Universidade do Minho

Mestrado Integrado de Engenharia Informática

Transformações Geométricas

Fase II

Autores:

Sara Pereira	A73700
Filipe Fortunato	A75008
Frederico Pinto	A73639

7 de Abril de 2018

Conteúdo

1 Introdução **2**

2 Transformações Geométricas **2**
2.1 Rotação 2
2.2 Translação 3
2.3 Escala 4

3 Arquitetura do Código **5**
3.1 Aplicações 5
3.1.1 Gerador 5
3.2 Motor 5
3.3 Classes 6
3.3.1 Rotação 6
3.3.2 Translação 6
3.3.3 Escala 7
3.3.4 Cor 7
3.3.5 Transformação 8
3.3.6 Transforms 9
3.3.7 Ponto 9

4 Gerador **10**
4.1 Torus 10
4.2 Algoritmo 10

5 Motor **13**
5.1 Leitura XML 13

6 Processo de Renderização **14**

7 Análise de Resultados **14**
7.1 Visualização 15

8 Conclusão **18**

9 Anexos **19**

1 Introdução

Após a conclusão da primeira fase do projeto, *Primitivas Gráficas*, é agora o momento de continuar com o projeto e iniciar a segunda fase. Esta, tem como objetivo efetuar alterações no trabalho desenvolvido na primeira fase, de maneira acrescentar transformações geométricas.

Como transformações geométricas presentes nesta fase, temos as rotações, translações e escalamento. Estas serão responsáveis pelo modo como as primitivas desenvolvidas anteriormente serão exibidas.

Todo o trabalho desenvolvido nesta fase, será aplicado a um modelo do Sistema Solar que incluirá o Sol, os Planetas e as respetivas Luas.

2 Transformações Geométricas

2.1 Rotação

Rotação de uma figura geométrica implica a fixação de, pelo menos, um ponto num eixo à escolha e a rotação dá-se em torno desse ponto, não havendo alteração na figura em si. Assim, um exemplo de uma rotação no eixo **Z**, de um objeto com coordenadas **(x,y,z)**, sendo as novas coordenadas **(x',y',z')**:

```
x' = x * cos(angle) + y * sin(angle)
y' = x * sin(angle) + y * cos(angle)
z' = z
```

Sendo angle o ângulo que desejamos para a rotção pretendida. Em termos de definição matricial:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Figura 1: Matriz relativa à rotação sobre o eixo Z

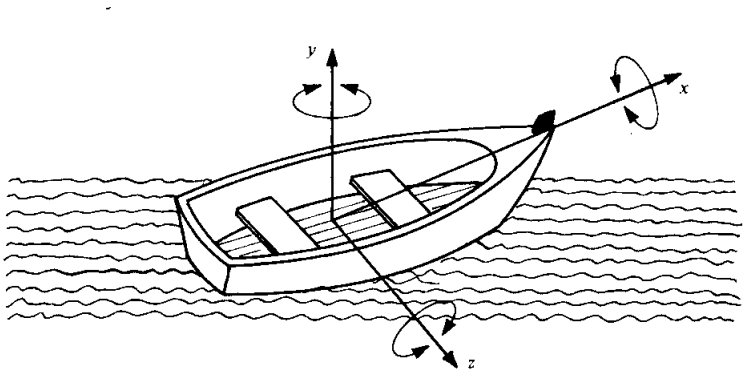

Figura 2: Exemplo de rotação sobre os diferentes eixos

2.2 Translação

Translação desloca um objeto segundo uma direção, sentido e comprimento (vetor), mais uma vez, sem alteração das dimensões do objeto. Assim, se as coordenadas de um objeto forem **(x,y,z)** e o vetor de translação for **(tx, ty, tz)**, as novas coordenadas **(x',y',z')** serão:

```
x' = x + tx
y' = y + ty
z' = z + tz
```

Em termos matriciais:

$$\begin{bmatrix} x_T \\ y_T \\ z_T \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Figura 3: Definição matricial para a translação

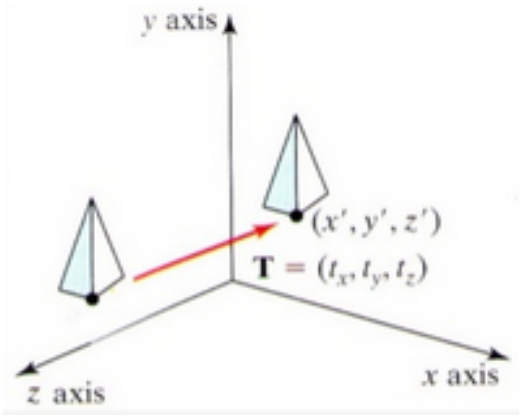

Figura 4: Exemplo de uma translação

2.3 Escala

Normalmente, define-se escala como a razão entre as medidas reais de um objeto e as medidas de desenho. No âmbito desta UC, a aplicação de um escalamento numa figura altera o tamanho da mesma, havendo, assim, multiplicação das coordenadas do mesmo por fatores **sx, sy, sz**. Assim, um objeto que tenho como coordenadas **(x,y,z)**, após a aplicação de uma escala passa a ter as seguintes coordenadas **(x',y',z')**:

```
x' = x * sx
y' = y * sy
z' = z * sz
```

Em termos matriciais:

$$\begin{bmatrix} x_s \\ y_s \\ z_s \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Figura 5: Definição matricial para a escala

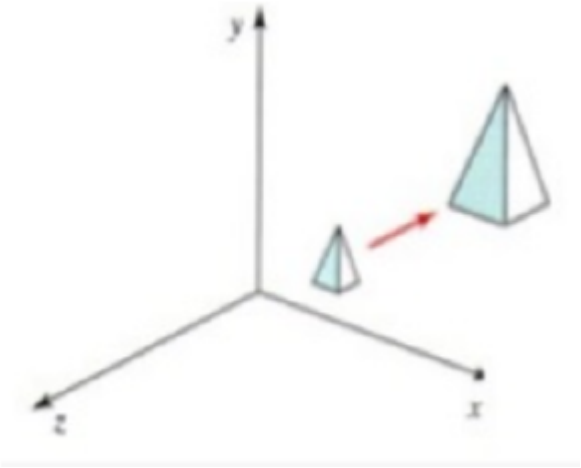

Figura 6: Exemplo de alteração da escala de um objeto

3 Arquitetura do Código

O desenvolvimento desta fase do projeto manteve a base do anterior, tendo sido feitas as alterações necessárias a cada uma das seguintes aplicações de maneira a serem cumpridos todos os requisitos.

3.1 Aplicações

Neste tópico serão apresentadas todas as composições feitas às duas principais aplicações do projeto. O grupo achou esta informação indispensável uma vez que houve completa reconstrução do ficheiro **XML** e, por consequência, alteração do motor de processamento do ficheiro. Ao mesmo tempo, também foram feitas alterações mínimas no gerador de primitivas gráficas.

3.1.1 Gerador

Indo ao encontro do que foi desenvolvido na primeira parte do trabalho, o gerador destina-se a gerar os vários pontos constituintes das diferentes primitivas gráficas conforme os parâmetros escolhidos. Nesta fase, de maneira a representar o anel de Saturno, foi acrescentada a primitiva **Torus** e os respetivos requisitos para o seu desenvolvimento.

3.2 Motor

O objetivo desta aplicação continua a ser o mesmo: permitir a apresentação de uma janela e exibição dos modelos requisitados. Além disso, permite, também,

a interação com os mesmos a partir de certos comandos. Tal como na versão anterior, existe um ficheiro **XML** que vai ser interpretado. No entanto, para esta fase do projeto, a arquitetura deste ficheiro evolui de maneira a cumprir os requisitos. Assim, com o aumento da complexidade do projeto, também aumenta a das primitivas, tendo estas que ter certas informações associadas, pedindo medidas de armazenamento diferentes e, consequentemente de renderização.

3.3 Classes

Na fase anterior, o grupo apenas criou as aplicações requisitadas sem ser necessário o armazenamento de informação na forma de classes. No entanto, para esta fase decidiu-se criar uma classe para cada transformação geométrica (translação, rotação e escala), assim como para a cor.

3.3.1 Rotação

Classe que guarda toda a informação referente a uma rotação. Sendo assim, é composta por quatro variáveis **angle**, **x_eixo**, **y_eixo** e **z_eixo**, que identificam em qual dos eixos vai ser efetuada a rotação.

```
1
2
3 #ifndef PROJECT_ROTACAO_H
4 #define PROJECT_ROTACAO_H
5
6
7 class Rotacao {
8     float angle, x_eixo, y_eixo, z_eixo;
9
10 public:
11     Rotacao();
12     Rotacao(float angle, float x, float y, float z);
13     float getAngle(){ return angle;}
14     float getX(){ return x_eixo; }
15     float getY(){ return y_eixo; }
16     float getZ(){ return z_eixo; }
17     void setAngle(float a){ angle = a; }
18     void setX(float x){ x_eixo = x;}
19     void setY(float y){ y_eixo = y;}
20     void setZ(float z){ z_eixo = z;}
21 };
22
23
24 #endif //PROJECT_ROTACAO_H
```

3.3.2 Translação

Classe que guarda toda a informação referente a uma translação. Sendo assim, é composta por três variáveis **x_eixo**, **y_eixo** e **z_eixo**, que identificam em qual dos eixos vai ser efetuada a Translação.

```
1
```

```
2
3 #ifndef PROJECT_TRANSLACAO_H
4 #define PROJECT_TRANSLACAO_H
5
6 class Translacao{
7     float x_eixo, y_eixo, z_eixo;
8
9 public:
10     Translacao();
11     Translacao(float x, float y, float z);
12     float getX(){ return x_eixo; }
13     float getY(){ return y_eixo; }
14     float getZ(){ return z_eixo; }
15     void setX(float x){ x_eixo = x;}
16     void setY(float y){ y_eixo = y;}
17     void setZ(float z){ z_eixo = z;}
18 };
19 #endif //PROJECT_TRANSLACAO_H
```

3.3.3 Escala

Classe que guarda toda a informação referente à execução de um redimensionamento. Sendo assim, é composta por três variáveis **x_eixo**, **y_eixo** e **z_eixo**, que identificam as dimensões(em relação as originais) dos diferentes eixos.

```
1
2
3 #ifndef PROJECT_ESCALA_H
4 #define PROJECT_ESCALA_H
5
6 class Escala{
7     float x_eixo, y_eixo, z_eixo;
8
9 public:
10     Escala();
11     Escala(float x, float y, float z);
12     float getX(){ return x_eixo; }
13     float getY(){ return y_eixo; }
14     float getZ(){ return z_eixo; }
15     void setX(float x){ x_eixo = x;}
16     void setY(float y){ y_eixo = y;}
17     void setZ(float z){ z_eixo = z;}
18 };
19 #endif
```

3.3.4 Cor

Classe que guarda toda a informação referente à alteração da cor. Sendo assim, é composta por três variáveis **red**, **green** e **blue**, que identificam as diferentes cores segundo o **modelo RGB**.

```
1
2
3 #ifndef PROJECT_COR_H
4 #define PROJECT_COR_H
```

```
5
6  class Cor{
7      float red;
8      float green;
9      float blue;
10
11 public:
12     Cor();
13     Cor(float r, float g, float b);
14     float getR(){ return red; }
15     float getG(){ return green; }
16     float getB(){ return blue; }
17     void setR(float r){ red = r; }
18     void setG(float g){ green = g; }
19     void setB(float b){ blue = b; }
20 };
21 #endif //PROJECT_COR_H
```

3.3.5 Transformação

Classe que guarda um objeto do tipo**Translacao**, **Rotacao**, **Escala** e **Cor**.

```
1
2  #include "Translacao.h"
3  #include "Rotacao.h"
4  #include "Escala.h"
5  #include "Cor.h"
6  #ifndef PROJECT_TRANSFORMACAO_H
7  #define PROJECT_TRANSFORMACAO_H
8
9
10 class Transformacao {
11     Translacao trans;
12     Rotacao rotacao;
13     Escala escala;
14     Cor cor;
15
16
17 public:
18     Transformacao();
19     Transformacao(Translacao trans, Rotacao rotacao, Escala escala,
       Cor cor);
20     Translacao getTrans() { return trans; }
21     Rotacao getRotacao() { return rotacao; }
22     Escala getEscala(){ return escala;}
23     Cor getCor(){ return cor; }
24     void setTrans(Translacao t){ trans = t; }
25     void setRotacao(Rotacao r){ rotacao = r; }
26     void setEscala(Escala esc){ escala = esc; }
27     void setCor(Cor c){ cor = c; }
28 };
29
30 #endif
```

3.3.6 Transforms

Classe que guarda toda a informação relativa a um conjunto de transformações totais de um grupo, incluindo os subgrupos (filhos). Sendo assim, constituída por um **tipo**, a transformação que irá ser aplicada **Transformacao t**, um vetor com os filhos **vector**$< Transforms >$ **subgrupo** e os pontos para o desenho da figura **vector**$< Ponto >$ **pontos**.

```
1
2  #include <vector>
3  #include "Transformacao.h"
4  #include "Ponto.h"
5  #include <fstream>
6  #include <iostream>
7  #include <string>
8
9  #ifdef __APPLE__
10 #include <GLUT/glut.h>
11 #else
12 #include <GL/glut.h>
13 #endif
14
15
16
17 using namespace std;
18 #ifndef PROJECT_TRANSFORMS_H
19 #define PROJECT_TRANSFORMS_H
20
21 #endif //PROJECT_TRANSFORMS_H
22
23 class Transforms{
24     string tipo;
25     Transformacao t;
26     vector<Transforms> subgrupo;
27     vector<Ponto> pontos;
28
29 public:
30     Transforms();
31     Transforms(string tipo, Transformacao t, vector<Transforms> sub
      , vector<Ponto> pontos);
32     string getTipo(){ return tipo; }
33     Transformacao getTrans(){ return t; }
34     vector<Transforms> getSubgrupo(){ return subgrupo; }
35     vector<Ponto> getPontos(){ return pontos; }
36     void setTipo(string t){ tipo = t; }
37     void setTrans(Transformacao trans){ t = trans;}
38     void setSubgrupo(vector<Transforms> sub){ subgrupo = sub; }
39     void setPontos(vector<Ponto> p){ pontos = p;}
40 };
```

3.3.7 Ponto

Classe que guarda toda a informação referente aos diferentes pontos necessários para construir um triângulo. Sendo assim, é composta por três variáveis **x**, **y** e **z**, que representam as suas coordenadas.

```
1
2
3  #ifndef PROJECT_PONTO_H
4  #define PROJECT_PONTO_H
5
6
7  class Ponto {
8
9      float x;
10     float y;
11     float z;
12
13 public:
14     Ponto();
15     Ponto(float, float, float);
16     float getX() { return x; }
17     float getY() { return y; }
18     float getZ() { return z; }
19     void setX( float a) {  x = a; }
20     void setY( float a) {  y = a; }
21     void setZ( float a) {  z = a; }
22
23
24 };
25
26
27 #endif //PROJECT_PONTO_H
```

4 Gerador

Como foi dito anteriormente, o gerador é o responsável pela realização de ficheiros que contêm os pontos necessários à triangulação de certas primitivas geométricas requisitadas. Nesta fase, foi acrescentada, às cinco anteriores, a primitiva **Torus**.

4.1 Torus

Esta primitiva apresenta o formato de uma câmara de pneu. Desta maneira, pode-se pensar como o espaço tridimensional formado por uma superfície plana com um raio interno r em torno de uma circunferência de raio R. Assim os parâmetros para a geração de um torus são **raioI**, **raioE**, **slices** e **stacks**.

4.2 Algoritmo

Toda a construção desta geometria baseia-se nos dois raios existentes, i.e, são usadas duas circunferências para o desenho do mesmo, sendo uma para a definição dos pontos internos e, por isso consideramos o raio desta o raio interno, e outra para a definição dos pontos externos e, assim, o raio deste é o raio externo. Para esta execução é, também, necessário definir quais os eixos responsáveis pela definição das duas circunferências. Assim, os eixos **X e Y**

definem a circunferência interior de raio R e os eixos **X, Y e Z** definem a exterior de raio r, como podemos comprovar pelas seguintes figuras.

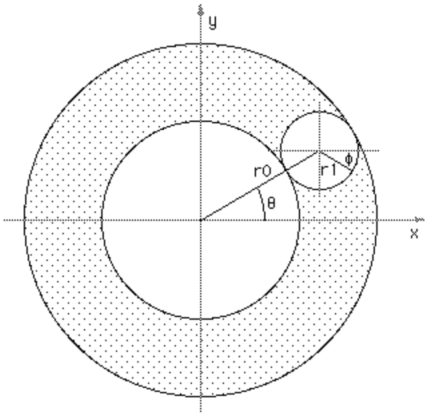

Figura 7: Representação do formato do Torus

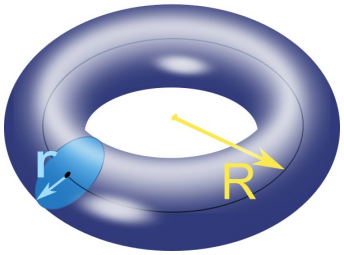

Figura 8: Representação tridimensional do Torus

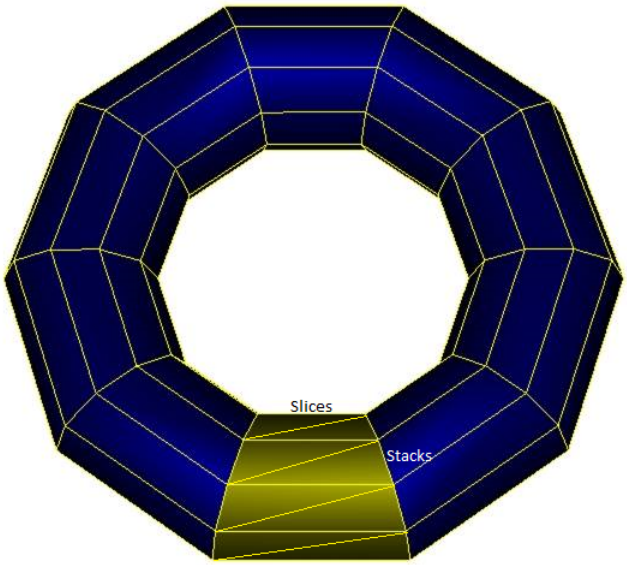

Figura 9: Esboço da construção do Torus

Como podemos observar na *Fig.6* acima, de maneira a percorrer as circunferências definidas usamos os parâmetros **slices** e **stacks**, para dividir a execução em diferentes partes com amplitudes **shiftT** e **shifP**:

```
shifT = 2 * M_PI / slices
shiftP = 2 * M_PI / stacks
```

E com a utilização das funções **sin** e **cos** obtemos os pontos que compõem a triangulação de cada circunferência. Os pontos de referência de amplitude têm como coordenadas **(x1,y1,z1)** e, após o deslocamento, **(x2,y2,z2)**:

```
x1 = cos(theta)*(raioI+raioE*cos(phi));
y1 = sin(theta) * (raioI + raioE * cos(phi));
z1 = raioE * sin(phi);

x2 = cos(theta + shiftT)*(raioI+raioE*cos(phi));
y2 = sin(theta + shiftT) * (raioI + raioE * cos(phi));
z2 = raioE * sin(phi);
```

Sendo estes os pontos delimitadores do anel a ser descrito, i.e, uma slice. Para a construção de uma stack, os pontos delimitadores são **(x3,y3,z3)** e **(x4,y4,z4)**:

```
x3 = cos(theta + shiftT)*(raioI+raioE*cos(phi + shiftP));
y3 = sin(theta + shiftT) * (raioI + raioE * cos(phi + shiftP));
z3 = raioE * sin(phi + shiftP);
```

```
x4 = cos(theta)*(raioI+raioE*cos(phi + shiftP));
y4 = sin(theta) * (raioI + raioE * cos(phi + shiftP));
z4 = raioE * sin(phi + shiftP);
```

Assim, conseguimos percorrer tanto a circunferência interna ao adicionar **shiftP** como a externa ao adicionar **shiftT**. As variáveis **theta** e **phi** representam o ângulo interno e o externo, respetivamente. Portanto, foi criado um ciclo para a iteração de cada circunferência onde a variável **i** corresponde a cada **slice** (começa em 0 e termina igual ao número de slices passado como parâmetro da função) e a variável **j** corresponde a cada stack (começa em 0 e termina igual ao número de stacks passadas como parâmetro). No final de cada iteração de construção da stack tem que se incrementar **phi** de maneira a formar um anel. Mas também, no final de cada anel construido tem que se incrementar o valor de **theta** de maneira a ser possível a passagem para a construção do anel imediatamente a seguir ao que foi gerado até completar toda a circunferência externa e formar o **torus**.

5 Motor

O motor tem como uma das suas funções receber os ficheiros de configuração escritos em XML. Na primeira fase do projeto, este tinha como simplesmente que reconhecer e representar o conteúdo que se encontra nos ficheiros XML. Na segunda fase foi preciso fazer algumas alterações de maneira a que seja possível renderizar tanto o conteúdo deste ficheiros como também as respetivas transformações associadas a estes, apresentando-as no fim num cenário ao utilizador.

5.1 Leitura XML

Para esta segunda fase do projeto, foi preciso alterar o parser de maneira a que este possa cumprir os novos requisitos desta segunda fase. Para podermos cumprir esses requisitos, usamos os métodos **FirstChildElement** e o **NextSiblingElement**, sendo que com o primeiro conseguimos obter a primeira tag do nível imediatamente a seguir ao da anterior e com o segundo obtemos a próxima tag do mesmo nível anterior.

Com o auxílio destes métodos, percorremos todas as transformações e para cada uma fazemos a verificação dos seus tipos e guardamos a respetiva informação nas respetivas classes desenvolvidas. Como o filho tem de herdar todas as informações e características do pai o passo seguinte foi fazer com que isso fosse possível. Para alterarmos as rotações e translações somamos os novos valores aos antigos, mas para a escala multiplicamos os novos valores pelos valores já presentes e para a cor apenas é herdada a cor do pai.

Depois de tudo isto, é feito a verificação de quais os modelos que estão a ser transformados e guardamos a respetiva informação na classe *Transforms* de modo a que mais adiante essa transformação possa ser desenhada com sucesso.

Finalmente, é verificamos se ainda falta realizar o *parse* aos seus irmãos e/ou aos filhos, sendo que se for preciso aplicamos novamente o processo acima

explicado.

6 Processo de Renderização

Para o desenho em si, uma vez que serão efetuadas transformações geométricas, implica uma alteração na matriz de transformação e por isso deve ser guardado o estado inicial desta, e depois de todas as transformações serem aplicadas, este estado deve ser reposto. Para isso usamos as funções **glPushMatrix()** e **glPopMatrix()** antes de aplicarmos a transformação e depois desta ser aplicada respetivamente. Para percorrermos as transformações implementamos um ciclo que percorre o vetor **transformacoes** que contém todas as transformações, e por cada iteração, fazemos um **getTrans()** para obtermos uma transformação.

De seguida, utilizando essa transformação, adquirimos os dados respetivos das rotações, translações, escala e cor e utilizamos esses valores como parâmetros nas funções *GLUT* para o precesso de desenho. Para esse desenho usamos as funções **glRotatef()**, **glTranslatef()**, **glScalef()** e **glColor3f()**. Depois de realizadas todas as transformações, percorremos o vetor **pontos** para procedermos ao desenho dos triângulos usando a função **glBegin(GL_TRIANGLES)**.

7 Análise de Resultados

O resultado final correspondeu ao idealizado pelo grupo em termos gráficos, pois tentou-se que todos os planetas correspondessem à escala visual da realidade, apesar de não corresponder à escala real. Ao mesmo tempo, foram aplicadas diferentes cores nos planetas e incluídos alguns satélites naturais dos mesmos.

7.1 Visualização

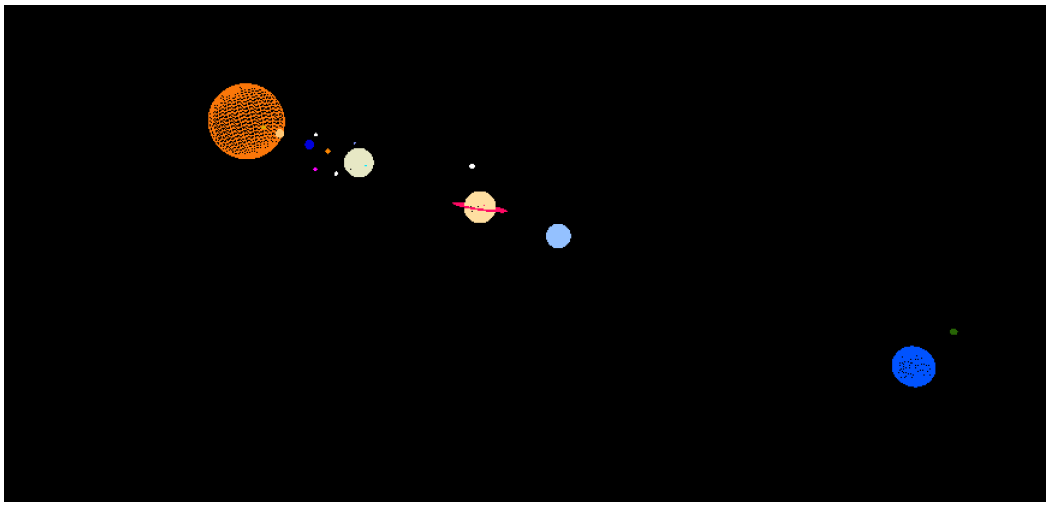

Figura 10: Sistema Solar completo

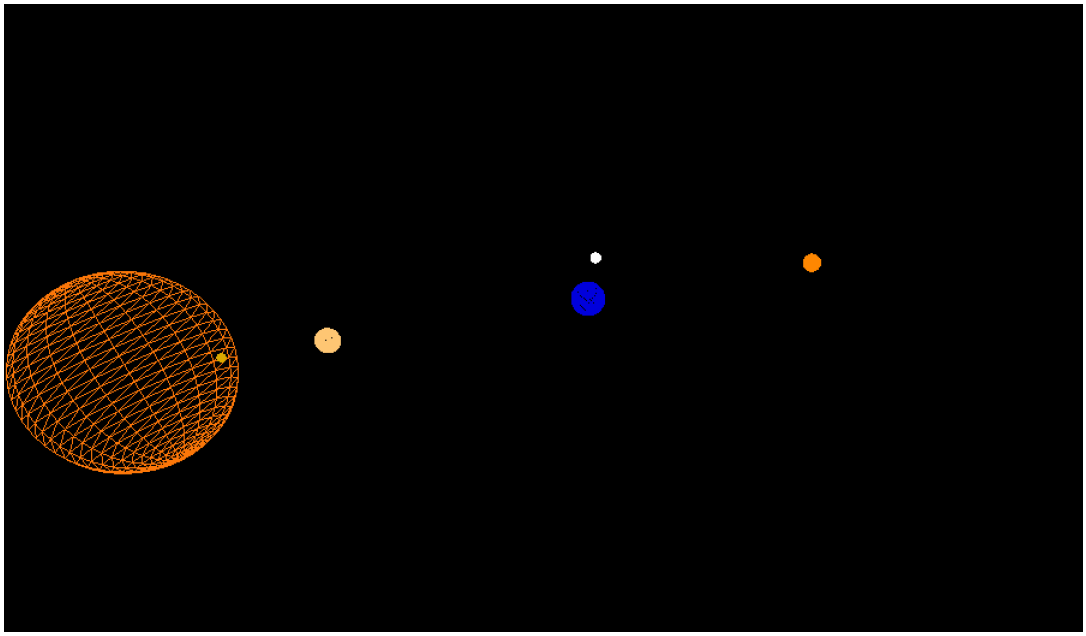

Figura 11: Quatro primeiros planetas e a Lua

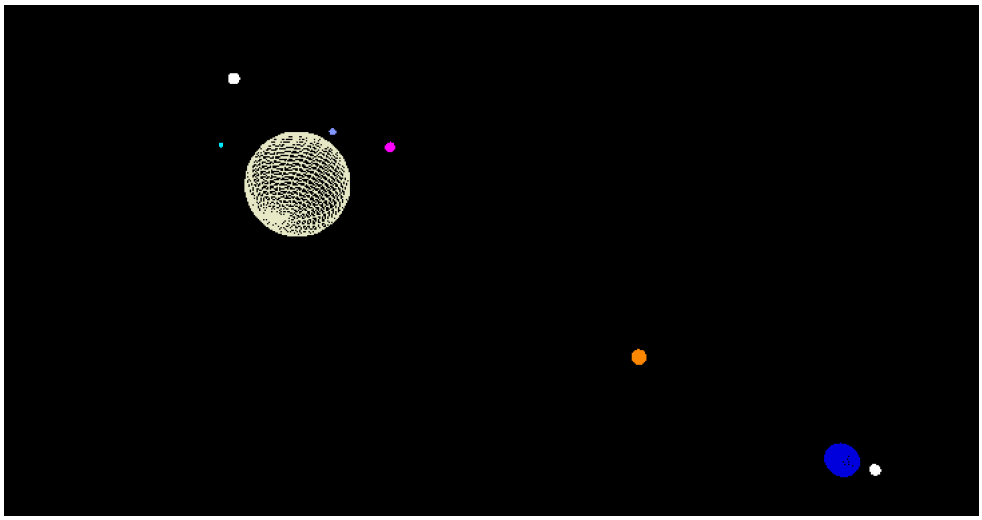

Figura 12: Júpiter e os satélites IO, Europa, Calisto e Ganímedes

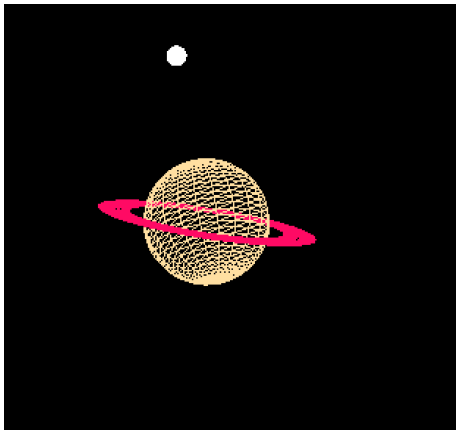

Figura 13: Saturno e o satélite Titã

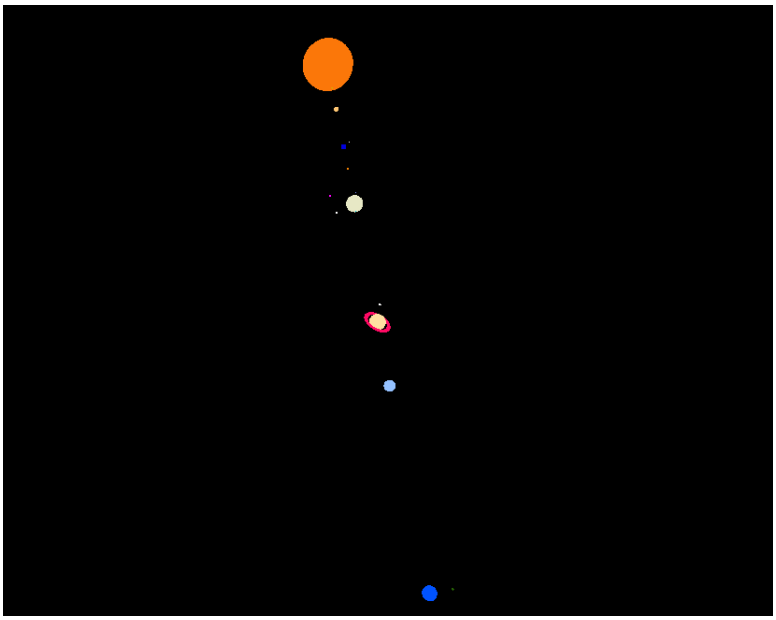

Figura 14: Sistema Solar preenchido

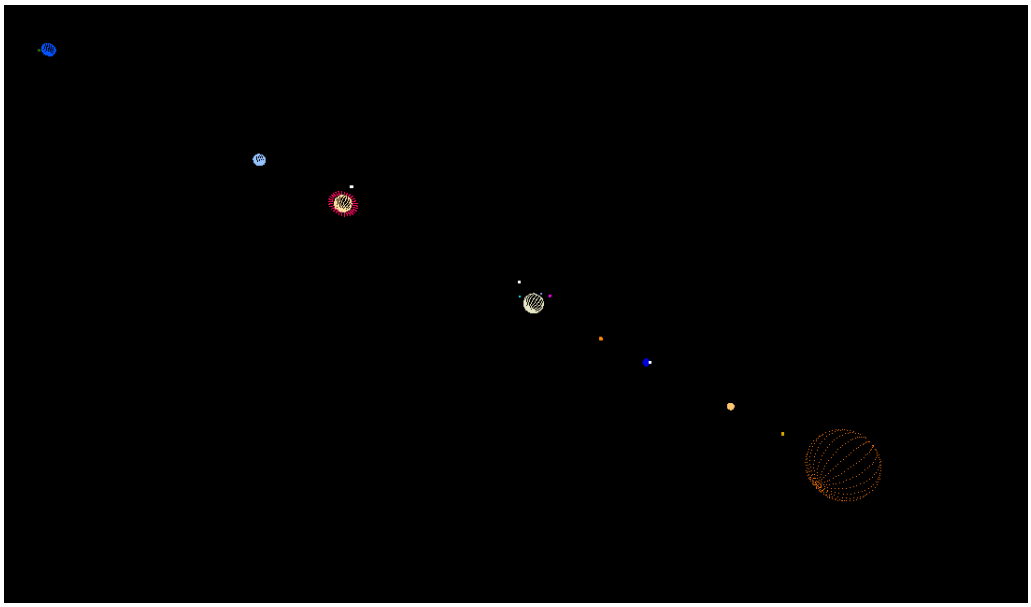

Figura 15: Sistema Solar em pontos

8 Conclusão

Nesta segunda fase do projeto foi-nos possível aprofundar os conhecimentos obtidos nas aulas, mas ao mesmo tempo, consideramos que esta fase foi relativamente menos demorada e trabalhosa em relação à anterior devido ao facto de já termos certos aspetos da matéria consolidada. No geral, consideramos que a maior parte dos resultados obtidos nesta fase correspondem aos esperados, pois o Sistema Solar corresponde ao espectado. No entanto houve um aspeto que tentámos desenvolver, a câmara FPS, mas os resultados obtidos não correspondiam na totalidade ao desejado. Em conclusão, esperamos que nas restantes fases ultrapassemos estes obstáculos e consigamos melhorar cada vez mais o modelo em questão, tornando-o mais realista e mais agradável visualmente.

9 Anexos

Segue em anexo o ficheiro **xml** de configuração do Sistema Solar

```
<scene>
    <!--sol-->
    <group>
        <group>
            <colour R="251" G="119" B="9" />
            <scale X="10" Y="10" Z="10"/>
            <models>
                <model fich = "esfera.3d"/>
            </models>
        </group>

        <!--mercurio-->
        <group>
            <translate X="10.8775" Y="0" Z="0" />
            <colour R="219" G="170" B="0" />
            <scale X="0.3525" Y="0.3525" Z="0.3525" />
            <models>
                <model fich = "esfera.3d"/>
            </models>
        </group>
         <group>
            <!--venus-->
             <translate X="19.9233" Y="0" Z="9.0897" />
             <colour R="254" G="198" B="115" />
             <scale X="0.8790" Y="0.8790" Z="0.8790" />
             <models>
                 <model fich = "esfera.3d"/>
             </models>
         </group>
         <group>
             <!--terra -->
             <translate X="33.8061" Y="0" Z="21.25" />
             <colour R="0" G="0" B="220" />
             <scale X="0.8865" Y="0.8865" Z="0.8865" />
             <models>
                 <model fich="esfera.3d" />
             </models>
             <group>
              <!--    lua-->
                 <rotate angle="2" X="0" Y="1" Z="1"/>
                 <translate X="0" Y="0.25" Z="3" />
                 <colour R="255" G="255" B="255" />
                 <scale X="0.30" Y="0.30" Z="0.30" />
                 <models>
                     <model fich = "esfera.3d"/>
                 </models>
             </group>
         </group>
         <group>
             <!-- marte -->
                  <translate X="40.9523" Y="0" Z="15.5806" />
                  <colour R="254" G="135" B="1" />
                  <scale X="0.3720" Y="0.3720" Z="0.3720" />
                  <models>
```

```
55                      <model fich = "esfera.3d"/>
56                  </models>
57              </group>
58          <group>
59              <!-- jupiter -->
60                      <translate X="51.12987" Y="0" Z="37.1480" />
61                      <colour R="231" G="232" B="197" />
62                      <scale X="2.5" Y="2.5" Z="2.5" />
63                      <models>
64                          <model fich = "esfera.3d" />
65                      </models>
66              <!--satelite IO-->
67              <group>
68                  <rotate angle="2" X="0" Y="0" Z="1"/>
69                  <translate X="0" Y="0" Z="3.375" />
70                  <colour R="130" G="150" B="255" />
71                  <scale X="0.052" Y="0.052" Z="0.052" />
72                  <models>
73                      <model fich = "esfera.3d" />
74                  </models>
75              </group>
76              <!--satelite Europa -->
77              <group>
78                  <translate X="2.025" Y="0" Z="3.5074" />
79                  <colour R="0" G="235" B="255" />
80                  <scale X="0.0353" Y="0.0353" Z="0.0353" />
81                  <models>
82                      <model fich = "esfera.3d" />
83                  </models>
84              </group>
85              <!--    satelite Ganimedes-->
86              <group>
87                  <rotate angle="4" X="0" Y="-1" Z="0"/>
88                  <translate X="-2.025" Y="0" Z="2.4646" />
89                  <colour R="255" G="0" B="255" />
90                  <scale X="0.1" Y="0.1" Z="0.1" />
91                  <models>
92                      <model fich = "esfera.3d" />
93                  </models>
94              </group>
95              <!-- satelite Calisto-->
96               <group>
97                   <rotate angle="3" X="0" Y="-1" Z="0"/>
98                   <translate X="2.525" Y="0" Z="0" />
99                   <colour R="255" G="255" B="255" />
100                  <scale X="0.11" Y="0.11" Z="0.11" />
101                  <models>
102                      <model fich = "esfera.3d" />
103                  </models>
104              </group>
105             </group>
106                 <group>
107                     <!--saturno -->
108                     <translate X="77.5005" Y="0" Z="-21.3021" />
109                     <colour R="255" G="223" B="161" />
110                     <scale X="1.995" Y="1.995" Z="1.995" />
111                         <models>
```

```
                            <model fich = "esfera.3d"/>
                        </models>
                    <!-- satelite Tit  -->
                     <group>
                         <rotate angle="2" X="0" Y="0" Z="1"/>
                         <translate X="0" Y="1.2" Z="3.375" />
                         <colour R="255" G="255" B="255" />
                         <scale X="0.15" Y="0.15" Z="0.15" />
                         <models>
                             <model fich = "esfera.3d" />
                         </models>
                     </group>
                    <!--anel-->
                    <group>
                        <rotate angle="90" X="1" Y="0" Z="0"/>
                        <colour R="255" G="255" B="255" />
                        <scale X="0.5" Y="0.5" Z="0.0"/>
                        <models>
                            <model fich="torus.3d" />
                        </models>
                    </group>
                </group>
        <group>
          <!-- urano -->
            <translate X="88.0747"  Y="0" Z="50.85" />
            <colour R="148" G="193" B="255" />
            <scale X="1.290" Y="1.290" Z="1.290" />
            <models>
                <model fich = "esfera.3d"/>
            </models>
        </group>
        <group>
            <!--neptuno-->
             <translate X="112.35529" Y="0" Z="46.53908" />
             <colour R="0" G="83" B="255" />
             <scale X="1.275" Y="1.275" Z="1.275" />
             <models>
                 <model fich = "esfera.3d"/>
             </models>
            <!-- satelite Tit nia-->
          <group>
              <rotate angle="1" X="0" Y="3" Z="1"/>
              <translate X="0" Y="1" Z="3.75" />
              <colour R="39" G="100" B="5" />
              <scale X="0.15" Y="0.15" Z="0.15" />
              <models>
                  <model fich = "esfera.3d" />
              </models>
          </group>
      </group>
    </group>
</scene>
```