Universidade do Minho
Escola de Engenharia

Computação Gráfica

2ª Fase

João Pedro da Santa Guedes A89588
Luís Pedro Oliveira de Castro Vieira A89601
Carlos Miguel Luzia Carvalho A89605
Bárbara Ferreira Teixeira A89610


A89588


A89601


A89605


A89610

4 de abril de 2021

Índice

1 Introdução

1.1 Contextualização

No ambito da Unidade Curricular de Computação Gráfica foi nos pedido o desenvolvimento de um projeto dividido em 4 fases, com o objetivo de através de ferramentas como o C++ e o OpenGL criar um mini mecanismo 3D baseado num cenário gráfico. O relatório presente é referente à segunda fase, na qual o objetivo é criar cenas hierárquicas usando transformações geométricas tendo como finalidade a criação de um modelo do Sistema Solar.

1.2 Resumo

Este relatório diz respeito à segunda fase do projeto prático desenvolvido na unidade curricular de Computação Gráfica. Tratando-se da segunda fase, várias funcionalidades implementadas na primeira fase foram mantidas e outras alteradas, de modo a melhor cumprirem os requisitos correspondentes a esta fase.

2 Arquitetura do Projeto

Sendo esta a segunda fase do projeto mantemos a estrutura desenvolvida na primeira fase, ou seja duas aplicações principais o gerador e o engine, sendo este último alvo de bastantes alterações nesta segunda fase atendendo aos requisitos a cumprir.

2.1 Aplicações

Uma vez que houve necessidade de alterar a estrutura dos ficheiros de configuração escritos em XML, foi também necessário alterar a forma como o engine processa esses mesmos ficheiros. Assim nesta secção serão apresentadas as aplicações fundamentais que permitem gerar os diferentes cenários pretendidos.

2.1.1 Gerador

generator.cpp - Como já enunciado no relatório da fase anterior, esta é a aplicação onde estão definidas as estruturações das diferentes formas geométricas, responsável pela geração dos seus respectivos vértices. Nesta segunda fase acrescentamos a primitiva Torus, sendo assim necessário para isso alterar o generator, porém tudo o resto se manteve idêntico ao desenvolvido na fase anterior sendo a única alteração ao generator.cpp este acrescento.

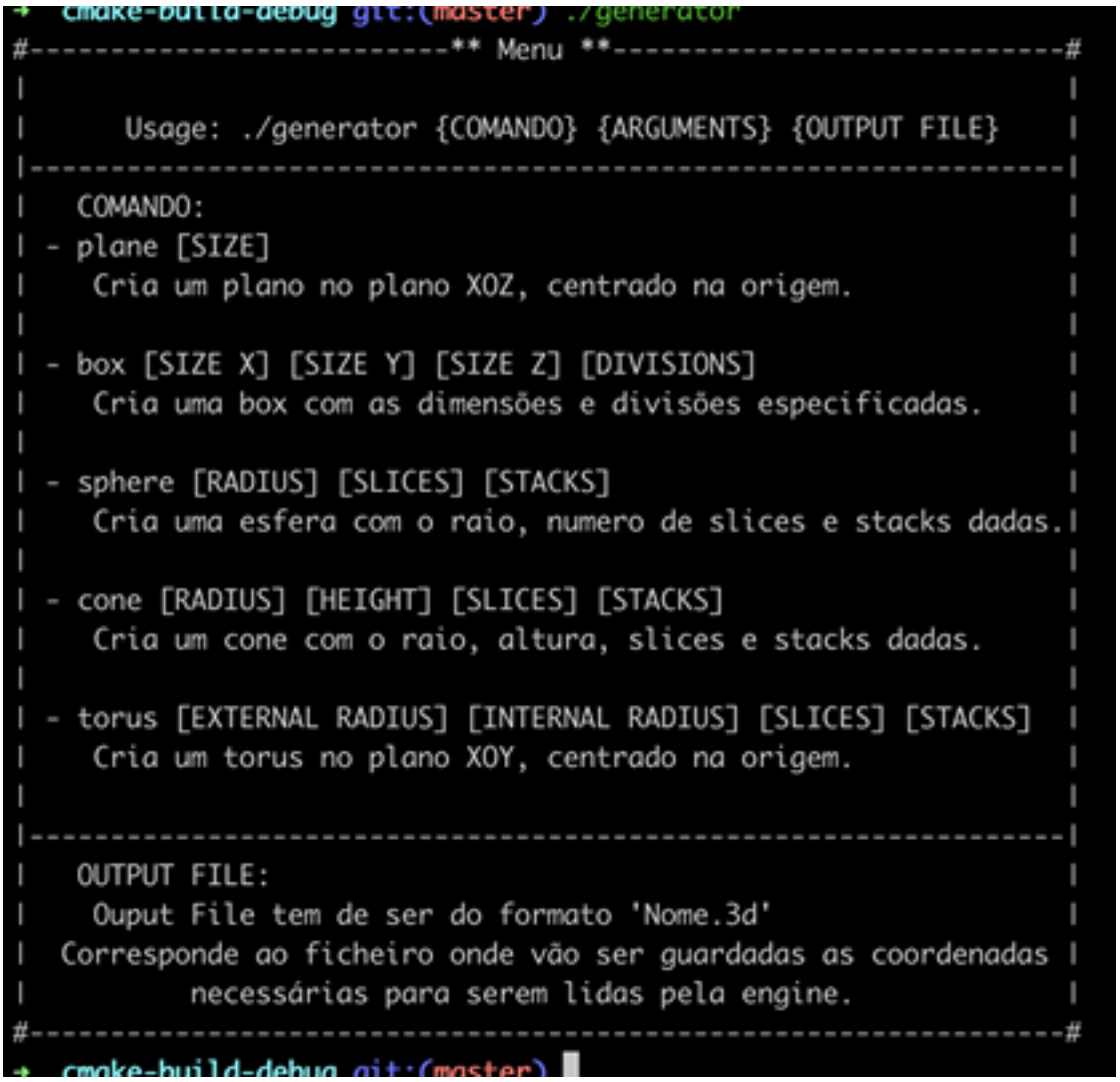

Figure 1: Menu de ajuda Generator

2.1.2 Engine

engine.cpp - Tal como dito na primeira fase esta é a aplicação que possui as funcionalidades principais. Permite a exibição e interação com os modelos. Com algumas alterações na estrutura do ficheiro XML, foi necessário alterar também o método de parsing (será explicado posteriormente). Assim, como passarão a existir grupos de primitivas com informações associadas, é necessário que o armazenamento da informação seja feito de maneira diferente, sendo esta renderizada pelo GLUT de uma forma diferente à fase anterior.

Figure 2: Menu de ajuda Engine

2.2 Classes

Para além das classes anteriormente criadas, o grupo decidiu criar 3 novas classes. Sendo destas uma para as transformações geométricas **transforms**, uma para a camera **camera** , e por último a classe **group** que irá armazenar um conjunto de formas, associando-as as respetivas transformações geométricas.

2.2.1 Transforms

transforms.h - Classe responsável por guardar as transformações realizadas num determinado modelo sendo assim necessária a existência de variáveis x,y,z

```
#ifndef GENERATOR_TRANSFORMS_H
#define GENERATOR_TRANSFORMS_H

#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <stdlib.h>
#include <GL/glut.h>
#endif

class Transform{
    float x_value;
    float y_value;
    float z_value;

    public:
        Transform();
        Transform(float x, float y, float z);
        float getX();
        float getY();
        float getZ();
        virtual void execute(){ };

};

class Translation : public Transform{
public:
    Translation();
    Translation(float x, float y, float z);
    void execute();
};

class Rotation : public Transform{
    float angle;
public:
    Rotation();
    Rotation(float a, float x, float y, float
    void execute();
};

class Scale : public Transform{
public:
    Scale();
    Scale(float x, float y, float z);
    void execute();
};

#endif //GENERATOR_TRANSFORMS_H
```

Figure 3: transforms.h

2.2.2 Camera

camera.h - Classe responsável por guardar as informações referentes a camera como as reacções a eventos.

```
#ifndef GENERATOR_CAMERA_H
#define GENERATOR_CAMERA_H
#define _USE_MATH_DEFINES
#include "point.h"

class Camera{
    private:
        Point* position;
        Point* direction;
        float yaw;
        float pitch;
        float speed;
        float rotationSpeed;
        float alphaStatic;
        float betaStatic;
    public:
        Camera();
        Point* getPosition();
        Point* getDirection();
        Point* getFocus();
        void moveFoward();
        void moveBackwards();
        void moveLeft();
        void moveRight();
        void turn(float,float);
        void turnStatic(unsigned char);
        Point* getStaticPosition();
};

#endif //GENERATOR_CAMERA_H
```

Figure 4: camera.h

2.2.3 Group

group.h - Classe cuja função é armazenar toda a informação referente a um determinado grupo. Esta é utilizada aquando a leitura dos ficheiros XML, na medida em que a cada grupo lido e interpretado, corresponde um objeto com informações relativas ás formas e ou modelos, ás transformações a que os modelos são sujeitos e ainda às mudanças de cor destes. Poderá também conter os grupos filhos de um determinado grupo, sendo estes grupos contidos neste grupo.

```
#ifndef GENERATOR_GROUP_H
#define GENERATOR_GROUP_H

#include <vector>
#include "transforms.h"
#include "figure.h"

class Group {
    std::vector<Transform*> transforms;
    std::vector<Figure*> figures;
    std::vector<Group*> childs;
    float R, G, B;

    public:
        Group();
        Group(std::vector<Transform*> t, std::vector<Figure*> f, std::vector<Group*> c);
        std::vector<Transform*> getTransforms();
        std::vector<Figure*> getFigures();
        std::vector<Group*> getChilds();
        void pushTransform(Transform* t);
        void pushFigure(Figure* f);
        void pushGroup(Group* g);
        float getR();
        float getG();
        float getB();
        void setRGB(float,float,float);
};

#endif //GENERATOR_GROUP_H
```

Figure 5: group.h

2.3 Ficheiros Auxiliares

2.3.1 Figure

figure.h - Classe responsável por armazenar os dados relativos a cada figura, sendo estes posteriormente utilizados pelo engine para desenhar as figuras.

```
#ifndef GENERATOR_FIGURES_H
#define GENERATOR_FIGURES_H

#include <vector>
#include "point.h"
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

class Figure{
    private:
        std::vector<Point*> vertexes;

    public:
        void pushVertex(Point* v);
        int getSize();
        std::vector<Point*> getVertexes();
        void draw(GLenum , float ,float,float);
};

#endif //GENERATOR_FIGURES_H
```

Figure 6: figure.h

2.3.2 Parser

parser.h - Classe responsável por dar parse, com o auxílio do tinyxml2, aos ficheiros que servem de input ao engine, e que utiliza como forma de guardar os dados a classe group, figure e transforms.

```
#ifndef GENERATOR_PARSER_H
#define GENERATOR_PARSER_H

#include "tinyxml2.h"
#include <string>
#include <regex>
#include <fstream>
#include <sstream>
#include "point.h"
#include <iostream>
#include "group.h"


int readXML(string file, Group* group);

#endif //GENERATOR_PARSER_H
```

Figure 7: parser.h

2.3.3 Tinyxml

tinyxml2 - Ferramenta utilizada para auxiliar o parsing dos ficheiros XML.

3 Generator

As responsabilidades do gerador mantêm-se relativamente à fase anterior, ou seja, o gerador continua a ser responsável por gerar os ficheiros que contêm informação sobre as primitivas geométricas a desenhar. No entanto, nesta fase, foi acrescentada uma nova primitiva, o *Torus*, o que faz com que o gerador esteja, agora, apto a gerar 6 primitivas geométricas.

3.1 Primitiva Geométrica: *Torus*

Um *Torus* é um sólido geométrico que se assemelha a um donut. Em geometria, pode ser definido como o lugar geométrico tridimensional formado pela rotação de uma superfície circular plana de raio r, em torno de uma circunferência de raio R.

3.2 Algoritmo

Como podemos verificar pela figura 1, a construção do torus baseia-se nos raios exterior e interior. Para tal, definimos os eixos X e Y como responsáveis para definir a circunferência com o raio exterior ***distance***, e os eixos X e Z definem uma circunferência com o raio interior ***radius***.

Facilmente conseguimos chegar ao ângulo correspondente ao shift de cada ciclo pelas seguintes expressões:

$$shift\_\phi = \frac{\pi}{stacks}$$

$$shift\_\theta = \frac{\pi}{slices}$$

Olhando para a circunferência interior, conseguimos perceber que o valor da coordenada X (parte positiva) é dada pelo raio exterior, somado a um valor desconhecido.

Esse valor, é:

- máximo (positivo): quando o ângulo ϕ é $0^{\underline{o}}$
- 0 : quando o ângulo ϕ é $90^{\underline{o}}$ / $270^{\underline{o}}$
- mínimo (negativo) : quando o ângulo ϕ é $180^{\underline{o}}$

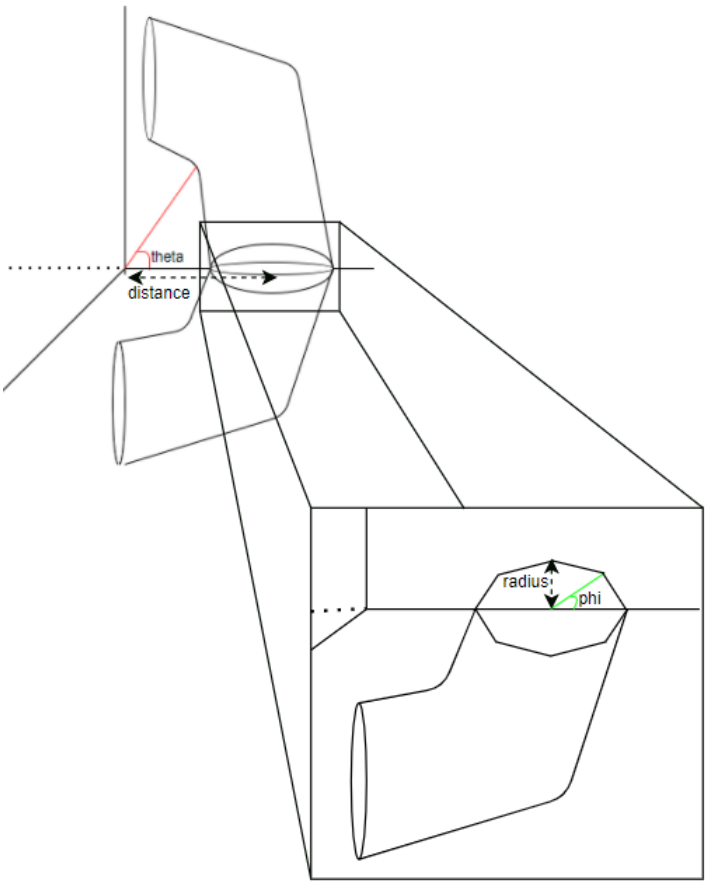

Figure 8: Representação do Torus

Ora, atendendo aos requisitos, podemos afirmar que :

$$\boldsymbol{x'} = radius * cos(\phi)$$

$$\boldsymbol{x} = distance + x'$$

$$\boldsymbol{x} = distance + radius * cos(\phi)$$

Além disso, também conseguimos calcular o valor de Z, que vai ser equivalente à "altura", sendo dado pelo **sin**

$$\boldsymbol{z} = radius * sin(\phi)$$

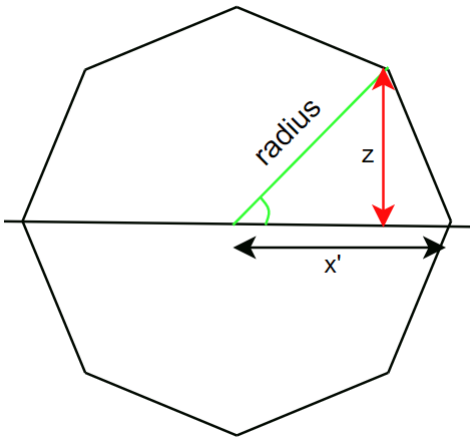

Figure 9: Circunferência interna

Para calcular a componente do Y, já tem de ser uma equação que vai depender tanto do ϕ como do θ:

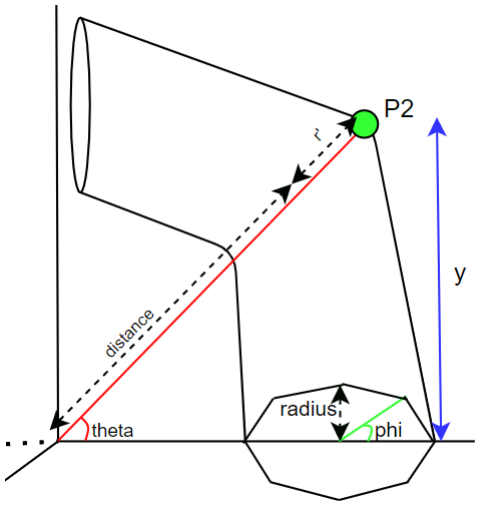

Figure 10: Demonstração do Y

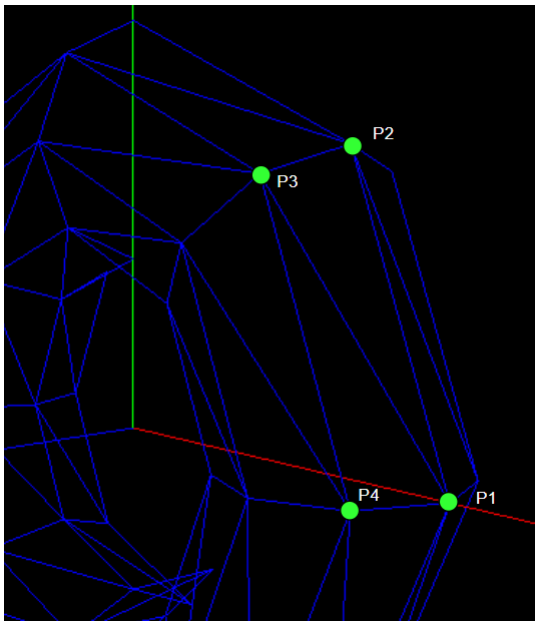

Figure 11: Exemplo de pontos a desenhar

Como no exemplo que estávamos a dar, a componente Y era obviamente

0, provavelmente não dava para observar alguns aspetos importantes. Assim, mudando o ponto para o ponto P2 (fig 3.), podemos verificar algumas coisas quanto à componente Y:

- A coordenada Y pode ser calculada através do sin(θ) a multiplicar pela hipotenusa
- A hipotenusa é dada pela ***distance*** + r'
- Vendo pela figura 2, o r' volta a corresponder a **radius * cos(ϕ)**
- Assim concluímos que a componente Y é dada por:

$$y = sin(\theta) * (distance + radius * cos(\phi))$$

Agora chegámos à parte de "por que ordem desenhar os pontos" e "como chegar de um ponto para outro", onde temos a figura 4 para auxiliar. Pela regra da mão direita, sabemos que temos de desenhar 2 triângulos, o primeiro sendo dado por exemplo por: P1, P2 e P3, seguido por P3, P4 e P1.

Pela explicação acima, conseguimos entender que o ponto P1 é composto por:

- x = cos(θ) * (distance + radius * cos(ϕ))
- y = sin(θ) * (distance + radius * cos(ϕ))
- z = radius * sin(ϕ)

Além disso sabemos o ângulo que temos de nos mover ao longo das slices (shift_θ) e das stacks (shift_ϕ).

Como neste caso, o ponto P2 está exatamente no mesmo plano XOY do P1, então só teremos de fazer alterações ao ângulo correspondente ao ângulo externo que neste caso é o θ, adicionando simplesmente ao ângulo já existente o valor do shift_θ, pois só é esse ângulo que altera.

Assim P2 é dado por:

- x = cos(θ + shift_θ) * (distance + radius * cos(ϕ))
- y = sin(θ + shift_θ) * (distance + radius * cos(ϕ))
- z = radius * sin(ϕ)

Seguindo esta lógica, também conseguimos chegar ao P3 através do ponto P2. Pela figura percebemos que o P3 avançou o ângulo correspondente ao shift das stacks, tendo apenas de adicionar aos ângulos que utilizam o ϕ, já existentes o valor do shift correspondente.

Assim, P3 é dado por:

- x = cos(θ + shift_θ) * (distance + radius * cos(ϕ + shift_ϕ))
- y = sin(θ + shift_θ) * (distance + radius * cos(ϕ + shift_ϕ))
- z = radius * sin(ϕ + shift_ϕ)

Para desenhar o outro triângulo, só falta as componentes do P4, visto que já temos tanto o ponto P1 como o ponto P3. Seguindo toda a lógica que foi descrita em cima, o P4 pode ser obtido através de uma rotação do eixo Y, do ângulo ϕ de valor shitf_ϕ Assim, conclui-se que o ponto P4 é dado por:

- x = cos(θ) * (distance + radius * cos(ϕ + shitf_ϕ))
- y = sin(θ) * (distance + radius * cos(ϕ + shitf_ϕ))
- z = radius * sin(ϕ + shitf_ϕ)

Posto isto, o Torus pode ser conseguido, com 2 loops (de 0 a stacks e outro de 0 a slices), desenhando cada quadrado de cada vez, incrementando o valor do θ e do ϕ por shift_θ e shift_ϕ unidades, respetivamente, por ciclo.

4 Engine

Na primeira fase, o funcionamento deste era simples, reconhecer e apresentar o conteúdo dos ficheiros modelo presente no ficheiro de configuração. Na segunda fase foram feitas alterações, sendo agora possível renderizar tanto o conteúdo dos ficheiros modelo como as respetivas transformações geométricas associadas a estes, ou até as cores dos mesmos, apresentando-as como um cenário ao utilizador.

4.1 Leitura

Todo o processo de leitura é auxiliado pelo **Parser.h**. A função principal, trata de percorrer o ficheiro xml e, dependendo do elemento que encontrar, trata-o de uma maneira específica. Esta função é a responsável por criar as estruturas apropriadas, para depois conseguir construir o modelo constituído pelo ficheiro xml. Trata-se de uma função recursiva que recebe, a cada chamada, o elemento do XML que se encontra a explorar, e a estrutura de dados onde toda a informação recolhida vai ser guardada.

- Se o elemento corresponder a uma transformação (rotação, translação, ou escala), é feito o parsing dos valores, cria-se a estrutura apropriada e é inserida num vetor de transformações que Group possui.

- Se o elemento for equivalente a um ***colour***, é dado o parse dos valores do RGB e é então inserido na estrutura Group, nos valores R, G e B respetivos.

- Caso o elemento seja do tipo ***models***, equivalentemente ao que acontece com as transformações, é criado um objeto da classe **Figure**, que possui um vetor de pontos, onde vão ser guardados todos os pontos que contem o ficheiro *.3d* relativo à figura. Finalmente, a figura é inserida no Group com que foi inicializado a função.

- Finalmente, quando o *element* corresponde a um ***group***, é criado um *group child*, que é inserido no group que foi recebido como argumento, e é chamada a função recursivamente, passando como argumento o *child* que foi criado.

Com este parse, é possível tratar o XML de forma hierárquica, tal como é suposto.

4.2 Estruturas de Dados

Como já explicado no segmento anterior, a estrutura de dados necessária para armazenar toda a informação recolhida depois do *parsing* é o *Group*. Desta forma, numa perspectiva de facilitar a execução das transformações e o desenho das figuras, são armazenados em vetores as transformações de cada grupo, e as respetivas figuras (os pontos necessários à formação das mesmas).

Tendo também em conta a importância hierárquica dos *groups* no ficheiro de configuração, foi também necessário a criação de um vetor de grupos filhos. Por fim, relativamente as cores, são armazenadas somente no group a que pertencem, não necessitando de mais nenhuma estrutura de dados.

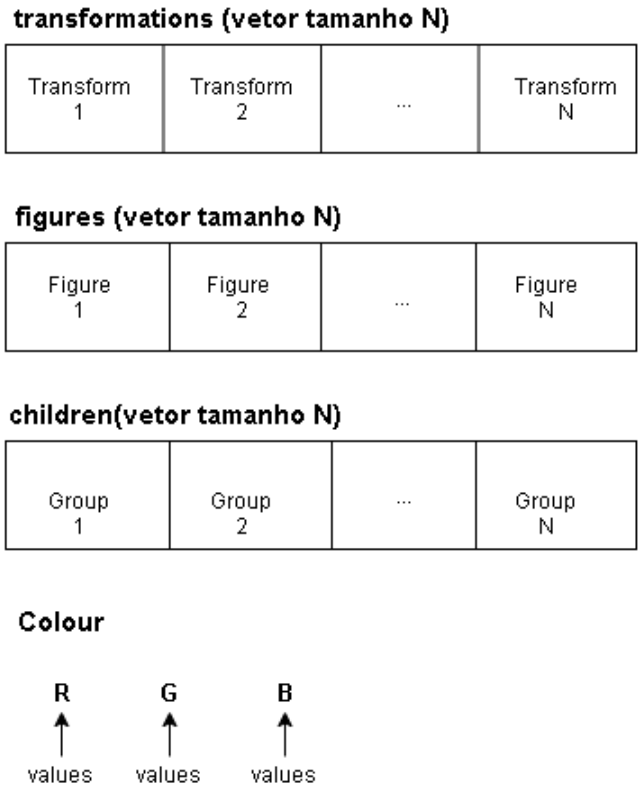

Figure 12: Representação estrutura de dados de cada Group

4.3 Renderização

O primeiro grupo criado após o parsing do ficheiro de configuração corresponde ao elemento scene do XML, sendo este como uma variável global de toda a aplicação uma vez que é através desta que conseguimos alcançar todos os restantes grupos (grupos-filhos).

A função responsável pela renderização do conteúdo tem o nome *renderScene* sendo muito semelhante à apresentada na 1º fase, porém faz a invocação à função *render* que vai percorrer todas as transformações e figuras a desenhar. A função *render* recebe um único argumento do tipo *Group** sendo que na primeira iteração, este corresponde à variável global *scene* referida em cima. Uma vez que serão efetuadas transformações geométricas, ou seja a matriz de transformação será alterada, é necessário primeiro que seja guardado o estado inicial e depois de efetuar as alterações este seja reposto, explicando assim a necessidade dos métodos *glPushMatrix()* e *glPopMatrix()*. A função começa por aplicar as várias transformações que possam haver, seguida da obtenção da cor que foi dada ao grupo e, finalmente, o desenho das figuras que fazem parte do *group*. Por último, como cada *group* podem conter outros grupos, é feita a renderização desses mesmos por recursividade.

5 Apresentação dos Modelos

De uma perspetiva geral, consideramo-nos satisfeitos com o resultado final desta fase, dado que cumprimos com os requisitos estabelecidos inicialmente. Tentamos representar o sistema solar o mais realista possível relativamente a escalas e até às cores escolhidas.

5.1 Visualização

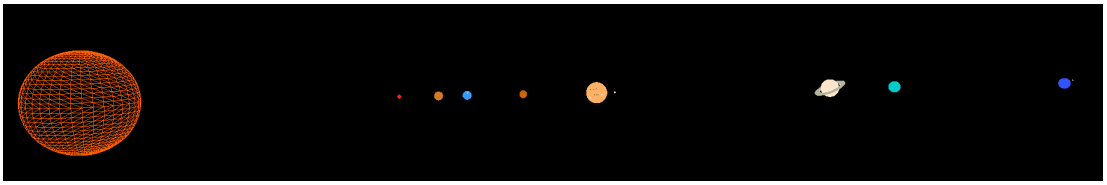

Figure 13: Sistema Solar completo

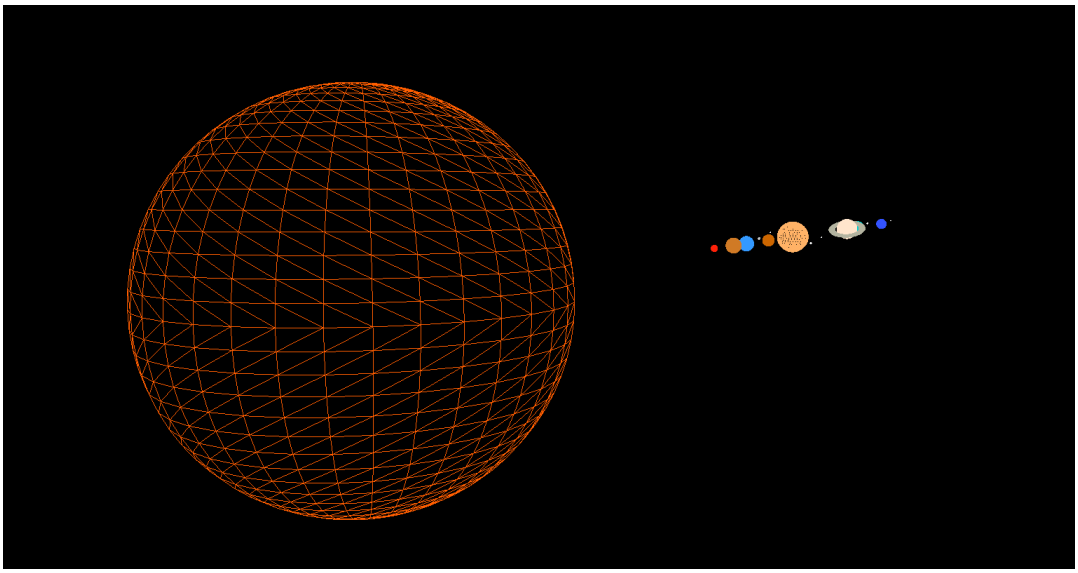

Figure 14: Sistema Solar visto do Sol

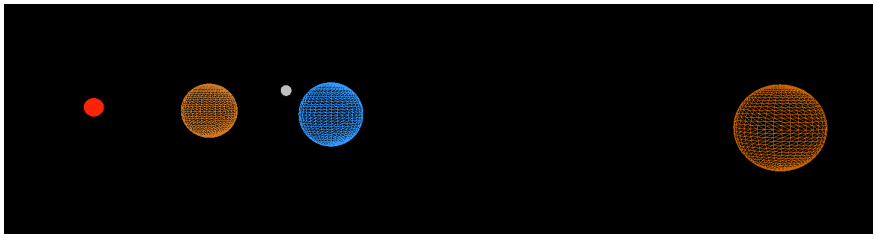

Figure 15: Visualização dos primeiros 4 planetas

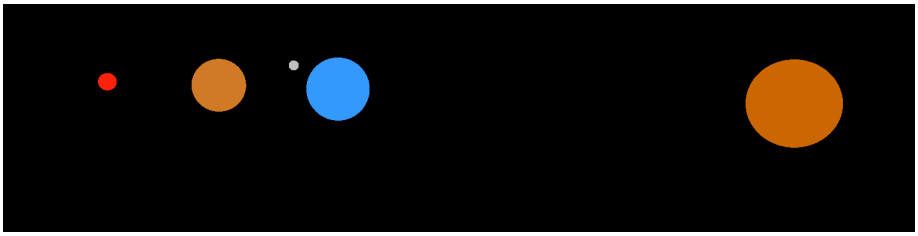

Figure 16: Visualização dos primeiros 4 planetas com preenchimento

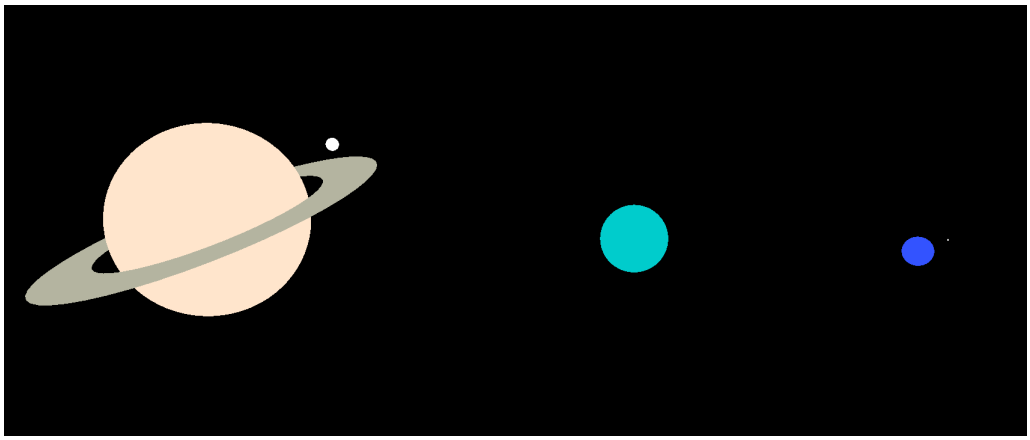

Figure 17: Visualização dos últimos 3 planetas com preenchimento

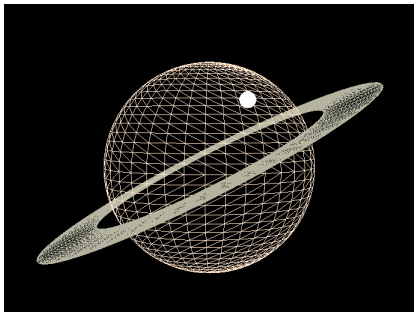

Figure 18: Visualização de Saturno

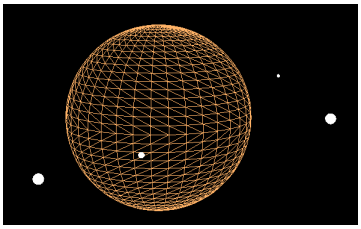

Figure 19: Visualização de Jupiter com linhas

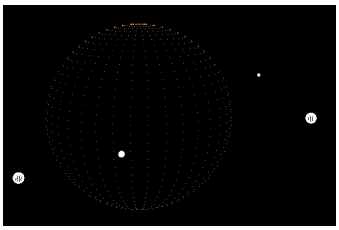

Figure 20: Visualização de Jupiter com pontos

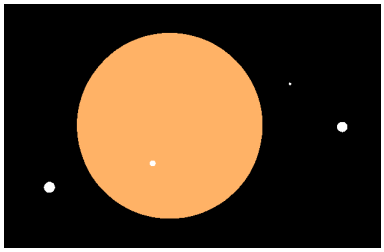

Figure 21: Visualização de Jupiter preenchido

6 Extras

6.1 Alternar entre câmara estática e FPS

Tendo diferentes utilidades e utilizações, a câmara que apresentamos no nosso trabalho pode ser alternada entre estática e FPS através de um clique com o botão direito do rato no menu de apresentação gráfico, permitindo uma rápido e fácil alternativa entre os dois modos. Para além disso possibilita ainda sair do programa e fechá-lo completamente

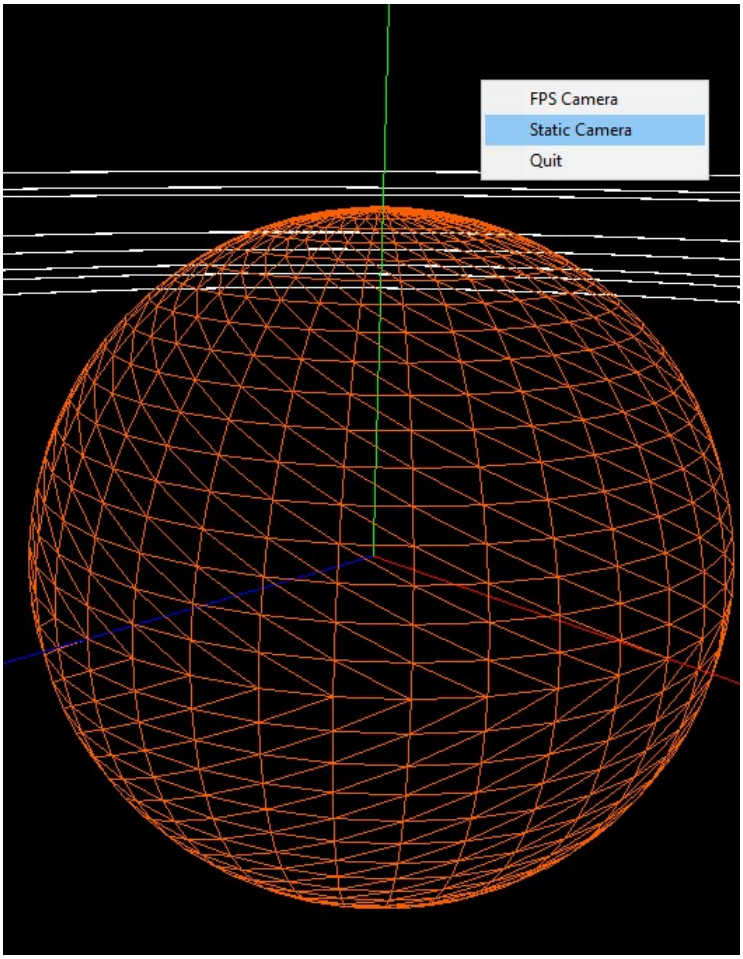

Figure 22: Menu da Câmara

7 Conclusão e Trabalho Futuro

Considerámos, em geral, que esta fase foi menos trabalhosa, isto deve-se, principalmente, ao facto da execução da primeira fase nos ter permitido adquirir as bases necessárias para facilitar a resolução de eventuais problemas nesta fase.

A nossa satisfação em relação a esta fase, não significa que não tenhamos grandes expectativas para as próximas fases. Após a realização de mais uma fase, considerámos que esta é imprescindível para a restante realização do projeto e considerámos que possuímos ainda mais conhecimento, o que permite que as próximas fases sejam desempenhadas com ainda mais eficiência.

Posto isto, julgamos ter cumprido com os requisitos desta fase e esperamos melhorar cada vez mais este projeto de forma a torná-lo sucessivamente mais realista e visualmente mais apelativo.

8 Anexos

8.1 Ficheiro de configuração do Sistema Solar

```
<scene>
        <!-- SUN  -->
        <group>
                <scale X='5' Y='5' Z='5' />
                <colour R='255' G='94' B='0' />
                <models>
                        <model file='sphere.3d'/>
                </models>
        </group>
        <!-- ORBITS -->

        <!-- MERCURY'S ORBIT -->
        <group>
                <rotate angle='90' axisX='1' />
                <scale X='29.8775' Y='29.8775' Z='29.8775'/>
                <colour R='255' G='255' B='255' />
                <models>
                        <model file='orbit.3d' />
                </models>
        </group>
        <!-- VENUS'S ORBIT -->
        <group>
                <rotate angle='90' axisX='1' />
                <scale X='33.9233' Y='33.9233' Z='33.9233'/>
                <colour R='255' G='255' B='255' />
                <models>
                        <model file='orbit.3d' />
                </models>
        </group>
        <!-- EARTH'S ORBIT -->
        <group>
                <rotate angle='90' axisX='1' />
                <scale X='36.9233' Y='36.9233' Z='36.9233'/>
                <colour R='255' G='255' B='255' />
                <models>
                        <model file='orbit.3d' />
                </models>
        </group>
        <!-- ASTEROID BELT -->
```

```
<!-- MARS'S ORBIT -->
<group>
        <rotate angle='90' axisX='1' />
        <scale X='42.9523' Y='42.9523' Z='42.9523'/>
        <colour R='255' G='255' B='255' />
        <models>
                <model file='orbit.3d' />
        </models>
</group>
<!-- JUPITER'S ORBIT -->
<group>
        <rotate angle='90' axisX='1' />
        <scale X='51.12987' Y='51.12987' Z='51.12987'/>
        <colour R='255' G='255' B='255' />
        <models>
                <model file='orbit.3d' />
        </models>
</group>
<!-- SATURN'S ORBIT -->
<group>
        <rotate angle='90' axisX='1' />
        <scale X='79.5005' Y='79.5005' Z='79.5005'/>
        <colour R='255' G='255' B='255' />
        <models>
                <model file='orbit.3d' />
        </models>
</group>
<!-- URANU'S ORBIT -->
<group>
        <rotate angle='90' axisX='1' />
        <scale X='88.0747' Y='88.0747' Z='88.0747'/>
        <colour R='255' G='255' B='255' />
        <models>
                <model file='orbit.3d' />
        </models>
</group>
```

```
<!-- NEPTUN'S ORBIT -->
<group>
        <rotate angle='90' axisX='1' />
        <scale X='112.35529' Y='112.35529' Z='112.35529'/>
        <colour R='255' G='255' B='255' />
        <models>
                <model file='orbit.3d' />
        </models>
</group>

<!-- PLANETS AND SATELITES -->

<!-- MERCURY -->
<group>
        <translate X='29.8775' />
        <scale X='0.17625' Y='0.17625' Z='0.17625'/>
        <colour R='255' G='33' B='9' /> <!-- red -->
        <models>
                <model file='sphere.3d' />
        </models>
</group>
<!-- VENUS -->
<group>
        <translate X='33.9233' />
        <scale X='0.4395' Y='0.4395' Z='0.4395'/>
        <colour R='208' G='122' B='37' />
        <models>
                <model file='sphere.3d' />
        </models>
</group>
<!-- EARTH -->
<group>
        <translate X='36.9233' />
        <colour R='51' G='153' B='255' />
        <scale X='0.4425' Y='0.4425' Z='0.4425'/>
        <models>
                <model file='sphere.3d' />
        </models>
```

```
        <!-- MOON -->
        <group>
                <translate Y='0.75' Z='1.75' />
                <colour R='192' G='192' B='192' />
                <scale X='0.15' Y='0.15' Z='0.15'/>
                <models>
                        <model file='sphere.3d' />
                </models>
        </group>
</group>

<!-- MARS -->
<group>
        <translate X='42.9523' />
        <scale X='0.1860' Y='0.1860' Z='0.1860'/>
        <colour R='204' G='102' B='0' />
        <models>
                <model file='sphere.3d' />
        </models>
</group>
<!-- JUPITER -->
<group>
        <translate X='51.12987' />
        <scale X='1.125' Y='1.125' Z='1.125'/>
        <colour R='255' G='178' B='102' />
        <models>
                <model file='sphere.3d' />
        </models>

        <!--SATELITE N.-->
        <group>
                <translate X='0' Y='0' Z='1.875' />
                <colour R='255' G='255' B='255' />
                <scale X='0.025' Y='0.025' Z='0.025' />
                <models>
                        <model file='sphere.3d' />
                </models>
        </group>
```

```
        <!--SATELITE N.-->
        <group>
                <translate X='1.425' Y='0.25' Z='-1.8074' />
                <colour R='255' G='255' B='255' />
                <scale X='0.0175' Y='0.0175' Z='0.0175' />
                <models>
                        <model file='sphere.3d' />axis
                </models>
        </group>
        <!--SATELITE N.-->
        <group>
                <translate X='-1.00875' Y='-0.35' Z='1.3666' />
                <colour R='255' G='255' B='255' />
                <scale X='0.05' Y='0.05' Z='0.05' />
                <models>
                        <model file='sphere.3d' />
                </models>
        </group>
        <!--SATELITE N.-->
        <group>
                <translate X='1.85175' Y='0' Z='0' />
                <colour R='255' G='255' B='255' />
                <scale X='0.055' Y='0.055' Z='0.055' />
                <models>
                        <model file='sphere.3d' />
                </models>
        </group>

</group>
<!-- SATURN-->
<group>
        <translate X='79.5005' />
        <scale X='0.9975' Y='0.9975' Z='0.9975'/>
        <colour R='255' G='229' B='204' />
        <models>
                <model file='sphere.3d' />
        </models>
```

```
        </models>
        <!-- SATURN'S RINGS -->
        <group>
                <rotate  angle='90' axisX='1' />
                <group>
                        <rotate angle='27' axisY='1' />
                        <scale X='1.5' Y='1.5' />
                        <colour R='180' G='180' B='160' />
                        <models>
                                <model file='ring.3d' />
                        </models>
                </group>
        </group>
        <!--SATELITE N.-->
        <group>
                <translate X="0" Y="0.6" Z="2..275" />
                <colour R="255" G="255" B="255" />
                <scale X="0.0575" Y="0.0575" Z="0.0575" />
                <models>
                        <model file="sphere.3d" />
                </models>
        </group>
</group>
<!-- URANUS -->
<group>
        <translate X='88.0747' />
        <scale X='0.645' Y='0.645' Z='0.645'/>
        <colour R='0' G='204' B='204' />
        <models>
                <model file='sphere.3d' />
        </models>
</group>
<!-- NEPTUN -->
<group>
        <translate X='112.35529' />
        <scale X='0.6375' Y='0.6375' Z='0.6375'/>
        <colour R='51' G='83' B='255' />
        <models>
                <model file='sphere.3d' />
```

```
                </models>
                <!--SATELITE N.-->
                <group>
                        <translate X="0" Y="0.75" Z="1.85" />
                        <colour R="255" G="255" B="255" />
                        <scale X="0.0575" Y="0.0575" Z="0.0575" />
                        <models>
                                <model file="sphere.3d" />
                        </models>
                </group>
        </group>
</scene>
```