
Computação Gráfica 2ª Parte

TRABALHO REALIZADO POR:

JOÃO FIGUEIREDO MARTINS PEIXE DOS SANTOS

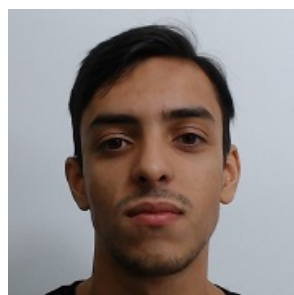
FRANCISCO ALVES ANDRADE

LUÍS FILIPE CRUZ SOBRAL

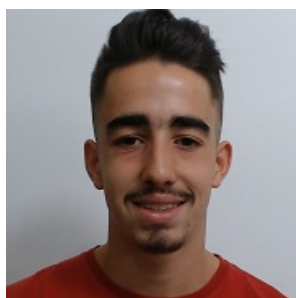
PAULO SILVA SOUSA



A89520 João Santos



A89474 Luís Sobral



A89465 Paulo Sousa



A89513 Francisco Andrade

GRUPO 14
PROJETO CG
2020/2021
UNIVERSIDADE DO MINHO

Índice

1	Introdução	1
2	Arquitetura do Código	2
2.1	Generator	2
2.2	Engine	4
2.2.1	Transformation	4
2.2.2	Shape	5
2.2.3	Group	5
2.2.4	Parser	6
2.2.5	Camera	7
2.2.6	Point	8
3	Demonstração	9
3.1	Generator	9
3.2	Engine	10
3.3	Escala utilizada	11
3.4	Sistema Solar	12
4	Conclusão e Trabalho Futuro	15
A	Anexos	16
A.1	Ficheiro XML - Sistema Solar	16

Índice de Figuras

1	Ciclo responsável por calcular o alpha e o beta	2
2	Esquema de controlo de espessura do Torus	3
3	Função que gera os pontos da figura	3
4	Torus Point	3
5	Torus Line	3
6	Tansformação translação	4
7	Tansformação rotação	4
8	Tansformação escala	4
9	Tansformação colorir	5
10	Método que desenha as figuras	5
11	Variáveis de instância da classe Group	5
12	Função que renderiza os constituintes de um grupo	6
13	Função que faz parse da informação de um grupo	6
14	Definições predefinidas da câmara	7
15	Movimentação da câmara	7
16	Orientação da câmara com o rato	8
17	Comando para gerar o ficheiro <i>sphere.3d</i>	9
18	Comando para gerar o ficheiro <i>orbit.3d</i>	9
19	Comando para gerar o ficheiro <i>asteroidsBelt.3d</i>	9
20	Comando para gerar o ficheiro <i>ring.3d</i>	9
21	Comando para correr o Sistema Solar	10
22	Menu de auxílio à utilização da câmara	10
23	Câmara superior próxima do Sol	12
24	Câmara lateral próxima do Sol	12
25	Câmara superior próxima de Júpiter e Saturno	13
26	Câmara lateral próxima de Júpiter e Saturno	13
27	Câmara superior dos Planetas Gasosos	14
28	Câmara próxima de Urano e Neptuno	14
29	Câmara próxima de Plutão e Cintura de Kuiper	14

1 Introdução

A segunda fase do projeto tem por base a utilização de modelos mais complexos, implementando novas funcionalidades ao desenvolvimento de representações de primitivas geométricas realizado na fase anterior.

Para isso, realizamos alterações na estrutura XML para que consigamos efetuar diversas transformações geométricas, tais como: rotações, translações e alterações da escala do referencial.

Introduzimos, adicionalmente, o *Thorus*, com o objetivo de nos auxiliar na representação do Sistema Solar que era pretendido, que conta com os elementos mais significativos do mesmo.

2 Arquitetura do Código

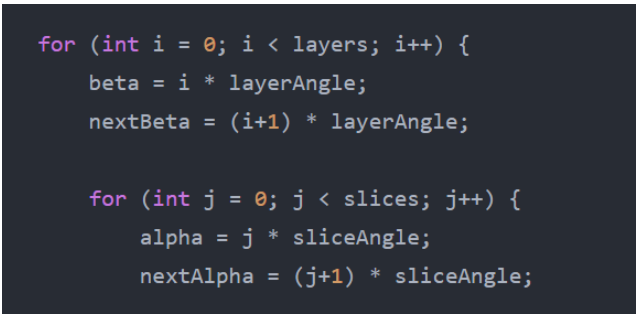
Tal como na primeira fase, continuamos a utilizar duas aplicações: *Generator* e *Engine*. Devido à nova estrutura XML, o trabalho foi sujeito não só a alterações em classes previamente criadas como também à criação de novas classes. Será nestas que nos vamos basear ao descrever as modificações e adições necessárias para suportar a nova disposição do sistema e as suas respetivas funcionalidades.

2.1 Generator

O *Generator* continua a estar encarregue da função de gerador dos pontos dos triângulos constituintes das diversas figuras geométricas construídas. Para além de tudo aquilo que foi desenvolvido na fase anterior, resolvemos implementar uma nova figura: o Torus. Esta implementação levou à criação de uma nova classe *torus.cpp*, com a respetiva primitiva:

- *torus(float radiusIn, float radiusOut, int slices, int layers);*

Para a implementação desta figura foi utilizada a forma de preencher uma camada sucessivamente até completar o Torus. São fornecidos dois raios (interno e externo) que são responsáveis por controlar o tamanho e a espessura da figura. Dividimos a figura em layers (camadas horizontais que cortam cada cilindro) e em slices (que formam uma fatia que parte do centro da figura e estende-se até à parte mais exterior). A nossa implementação passa por gerar todas as slices de cada layer, passando de seguida para a próxima layer, até completarmos o desenho da figura delimitada pelos anéis interior e exterior.



```
for (int i = 0; i < layers; i++) {  
    beta = i * layerAngle;  
    nextBeta = (i+1) * layerAngle;  
  
    for (int j = 0; j < slices; j++) {  
        alpha = j * sliceAngle;  
        nextAlpha = (j+1) * sliceAngle;  
    }  
}
```

Figure 1: Ciclo responsável por calcular o alpha e o beta

A imagem abaixo demonstra a forma como controlamos a espessura através dos anéis. Temos ainda em consideração dois ângulos que denominamos como *alpha* e *beta* que são utilizados na construção descrita em cima: o *alpha* nas slices e o *beta* nas layers.

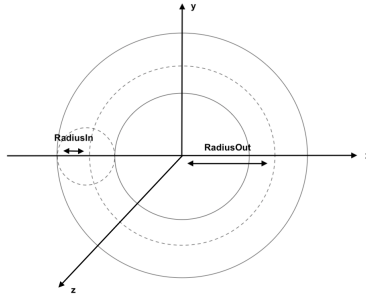


Figure 2: Esquema de controlo de espessura do Torus

Para gerar os pontos da figura criamos uma função *pointCoord*, na qual utilizamos expressões matemáticas para gerar cada ponto, em função dos ângulos *alfa* e *beta*.

```
Point* pointCoord(float radiusIn, float radiusOut, float alpha, float beta) {
    return new Point((radiusOut + radiusIn * (cos(beta))) * cos(alpha),
                    (radiusOut + radiusIn * (cos(beta))) * sin(alpha),
                    radiusIn * sin(beta));
}
```

Figure 3: Função que gera os pontos da figura

$$x = (radiusOut + radiusIn * (\cos(\alpha))) * \cos(\alpha)$$

$$y = (radiusOut + radiusIn * (\cos(\beta))) * \sin(\alpha)$$

$$z = radiusIn * \sin(\beta)$$

Estas são as expressões que permitem a elaboração do exterior do Torus. A expressão do *z* não tem intervenção da variável *radiusOut*, visto que o Torus é criado no plano xOy.

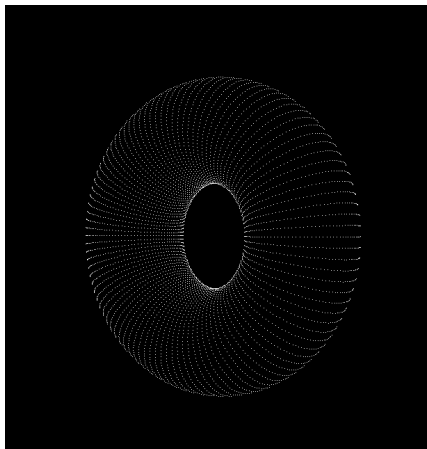


Figure 4: Torus Point

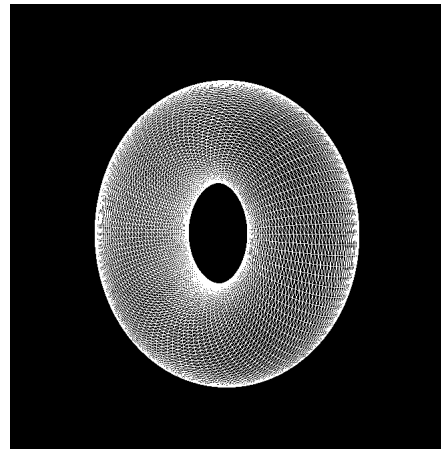


Figure 5: Torus Line

2.2 Engine

Em relação à última fase, o *Engine* sofreu algumas alterações para conseguirmos atingir o objetivo desta etapa e implementarmos as funcionalidades pretendidas.

2.2.1 Transformation

De modo a guardar a informação sobre as transformações aplicadas às figuras criamos a classe *Transformation*. Esta classe tem como variáveis de instância três valores *float* (x, y e z), que são comuns a todas as transformações. Declaramos também um método denominado *transform* que aplica a respetiva transformação.

Como todas as transformações operam de forma distinta, criamos uma subclasse para cada uma delas:

Translation

Esta subclasse não adiciona mais nenhuma variável de instância e implementa o método *transform* da seguinte forma:

```
void Translation::transform() {  
    glTranslatef( x: getX(), y: getY(), z: getZ());  
}
```

Figure 6: Transformação translação

Rotation

Esta subclasse adiciona a variável de instância *angle* que corresponde ao ângulo de rotação e implementa o método *transform* da seguinte forma:

```
void Rotation::transform(){  
    glRotatef( angle: getAngle(), x: getX(), y: getY(), z: getZ());  
}
```

Figure 7: Transformação rotação

Scale

Esta subclasse não adiciona mais nenhuma variável de instância e implementa o método *transform* da seguinte forma:

```
void Scale::transform() {  
    glScalef( x: getX(), y: getY(), z: getZ());  
}
```

Figure 8: Transformação escala

Colour

Decidimos também implementar a transformação *colour* de modo a podermos colorir as figuras implementadas no nosso sistema.

Sendo que as cores *RGB* variam no intervalo $[0; 255]$ e o Glut apenas aceita os valores entre 0 e 1, utilizamos um método de conversão que consiste na divisão do valor pretendido para cada parâmetro por 255, i.e $(R/255, G/255, B/255)$.

```
void Colour::transform() {  
    glColor3f( red: getX()/255, green: getY()/255, blue: getZ()/255);  
}
```

Figure 9: Tansformação colorir

2.2.2 Shape

De modo a organizar o código da melhor maneira e a utilizar as funcionalidades do paradigma da programação orientada aos objetos, desenvolvemos o método responsável pelo desenho das figuras do nosso sistema na classe em questão.

```
void Shape::draw() {  
    for(int i=0; i<points.size(); i+=3) {  
  
        glBegin(GL_TRIANGLES);  
        glVertex3f( x: points[i]->getX(), y: points[i]->getY(), z: points[i]->getZ());  
        glVertex3f( x: points[i+1]->getX(), y: points[i+1]->getY(), z: points[i+1]->getZ());  
        glVertex3f( x: points[i+2]->getX(), y: points[i+2]->getY(), z: points[i+2]->getZ());  
        glEnd();  
    }  
}
```

Figure 10: Método que desenha as figuras

2.2.3 Group

Esta classe é responsável por armazenar informação sobre as nossas figuras, as suas transformações e possíveis subgrupos que esta possa apresentar. Para isso, temos como variáveis de instância três vetores: de tranformações, de figuras e de subgrupos.

```
private:  
    vector<Transformation*> transf;  
    vector<Shape*> models;  
    vector<Group*> groups;
```

Figure 11: Variáveis de instância da classe Group

Desenvolvemos, ainda, o método *Render* responsável por renderizar os constituintes do grupo. Caso este último apresente subgrupos, este método será chamado recursivamente.

```
void Group::render() {  
    for(int i=0; i<transf.size(); i++)  
        transf[i]->transform();  
  
    for(int i=0; i<models.size(); i++)  
        models[i]->draw();  
  
    for(int i=0; i<groups.size(); i++) {  
        glPushMatrix();  
        groups[i]->render();  
        glPopMatrix();  
    }  
}
```

Figure 12: Função que renderiza os constituintes de um grupo

2.2.4 Parser

Nesta fase, este ficheiro foi alvo do maior número de alterações, no sentido em que foi necessária uma adaptação a um novo tipo de ficheiro XML, com vista a permitir a construção das figuras geométricas e a representação das suas respetivas transformações.

O principal método deste ficheiro, *parseXML*, abre o ficheiro a partir do caminho fornecido como argumento na execução do programa e itera por todos os grupos principais, chamando, para cada um deles, a função *parseGroup*.

A função *parseGroup* itera por todos os elementos do grupo e verifica a sua tag XML. Consoante esta última, vai chamar a função de parsing respetiva e adicionar o resultado ao vetor referente.

```
Group *parseGroup(XMLElement *element) {  
    vector<Transformation *> transf;  
    vector<Group *> groups;  
    vector<Shape *> models;  
  
    for (XMLElement *elem = element->FirstChildElement(); elem; elem = elem->NextSiblingElement())  
    {  
        if (strcmp(elem->Name(), "translate") == 0)  
            transf.push_back(parseTranslate(elem));  
        if (strcmp(elem->Name(), "rotate") == 0)  
            transf.push_back(parseRotate(elem));  
        if (strcmp(elem->Name(), "scale") == 0)  
            transf.push_back(parseScale(elem));  
        if (strcmp(elem->Name(), "colour") == 0)  
            transf.push_back(parseColour(elem));  
        if (strcmp(elem->Name(), "models") == 0)  
            models = parseModel(elem);  
        if (strcmp(elem->Name(), "group") == 0)  
            groups.push_back(parseGroup(elem));  
    }  
  
    return new Group(transf, models, groups);  
}
```

Figure 13: Função que faz parse da informação de um grupo

2.2.5 Camera

No que diz respeito à câmara, implementamos um sistema que nos permite a movimentação pelo Sistema Solar através da utilização do teclado. Para isso temos um menu no *Engine* com todas as opções possíveis de deslocação, desde a simples translação até a alteração da velocidade da câmara.

A câmara tem predefinida uma localização inicial com coordenadas atribuídas, e tem também dois ângulos (alpha e beta) que definem a posição da câmara no plano xOz e xOy repetitivamente.

```
void Camera::defaultCam()
{
    startX = 400.0f;
    startY = 400.0f;
    alpha = -90.0f;
    beta = 0.0f;
    first_mouse = true;
    cameraPosition = new Point( x1: 0.0f, y1: 0.0f, z1: 250.0f);
    cameraFront = new Point( x1: 0.0f, y1: 0.0f, z1: -1.0f);
    cameraUp = new Point( x1: 0.0f, y1: 1.0f, z1: 0.0f);
    speedMultiplier = 0.01f;
    mouseSensitivity = 1.0f;
}
```

Figure 14: Definições predefinidas da câmara

Toda a movimentação é executada através da utilização de operações vetoriais (soma, subtração e produto) entre a posição original e o tipo de deslocação pretendida.

```
case 'w':
    v = multiplyVectorBySpeed(cameraFront);
    addVectors(cameraPosition, v);
    break;
case 'a':
    v = multiplyVectorBySpeed( p: normalizeVector( p: crossVectors(cameraFront, cameraUp)));
    subVectors(cameraPosition, v);
    break;
case 's':
    v = multiplyVectorBySpeed(cameraFront);
    subVectors(cameraPosition, v);
    break;
case 'd':
    v = multiplyVectorBySpeed( p: normalizeVector( p: crossVectors(cameraFront, cameraUp)));
    addVectors(cameraPosition, v);
    break;
```

Figure 15: Movimentação da câmara

Para além deste tipo de movimentos, definimos uma forma de visualização livre do sistema solar que acontece numa qualquer coordenada do sistema e é utilizada com recurso ao rato. Isto permite a observação de diversas parcelas do sistema, estando fixo na mesma posição. Tal como a técnica falada anteriormente, o sistema regista a posição do rato inicial e depois adapta-se à sua movimentação automaticamente, produzindo o devido resultado no ecrã.

```
void Camera::processMouseMotion(int xx, int yy)
{
    if (first_mouse)
    {
        startX = xx;
        startY = yy;
        first_mouse = false;
    }

    float deltaX = (xx - startX) * mouseSensitivity;
    float deltaY = (startY - yy) * mouseSensitivity;
    startX = xx;
    startY = yy;

    alpha += deltaX;
    beta += deltaY;

    if (beta > 89.0f)
        beta = 89.0f;
    else if (beta < -89.0f)
        beta = -89.0f;

    cameraFront->update( x1: cos(alpha * 3.14 / 180.0) * cos(beta * 3.14 / 180.0),
                        y1: sin(beta * 3.14 / 180.0),
                        z1: sin(alpha * 3.14 / 180.0) * cos(beta * 3.14 / 180.0));
    cameraFront->normalize();
}
```

Figure 16: Orientação da câmara com o rato

2.2.6 Point

Tendo em conta a utilização da classe *Point* para a representação dos vetores da câmara, foi necessária a adição de funcionalidades à classe respetiva:

- **Update:** atualiza os valores da instância;
- **Add:** soma dois pontos;
- **Sub:** subtrai dois pontos;
- **Multiply:** multiplica um ponto por um escalar;
- **Clone:** duplica uma instância;
- **Cross:** produto vetorial de dois vetores;
- **Normalize:** normalização de um vetor.

3 Demonstração

3.1 Generator

Para a executar a demonstração do Sistema Solar é necessário gerar 4 figuras: *sphere.3d*, *orbit.3d*, *asteroidsBelt.3d* e *ring.3d*. Nas figuras em baixo indicamos como gerar essas figuras.

```
~/G/U/3/2/CG/CG_Project/cmake-build-debug main !1 > ./generator sphere 1 100 100 sphere.3d
File: sphere.3d
```

Figure 17: Comando para gerar o ficheiro *sphere.3d*

```
~/G/U/3/2/CG/CG_Project/cmake-build-debug main !1 > ./generator torus 0.0005 1 100 100 orbit.3d
File: orbit.3d
```

Figure 18: Comando para gerar o ficheiro *orbit.3d*

```
~/G/U/3/2/CG/CG_Project/cmake-build-debug main > ./generator torus 0.01 1 100 100 asteroidsBelt.3d
File: asteroidsBelt.3d
```

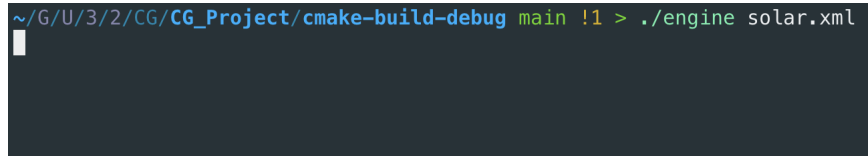
Figure 19: Comando para gerar o ficheiro *asteroidsBelt.3d*

```
~/G/U/3/2/CG/CG_Project/cmake-build-debug main > ./generator torus 0.1 1 100 100 ring.3d
File: ring.3d
```

Figure 20: Comando para gerar o ficheiro *ring.3d*

3.2 Engine

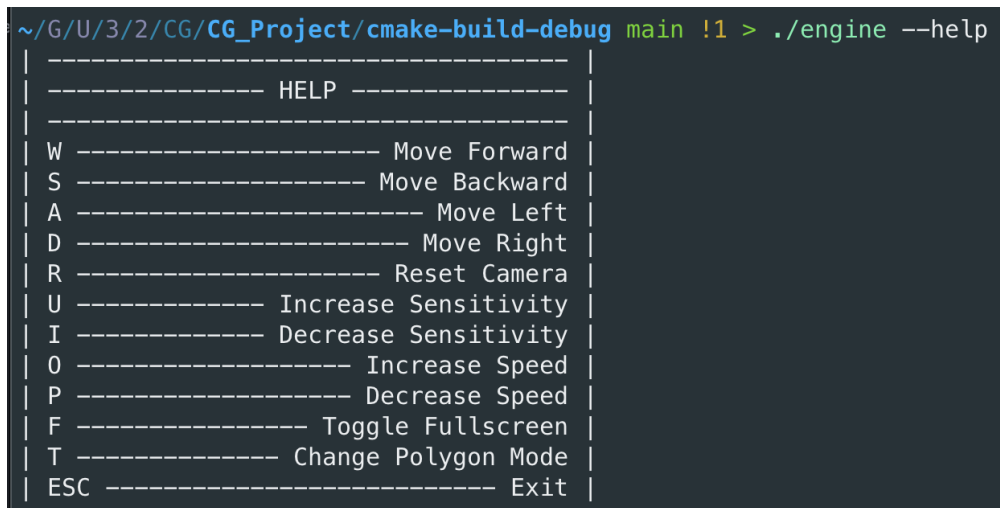
Como é possível ver na figura abaixo, para representar o sistema solar temos de passar como parâmetro o ficheiro *solar.xml* ao correr o engine. Assim, o sistema irá renderizar o sistema de acordo com os ficheiros *.3d* gerados anteriormente.



```
~/G/U/3/2/CG/CG_Project/cmake-build-debug main !1 > ./engine solar.xml
```

Figure 21: Comando para correr o Sistema Solar

Para aceder ao menu de ajuda do Engine, é necessário correr este com a flag *-help* ou com a flag *-h*. Neste menu estão representadas todas as ações que podem ser efetuadas no nosso projeto.



```
~/G/U/3/2/CG/CG_Project/cmake-build-debug main !1 > ./engine --help
| ----- |
| ----- HELP ----- |
| ----- |
| W ----- Move Forward |
| S ----- Move Backward |
| A ----- Move Left |
| D ----- Move Right |
| R ----- Reset Camera |
| U ----- Increase Sensitivity |
| I ----- Decrease Sensitivity |
| O ----- Increase Speed |
| P ----- Decrease Speed |
| F ----- Toggle Fullscreen |
| T ----- Change Polygon Mode |
| ESC ----- Exit |
```

Figure 22: Menu de auxílio à utilização da câmara

3.3 Escala utilizada

Com o intuito de estabelecer uma escala do nosso sistema que fosse fiável e, ao mesmo tempo, que fosse plausível de ser representada permitindo uma observação clara de todos os seus componentes, decidimos utilizar a formula fornecida pela nasa para calcular o tamanho dos planetas e a distância entre eles.

Fórmula para calcular a escala dos planetas (Entre Sol e Marte):

$$x = \frac{10000}{4495060000} * num$$

Para os restantes planetas decidimos fazer uma aproximação, visto que, com as distâncias originais, seria impossível de simular devido ao enorme tamanho do sistema solar, dificultando a representação exata da distância entre alguns dos planetas, por exemplo.

Formula para calcular o diâmetro dos planetas:

$$x = \frac{20}{71492} * num$$

Devido ao enorme tamanho do Sol comparativamente ao resto dos componentes do sistema solar, decidimos reduzir o seu diâmetro para 60 unidades.

Planeta/Estrela		Diametro (km)	Distancia (km)	Diametro (unidades)	Distancia (unidades)
Sol		1392694	-	60	-
Mercúrio		4879	57909000	1.365	128.828
Vénus		12103	108160000	3.386	240.62
Terra		12756	149600000	3.569	332.81
Satélites	Lua	1738	384400	0.486	4.423
Marte		6792	227990000	1.9	507.201
Satélites	Fobos	11	6000	0.003	1.913
	Deimos	6.2	23460	0.002	1.952
Cintura de Asteroides		-	413832947	-	800
Jupiter		142984	778330000	40.0	1100
Satélites	IO	3642	422000	1.019	40.939
	Europa	1560	670900	0.436	41.493
	Ganímedes	5268	1070400	1.474	42.381
	Calisto	4806	1882700	1.344	44.188
Saturno		120536	1429400000	33.72	1400
Satélites	Titã	5150	1221870	1.441	36.438
	Reia	1527	527108	0.427	34.893
	Jápeto	1470	3560820	0.411	41.692
	Dione	1123	377396	0.314	34.56
Urano		51108	2870990000	14.298	1700
Satélites	Titânia	1576	435910	0.441	15.268
	Oberon	1522	583520	0.426	15.596
Neptuno		49538	4504300000	13.858	2000
Satélites	Tritão	2705	354759	0.757	14.647
Plutão		1188	5906380000	0.332	2200
Cintura de Kupier		-	5983914828	-	2500

Table 1: Escalas para demonstração do Sistema Solar

3.4 Sistema Solar

Na constituição do Sistema Solar focamo-nos nos seus principais constituintes, isto é, planetas e suas respectivas órbitas, disposição, tamanho e escala. Tivemos também em mente os satélites naturais constituintes de cada um dos planetas e, ainda, decidimos adicionar a cintura de *Kuiper*. Nesta última, optamos por apenas representar a sua porção mais densa, situada depois da órbita de Plutão, com o intuito de não ocultar outros componentes do sistema em questão e, também, por não possuímos um poder de processamento que fosse capaz de suportar uma extensão tão grande como a da cintura.

Para isso elaboramos um ficheiro XML (*solar.xml*), que permite a construção do Sistema Solar tendo em conta todas as condicionantes e adições referidas acima. Atribuímos cores aos diferentes constituintes e o resultado final foi o seguinte:

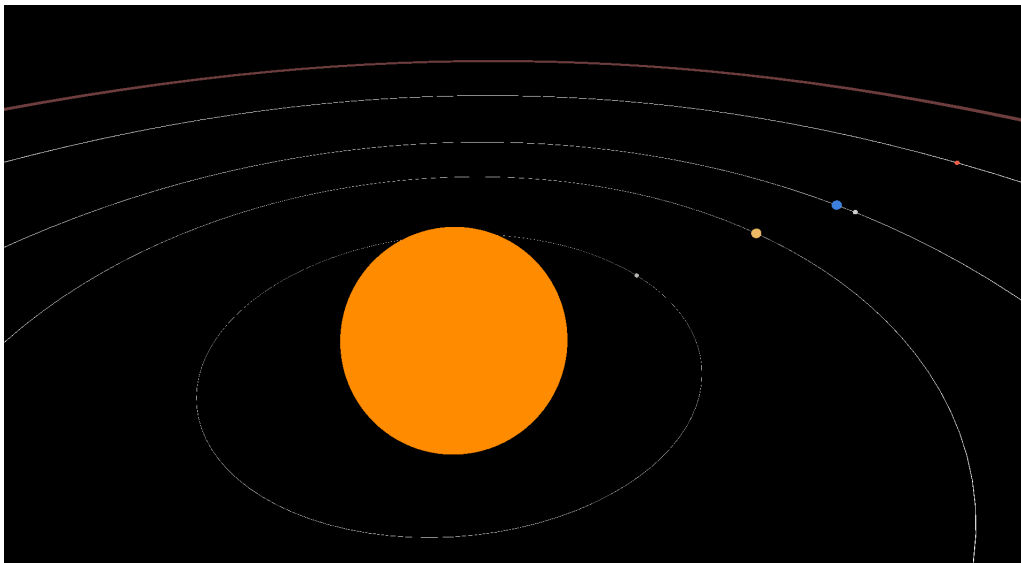


Figure 23: Câmara superior próxima do Sol

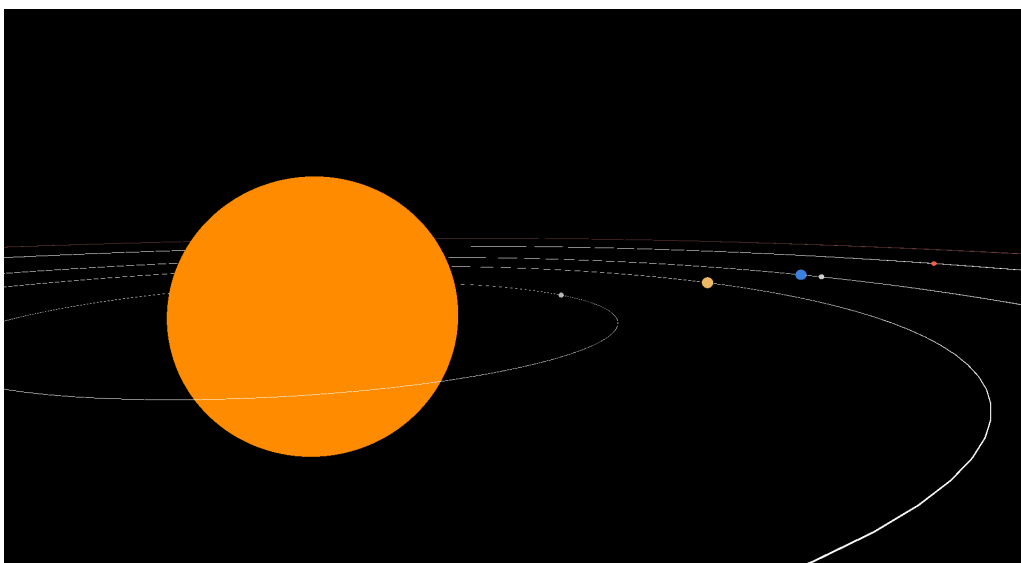


Figure 24: Câmara lateral próxima do Sol

De modo a representar os anéis de Saturno decidimos recorrer ao Torus. Assim, com a utilização de duas destas figuras, e com a utilização de escalas e rotações, conseguimos produzir o efeito pretendido.

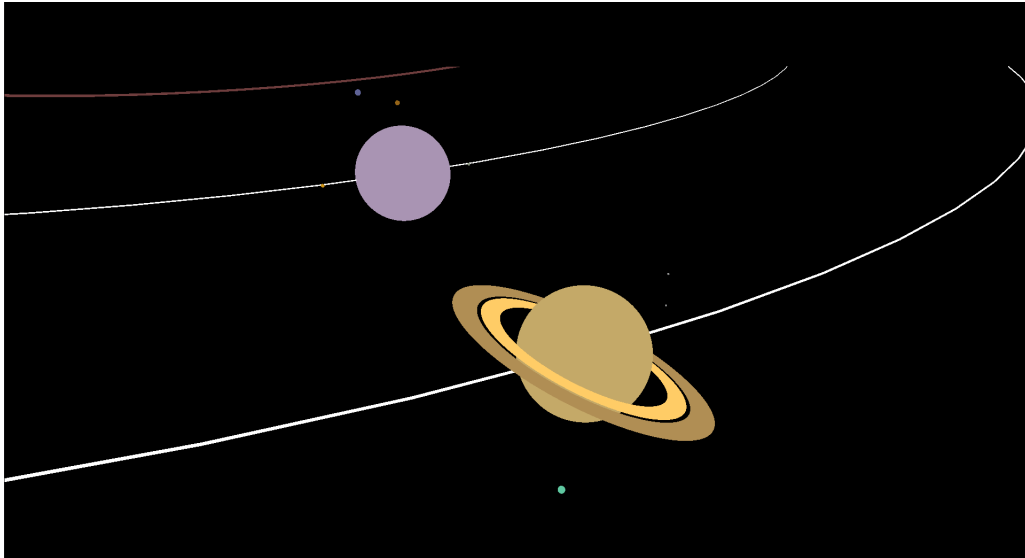


Figure 25: Câmara superior próxima de Júpiter e Saturno

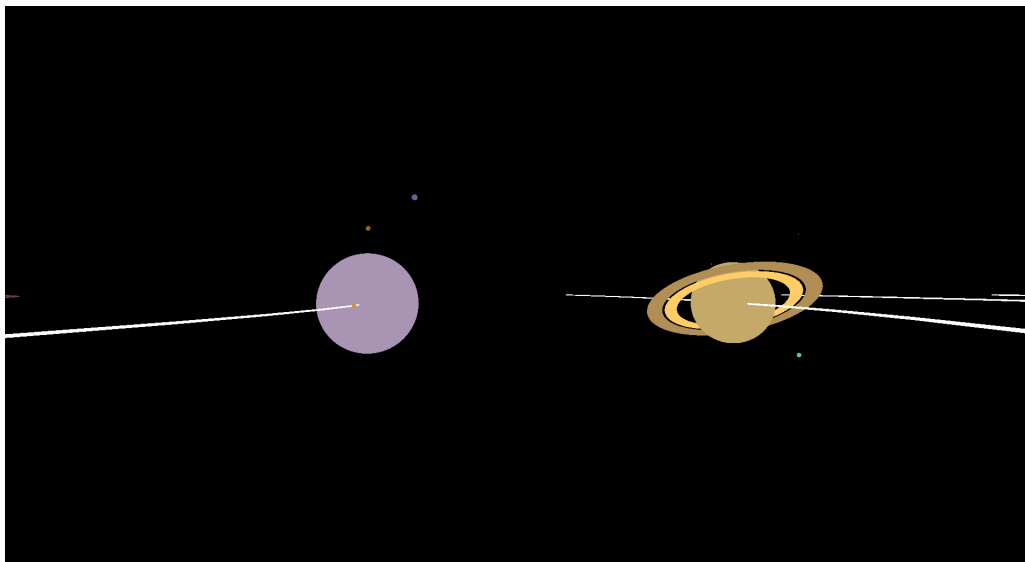


Figure 26: Câmara lateral próxima de Júpiter e Saturno

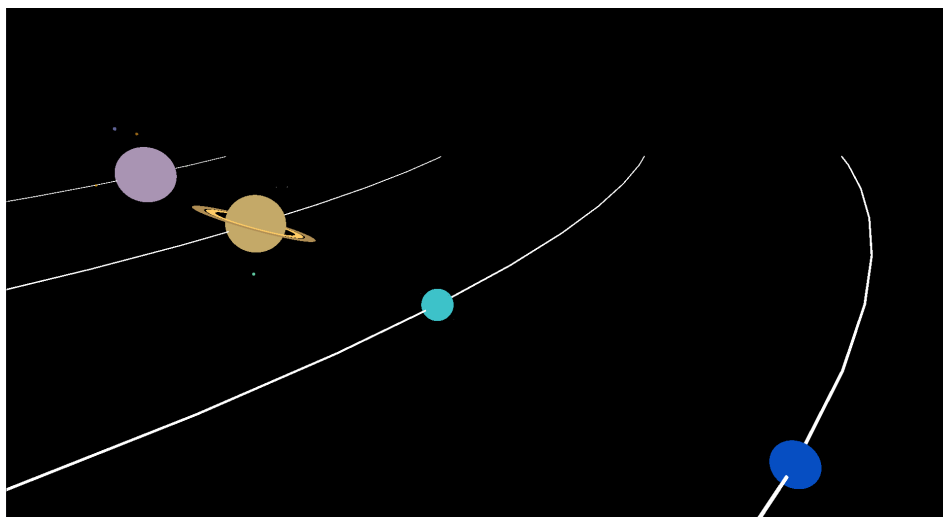


Figure 27: Câmara superior dos Planetas Gasosos

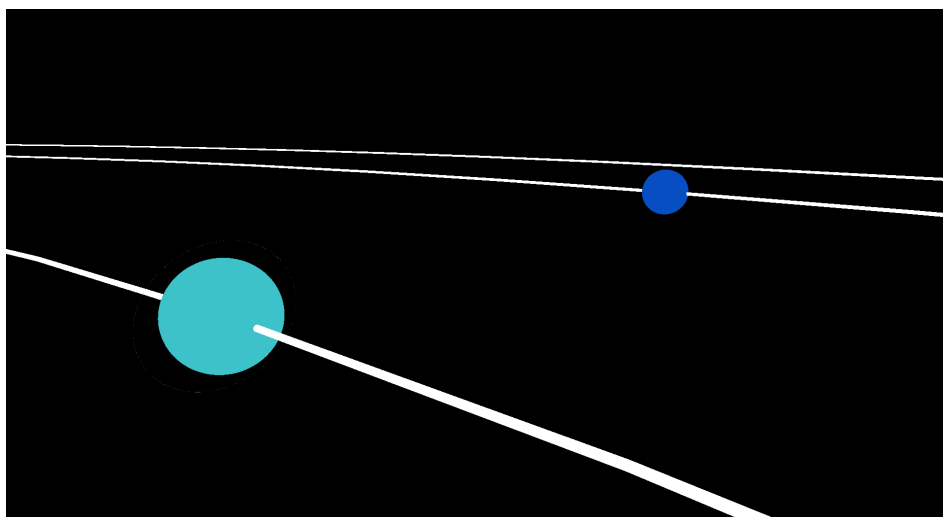


Figure 28: Câmara próxima de Urano e Neptuno

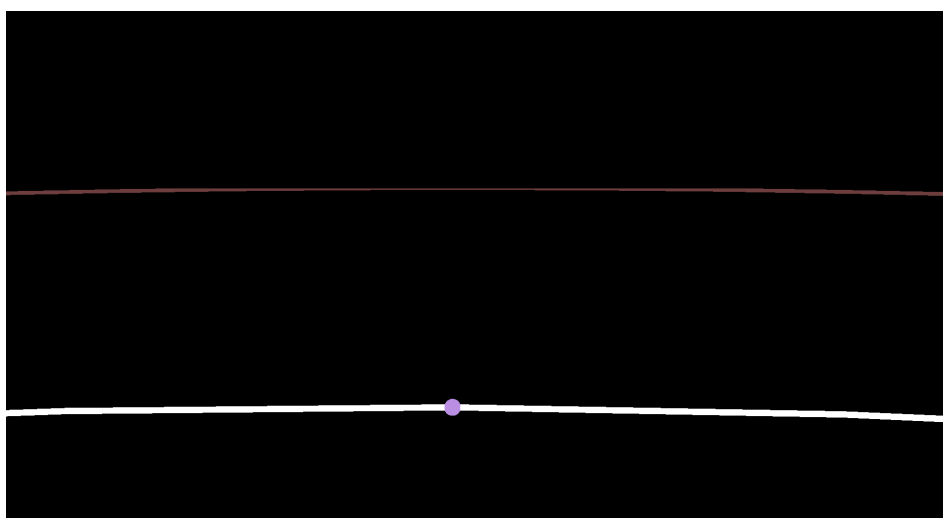


Figure 29: Câmara próxima de Plutão e Cintura de Kuiper

4 Conclusão e Trabalho Futuro

Nesta segunda fase, foi necessária uma reformulação de algum do trabalho feito anteriormente a fim de suportar melhor tudo aquilo que pretendíamos implementar relativamente ao Sistema Solar.

Para a construção do Sistema Solar, foi criado um novo ficheiro XML, tendo, para isso, sido necessárias alterações na realização do processo de *parsing*. Criamos também uma nova figura, o *Torus*, que nos auxiliou bastante no desenvolvimento de algumas das figuras no sistema, como, por exemplo, nos anéis de alguns dos planetas.

No que diz respeito à visualização do sistema, utilizamos o rato para movimentação da câmara e o teclado para nos mobilizarmos ao longo do Sistema Solar, obtendo assim diferentes perspetivas do mesmo.

Assim, consideramos que foi alcançada uma melhor consolidação da matéria lecionada nas aulas, nomeadamente sobre a utilização e domínio de ferramentas e conceitos associados à Computação Gráfica, permitindo, também, a exploração de novos domínios totalmente novos.

A Anexos

A.1 Ficheiro XML - Sistema Solar

```
<scene>
  <!-- Sol -->
  <group>
    <scale X="60" Y="60" Z="60" />
    <colour R="255" G="140" B="0" />
    <models>
      <model file="sphere.3d" />
    </models>
  </group>

  <!-- ===== -->

  <!-- Órbitas -->

  <!-- Órbita Mercúrio -->
  <group>
    <scale X="128.828" Y="128.828" Z="128.828" />
    <rotate angle="90" axisX="1" />
    <colour R="255" G="255" B="255" />
    <models>
      <model file="orbit.3d" />
    </models>
  </group>

  <!-- Órbita Venus -->
  <group>
    <scale X="240.62" Y="240.62" Z="240.62" />
    <rotate angle="90" axisX="1" />
    <colour R="255" G="255" B="255" />
    <models>
      <model file="orbit.3d" />
    </models>
  </group>

  <!-- Órbita Terra -->
  <group>
    <scale X="332.81" Y="332.81" Z="332.81" />
    <rotate angle="90" axisX="1" />
    <colour R="255" G="255" B="255" />
    <models>
      <model file="orbit.3d" />
    </models>
  </group>

  <!-- Órbita Marte -->
  <group>
    <scale X="507.201" Y="507.201" Z="507.201" />
    <rotate angle="90" axisX="1" />
```

```

        <colour R="255" G="255" B="255" />
        <models>
            <model file="orbit.3d" />
        </models>
    </group>

    <!-- Órbita Júpiter -->
    <group>
        <scale X="1100" Y="1100" Z="1100" />
        <rotate angle="90" axisX="1" />
        <colour R="255" G="255" B="255" />
        <models>
            <model file="orbit.3d" />
        </models>
    </group>

    <!-- Órbita Saturno -->
    <group>
        <scale X="1400" Y="1400" Z="1400" />
        <rotate angle="90" axisX="1" />
        <colour R="255" G="255" B="255" />
        <models>
            <model file="orbit.3d" />
        </models>
    </group>

    <!-- Órbita Urano -->
    <group>
        <scale X="1700" Y="1700" Z="1700" />
        <rotate angle="90" axisX="1" />
        <colour R="255" G="255" B="255" />
        <models>
            <model file="orbit.3d" />
        </models>
    </group>

    <!-- Órbita Neptuno -->
    <group>
        <scale X="2000" Y="2000" Z="2000" />
        <rotate angle="90" axisX="1" />
        <colour R="255" G="255" B="255" />
        <models>
            <model file="orbit.3d" />
        </models>
    </group>

    <!-- Órbita Plutão -->
    <group>
        <scale X="2200" Y="2200" Z="2200" />
        <rotate angle="90" axisX="1" />
        <colour R="255" G="255" B="255" />

```

```

    <models>
      <model file="orbit.3d" />
    </models>
  </group>

  <!-- ===== -->

  <!-- Planetas e Satélites Naturais-->

  <!-- Mercurio -->
  <group>
    <translate X="128.828"/>
    <scale X="1.365" Y="1.365" Z="1.365" />
    <colour R="186" G="184" B="181" />
    <models>
      <model file="sphere.3d" />
    </models>
  </group>

  <!-- Venus -->
  <group>
    <translate X="240.62" />
    <scale X="3.386" Y="3.386" Z="3.386" />
    <colour R="237" G="184" B="100" />
    <models>
      <model file="sphere.3d" />
    </models>
  </group>

  <!-- Terra -->
  <group>
    <translate X="332.81" />
    <scale X="3.569" Y="3.569" Z="3.569" />
    <colour R="61" G="129" B="224" />
    <models>
      <model file="sphere.3d" />
    </models>
    <!-- Lua -->
    <group>
      <translate Z="4.423" />
      <scale X="0.486" Y="0.486" Z="0.486" />
      <colour R="207" G="207" B="207" />
      <models>
        <model file="sphere.3d" />
      </models>
    </group>
  </group>

  <!-- Marte -->
  <group>

```

```

<translate X="507.201" />
<scale X="1.9" Y="1.9" Z="1.9" />
<colour R="240" G="95" B="70" />
<models>
  <model file="sphere.3d" />
</models>
<!-- Phobos -->
<group>
  <translate Z="3" />
  <scale X="0.05" Y="0.05" Z="0.05" />
  <colour R="155" G="155" B="155" />
  <models>
    <model file="sphere.3d" />
  </models>
</group>
<!-- Deimos -->
<group>
  <translate Z="-3" />
  <scale X="0.025" Y="0.025" Z="0.025" />
  <colour R="133" G="113" B="113" />
  <models>
    <model file="sphere.3d" />
  </models>
</group>
</group>

<!-- Cintura de Asteroides -->

<group>
  <scale X="700" Y="0" Z="700" />
  <rotate angle="90" axisX="1" />
  <colour R="110" G="61" B="61" />
  <models>
    <model file="asteroidsBelt.3d" />
  </models>
</group>

<!-- Jupiter -->
<group>
  <translate X="1100"/>
  <scale X="40.0" Y="40.0" Z="40.0" />
  <colour R="169" G="148" B="179" />
  <models>
    <model file="sphere.3d" />
  </models>

<!-- Io -->
<group>
  <translate Z="1.7"/>
  <scale X="0.03643" Y="0.03643" Z="0.03643" />
  <colour R="209" G="150" B="23" />

```

```

        <models>
            <model file="sphere.3d" />
        </models>
    </group>

    <!-- Europa -->
    <group>
        <translate Z="-1.5"/>
        <scale X="0.03121" Y="0.03121" Z="0.03121" />
        <colour R="141" G="152" B="131" />
        <models>
            <model file="sphere.3d" />
        </models>
    </group>

    <!-- Ganymede -->
    <group>
        <translate X="1" Y="2" Z="1"/>
        <scale X="0.0562" Y="0.0562" Z="0.0562" />
        <colour R="95" G="99" B="150" />
        <models>
            <model file="sphere.3d" />
        </models>
    </group>

    <!-- Calisto -->
    <group>
        <translate Y="1.5"/>
        <scale X="0.0482" Y="0.0482" Z="0.0482" />
        <colour R="150" G="101" B="24" />
        <models>
            <model file="sphere.3d" />
        </models>
    </group>
</group>

<!-- Saturno -->
<group>
    <translate X="1400"/>
    <scale X="33.72" Y="33.72" Z="33.72" />
    <colour R="196" G="169" B="104" />
    <models>
        <model file="sphere.3d" />
    </models>
    <!-- Rings -->
    <!-- Inner Ring -->
    <group>
        <rotate angle="63" axisX="1" />
        <scale X="1.5" Y="1.5" Z="0.1" />
        <colour R="255" G="204" B="102" />
        <models>

```

```

        <model file="ring.3d" />
    </models>
</group>
<!-- Outer Ring -->
<group>
    <rotate angle="63" axisX="1" />
    <scale X="1.9" Y="1.9" Z="0.1" />
    <colour R="176" G="142" B="84" />
    <models>
        <model file="ring.3d" />
    </models>
</group>
<!-- Titã -->
<group>
    <translate X="1.2" Y="-1.2" Z="1"/>
    <scale X="0.05149" Y="0.05149" Z="0.05149" />
    <colour R="95" G="200" B="161" />
    <models>
        <model file="sphere.3d" />
    </models>
</group>
<!-- Reia -->
<group>
    <translate Y="1" Z="-1.4"/>
    <scale X="0.01527" Y="0.01527" Z="0.01527" />
    <colour R="155" G="155" B="155" />
    <models>
        <model file="sphere.3d" />
    </models>
</group>
<!-- Jápeto -->
<group>
    <translate Z="1"/>
    <scale X="0.0147" Y="0.0147" Z="0.0147" />
    <colour R="150" G="140" B="110" />
    <models>
        <model file="sphere.3d" />
    </models>
</group>
<!-- Diome -->
<group>
    <translate X="1.7" Y="1.7" />
    <scale X="0.01123" Y="0.01123" Z="0.01123" />
    <colour R="155" G="155" B="155" />
    <models>
        <model file="sphere.3d" />
    </models>
</group>
</group>

<!-- Urano -->

```

```

<group>
  <translate X="1700"/>
  <scale X="14.298" Y="14.298" Z="14.298" />
  <colour R="61" G="194" B="201" />
  <models>
    <model file="sphere.3d" />
  </models>
  <!-- Rings -->
  <group>
    <scale X="1.5" Y="1.5" Z="0" />
    <rotate angle="30" axisX="1" />
    <colour R="100" G="100" B="100" />
    <models>
      <model file="orbit.3d" />
    </models>
  </group>
</group>

<!-- Neptuno -->
<group>
  <translate X="2000"/>
  <scale X="13.858" Y="13.858" Z="13.858" />
  <colour R="6" G="78" B="194" />
  <models>
    <model file="sphere.3d" />
  </models>
</group>

<!-- Plutão -->
<group>
  <translate X="2200"/>
  <scale X="3" Y="3" Z="3" />
  <colour R="186" G="143" B="227" />
  <models>
    <model file="sphere.3d" />
  </models>
</group>

<!-- Cintura de Kuiper -->
<group>
  <scale X="3000" Y="0" Z="3000" />
  <rotate angle="90" axisX="1" />
  <colour R="110" G="61" B="61" />
  <models>
    <model file="asteroidsBelt.3d" />
  </models>
</group>

</scene>

```
