximar o diâmetro dos planetas à realidade, uma vez que, sem este, os planetas tornar-se-iam demasiado grandes.

No que toca aos satélites naturais dos planetas, o diâmetro destes é dado pela seguinte fórmula:

$$x = \frac{d_{satelite}}{d_{planeta}}$$

Onde, o $d_{satelite}$ é o diâmetro real do satélite, e o $d_{planeta}$ é o diâmetro real do planeta do satélite, com a fração destes dois obtemos a escala do planeta satélite.

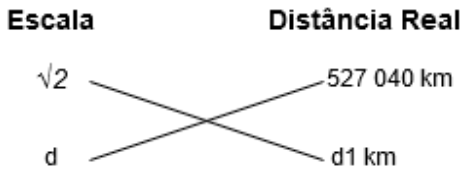
Na distância dos satélites naturais ao planeta, primeiro é necessário escolher um satélite natural, todos os outros são calculados tendo por base o primeiro satélite natural escolhido.

Tendo em conta o planeta de Saturno, e o satélite natural de **Reia**, possuímos os seguintes dados, o grupo estabeleceu uma translação no eixo x de 1 e no eixo y de 1 em relação ao planeta.

Portanto, possuímos os seguintes dados, a norma de Reia será de:

$$d = \sqrt{x^2 + y^2 + z^2} = \sqrt{1^2 + 1^2} = \sqrt{2}$$

Sendo possível estabelecer a seguinte regra de três simples:



Onde neste caso, a distância real de Reia a Saturno é de 527 040 quilómetros, de seguida, resolvendo a regra de três simples resolvida em ordem a d será de:

$$d = (\frac{\sqrt{2} * d_1}{527040})/1.5$$

Temos que, d_1 é a distância real do próximo satélite que pretendemos calcular em relação a apenas Saturno, e novamente, é necessário dividir por 1.5 porque, caso contrário, não se aproximava à realidade a escala calculada.

De seguida, a título exemplificativo, a escala utilizada para calcular o próximo satélite de Saturno, Titã, irá ter o seguinte funcionamento.

Primeiramente, utiliza-se a fórmula anterior, substituindo o d_1 pela distância real de Titã a Saturno:

$$d = (\frac{\sqrt{2} * 1222830}{527040})/1.5 = 2.185702$$

Portanto, neste ponto possuímos a norma de Titã em relação à Saturno. De seguida, necessitamos de conhecer tanto a coordenada x como a coordenada y de modo a efetuar o *translate* da esfera correspondente a Titã, como tal:

$$d = \sqrt{x^2 + y^2} \Leftrightarrow d^2 = x^2 + y^2 \Leftrightarrow 2.185702^2 = x^2 + y^2$$

Agora, é necessário atribuir um valor aleatório ou ao x ou ao y, contudo esse valor só poderá variar no seguinte intervalo de valores de [-2,2], tal que, por exemplo, atribuindo o valor de 1.5 ao x:

$$2.185702^2 = x^2 + y^2 \Leftrightarrow$$
$$2.185702^2 = 1.5^2 + y^2 \Leftrightarrow$$
$$y^2 = 2.185702^2 - 1.5^2 \Leftrightarrow$$
$$y = 1.589746$$

Portanto, agora possuímos ambos os valores x e y necessários para efetuar a *translation* da esfera representante de Titã para a posição correspondente a $x = 1.5$ e $y = 1.589746$, uma estimativa da escala mais próxima da realidade.

5 Representação do sistema solar

Após todos os processos anteriores, o resultado final obtido pelo grupo é o seguinte:

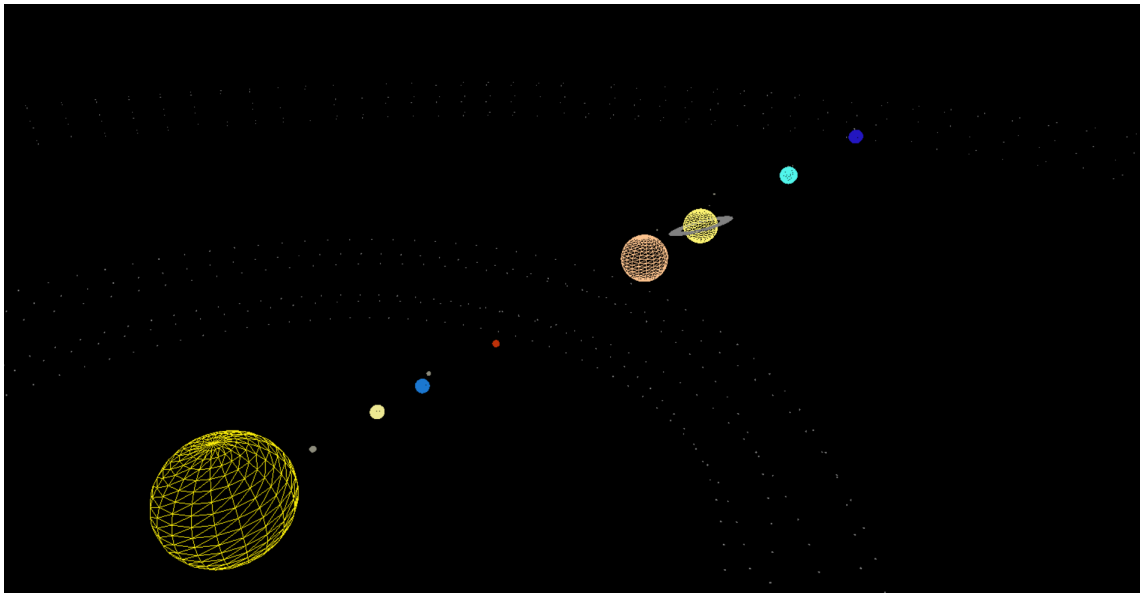


Figura 5.1: Sistema solar.

Onde são visíveis todos os planetas do sistema solar, construídos à escala, sendo que cada um deles possui alguns dos seus satélites, caso assim o possua na realidade, é possível também ver a cintura de asteróides, bem como a cintura de Kuiper.

Visto de outro ângulo:

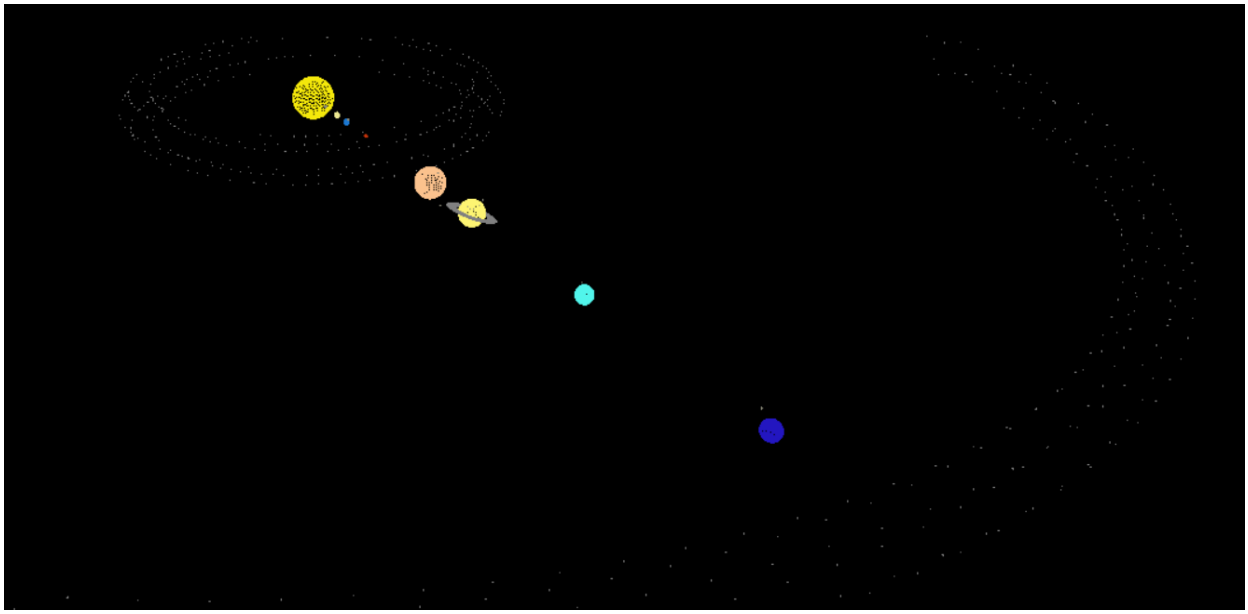


Figura 5.2: Sistema solar visto de outro ângulo.

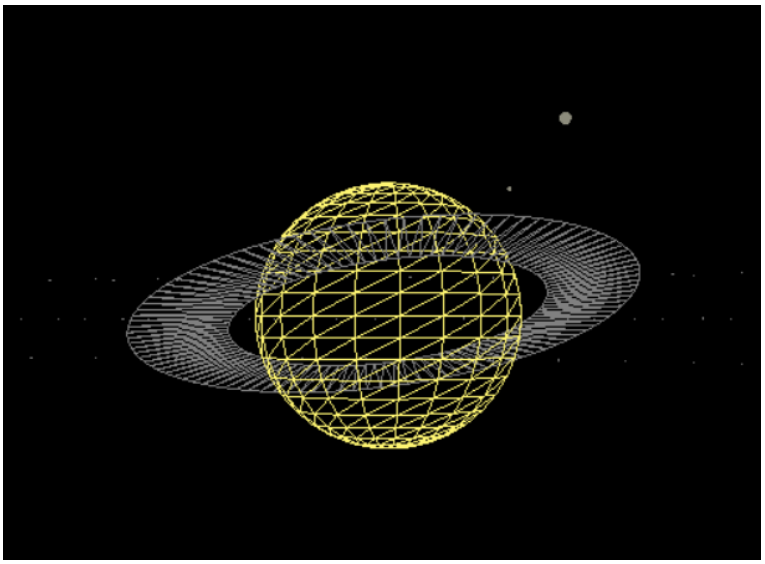


Figura 5.3: Saturno visto de uma perspetiva aproximada.

Por exemplo, dado o caso de Saturno, é possível ver o seu anel bem como os seus satélites naturais, Reia e Titã.

6 Conclusão

Nesta fase do projeto, o grupo de trabalho deparou-se com um problema que, em geral, era menos exigente do que o problema exigido na fase anterior não obstante, houve uma dificuldade acrescida no que toca ao estabelecimento de uma escala para o sistema solar, de modo a tornar este o mais real possível. Contudo, no que toca à modelação do trabalho, o tarefa desenvolvido anteriormente serviu como um base para esta fase e, os critérios pedidos para esta fase do projeto não foram tão exigentes como os da fase anterior.

Mesmo assim, o grupo de trabalho ganhou experiência, no que toca à elaboração de modelos exigentes, e com um maior grau de complexidade, ao se tornar necessário compreender o conceito da hierarquia ao construir as *scenes* e as consequentes transformações geométricas aplicadas aos elementos da *scene*..

Nas consequentes fases do projeto, é expectável que o modelo do sistema solar construído seja ainda melhor e ainda mais realista, o que é algo que motiva o grupo de trabalho.

7 Anexo

7.1 Sistema Solar XML

```
<?xml version="2.0" ?>
<scene>
        <group>
                <!-- SOL !-->
                <scale X="16.0" Y="16.0" Z="16.0"/>
                <colour X="0.956" Y="0.909" Z="0.043"/>
                <models>
                        <model file='../3dFiles/sphere.3d'/>
                </models>
                <group>
                        <!-- MERCURIO !-->
                        <translate X="1.5" Y="0" Z="0"/>
                        <scale X="0.05" Y="0.05" Z="0.05"/>
                        <colour X="0.552" Y="0.549" Z="0.486"/>
                        <models>
                                <model file = '../3dFiles/sphere.3d'/>
                        </models>
                </group>
                <group>
                        <!-- VENUS !-->
                        <translate X="2.8421" Y="0" Z="0"/>
                        <scale X="0.1244" Y="0.1244" Z= "0.1244"/>
                        <colour X="0.929" Y="0.909" Z="0.572"/>
                        <models>
                            <model file='../3dFiles/sphere.3d'/>
                        </models>
                </group>
                <group>
                        <!-- TERRA !-->
                        <translate X="3.9211" Y="0" Z="0"/>
                        <scale X="0.1310" Y="0.1310" Z= "0.1310"/>
                        <colour X="0.105" Y="0.466" Z="0.815"/>
                        <models>
                            <model file='../3dFiles/sphere.3d'/>
                        </models>
```

```
        <group>
                <!-- SATELITE - LUA !-->
                <translate X="1.4" Y="1.4" Z="0"/>
                <scale X="0.2727" Y="0.2727" Z="0.2727"/>
                <colour X="0.552" Y="0.549" Z="0.486"/>
                <models>
                        <model file = '../3dFiles/sphere.3d'/>
                </models>
        </group>
</group>
<group>
        <!-- MARTE !-->
        <translate X="6" Y="0" Z="0"/>
        <scale X="0.0695" Y="0.0695" Z= "0.0695"/>
        <colour X="0.745" Y="0.192" Z="0.035"/>
        <models>
            <model file='../3dFiles/sphere.3d'/>
        </models>
</group>
<group>
        <!-- Cintura de Asteroides !-->
        <rotate Angle="90" X="1" Y="0" Z="0"/>
        <scale X="1" Y="1" Z= "1"/>
        <models>
            <model file='../3dFiles/cintura.3d'/>
        </models>
</group>
<group>
        <!-- JUPITER !-->
        <translate X="12" Y="0" Z="0"/>
        <scale X="0.6294" Y="0.6294" Z= "0.6294"/>
        <colour X="0.984" Y="0.760" Z="0.552"/>
        <models>
            <model file='../3dFiles/sphere.3d'/>
        </models>
        <group>
                <!-- SATELITE - IO !-->
                <translate X="1.0" Y="1.0" Z="0"/>
                <scale X="0.025479" Y="0.025479" Z="0.025479"/>
                <colour X="0.552" Y="0.549" Z="0.486"/>
                <models>
                        <model file = '../3dFiles/sphere.3d'/>
                </models>
        </group>
         <group>
                <!-- SATELITE - EUROPA !-->
                <translate X="1.35965" Y="-1" Z="0"/>
                <scale X="0.0218318" Y="0.0218318" Z="0.0218318"/>
                <colour X="0.552" Y="0.549" Z="0.486"/>
                <models>
                        <model file = '../3dFiles/sphere.3d'/>
                </models>
        </group>
</group>
```

```
<group>
        <!-- SATURNO !-->
        <translate X="15.285" Y="0" Z="0"/>
        <scale X="0.5126" Y="0.5126" Z= "0.5126"/>
        <colour X="0.992" Y="0.956" Z="0.447"/>
        <models>
            <model file='../3dFiles/sphere.3d'/>
        </models>
        <group>
                <!-- SATELITE - REIA !-->
                <translate X="1.0" Y="1.0" Z="0"/>
                <scale X="0.013119" Y="0.013119" Z="0.013119"/>
                <colour X="0.552" Y="0.549" Z="0.486"/>
                <models>
                        <model file = '../3dFiles/sphere.3d'/>
                </models>
        </group>
        <group>
                <!-- SATELITE - TITA !-->
                <translate X="1.5" Y="1.589746" Z="0"/>
                <scale X="0.044224" Y="0.044224" Z="0.044224"/>
                <colour X="0.552" Y="0.549" Z="0.486"/>
                <models>
                        <model file = '../3dFiles/sphere.3d'/>
                </models>
        </group>
        <group>
                <!-- SATELITE - Anel !-->
                <rotate Angle="70" X="1" Y="0" Z="0"/>
                <scale X="1" Y="1" Z="1"/>
                <models>
                        <model file = '../3dFiles/torus.3d'/>
                </models>
        </group>
</group>
<group>
        <!-- URANO !-->
        <translate X="22.455" Y="0" Z="0"/>
        <scale X="0.31" Y="0.31" Z= "0.31"/>
        <colour X="0.317" Y="0.960" Z="0.921"/>
        <models>
            <model file='../3dFiles/sphere.3d'/>
        </models>
        <group>
                <!-- SATELITE - ARIEL !-->
                <translate X="1.0" Y="1.0" Z="0"/>
                <scale X="0.025666" Y="0.025666" Z="0.025666"/>
                <colour X="0.552" Y="0.549" Z="0.486"/>
                <models>
                        <model file = '../3dFiles/sphere.3d'/>
                </models>
        </group>
```

```
                        <group>
                                <!-- SATELITE - UMBRIEL !-->
                                <translate X="-0.5" Y="1.215517" Z="0"/>
                                <scale X="0.024913" Y="0.024913" Z="0.024913"/>
                                <colour X="0.552" Y="0.549" Z="0.486"/>
                                <models>
                                        <model file = '../3dFiles/sphere.3d'/>
                                </models>
                        </group>
                </group>
                <group>
                        <!-- NEPTUNO !-->
                        <translate X="30.75" Y="0" Z="0"/>
                        <scale X="0.3" Y="0.3" Z= "0.3"/>
                        <colour X="0.137" Y="0.086" Z="0.752"/>
                        <models>
                            <model file='../3dFiles/sphere.3d'/>
                        </models>
                        <group>
                                <!-- SATELITE - TRITAO !-->
                                <translate X="-1.4" Y="1.4" Z="0"/>
                                <scale X="0.059579" Y="0.059579" Z="0.059579"/>
                                <colour X="0.552" Y="0.549" Z="0.486"/>
                                <models>
                                        <model file = '../3dFiles/sphere.3d'/>
                                </models>
                        </group>
                </group>
                <group>
                        <!-- Cintura de Kuiper !-->
                        <rotate Angle="90" X="1" Y="0" Z="0"/>
                        <scale X="1" Y="1" Z= "1"/>
                        <models>
                            <model file='../3dFiles/cinturaFora.3d'/>
                        </models>
                </group>
        </group>
</scene>
```

Bibliografia

[1] NASA GOV, https://go.nasa.gov/2CBTOZ3

[2] Exploratorium, https://bit.ly/2Yh1cT7