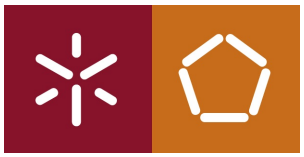

Universidade do Minho

Computação Gráfia

2018/2019

Phase 2 – Geometric Transforms

Trabalho realizado por:	*Número de Aluno:*
Nelson Gonçalves	A78713
João Aloísio	A77953
César Henriques	A64321
Ricardo Ponte	A79097

25 de Março de 2019

Conteúdo

1 Introdução

A vertente prática da unidade curricular de Computação Gráfica tem como base a utilização do OpenGL, recorrendo à biblioteca GLUT, para a construção de modelos 3D.

A produção dos modelos referidos envolve diversos temas, entre os quais, transformações geométricas, curvas e superfícies, iluminação e texturas. Com o objetivo de aplicar e demonstrar a aprendizagem destes tópicos, foi proposta a realização de um projeto prático, que se encontra dividido em quatro fases, cada uma com uma data específica de entrega.

A existência destas quatro fases tem como finalidade a organização e simplificação do desenvolvimento do projeto, sendo que cada uma deverá respeitar os requisitos básicos, inicialmente estipulados no enunciado do trabalho.

Este relatório diz respeito à realização da segunda fase do projeto mencionado, que tem como assunto principal as Transformações Geométricas. Para uma melhor compreensão do objetivo desta fase, na secção seguinte será devidamente descrito o problema a resolver.

2 Descrição do Problema

A segunda fase deste projecto tem como objectivo o desenvolvimento de novas funcionalidades aplicadas ao Motor desenvolvido na primeira fase. Este deve agora processar Transformações Geométricas tais como translações, escalas e rotações, descritas no ficheiro de configuração escrito em XML. Nesta fase pretende-se representar um modelo estático do Sistema Solar (sol, planetas e satélites). Assim, os requisitos para esta segunda fase do projeto, são:

- Gerador (recebe como parâmetros o tipo da primitiva gráfica, parâmetros relativos ao modelo e o nome do ficheiro onde vão ser guardados os vértices):
  - **ring file**.3d ***radius outter_radius slices***
    Anel: coroa circular no plano xz, centrado na origem, desenhado tanto no sentido positivo, como no sentido negativo do eixo y.
- Motor :
  - lê os grupos do ficheiro XML.
  - **armazena as transformações e os modelos** de cada grupo em memória.
  - **desenha**, utilizando triângulos, as primitivas relativas aos modelos existentes em cada grupo, com as respectivas transformações associadas.
- Modelo do Sistema Solar :
  - utiliza a **esfera e o anel** como primitivas, para representar o sol, os planetas e os satélites.
  - serve-se de uma **escala** calculada previamente.

3 Resolução do Problema

Nesta fase serão realizadas alterações, tanto ao Motor como ao Gerador desenvolvidos anteriormente, com o objectivo de cumprir com os requesitos propostos para a segunda fase.

3.1 Gerador

De forma a demonstrar devidamente o modelo do sistema solar, necessitou-se de atualizar o Gerador desenvolvido na fase anterior. Posto isto, decidiu-se criar uma primitiva que calcula os pontos necessários para desenhar uma coroa circular, que será utilizada para representar os anéis de Saturno.

3.1.1 Ring

De modo a representar o planeta Saturno, foi necessário criar um anel que envolve a sua forma esférica. Este anel é composto por triângulos que são desenhados nos sentidos positivo e negativo do eixo y, e é centrado na origem sobre os eixos x e z.

Este anel tem um raio interior, *radious*, um raio exterior, *outter_radious*, e um número de *slices* pelo qual vai ser dividido.

- $\alpha = \frac{2\pi}{slices} \times \mathrm{j}, \mathrm{j} = [0, slices]$

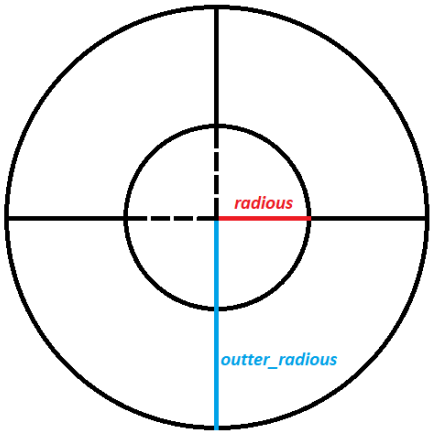

Figura 1: Representação simples da divisão do anel em slices (exemplo com 4 slices)

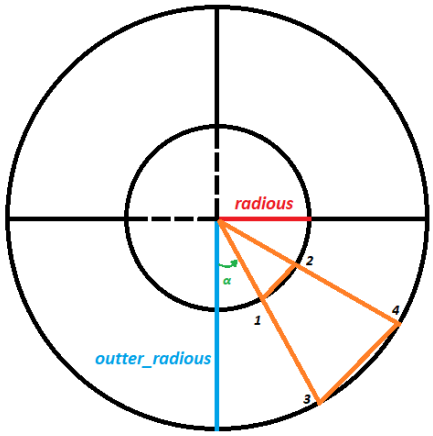

Figura 2: Representação simples da divisão do anel em slices (exemplo com 4 slices)

É possível calcular as coordenadas um ponto recorrendo à distância deste à origem (*radious* - raio interior e *outter_radious* - raio total) e ao ângulo α. Isto permitirá calcular todos os pontos x e z à volta do anel tendo em conta que a coordenada y tem o valor 0. E de notar ainda que o ângulo α vai aumentar $\frac{2\pi}{slices}$ por cada divisão, até à divisão *slices*. As fórmulas para o cálculo de x e z são as seguintes:

- $x = distance \times sin(\alpha)$
- $z = distance \times cos(\alpha)$

A *distance* vai depender do ponto que se pretende calcular.Os pontos 1, 2 serão calculados com base no *radius* enquanto que os pontos 3 e 4 serão calculados usando o *outter_radius*. Tendo isto em conta os valores das coordenadas x e z são caculados através das seguintes equações:

- $x1 = radious \times \sin(\alpha 1)$
- $z1 = radious \times \cos(\alpha 1)$
- $x2 = radious \times \sin(\alpha 2)$
- $z2 = radious \times \cos(\alpha 2)$
- $x3 = outter\_radious \times \sin(\alpha 1)$
- $z3 = outter\_radious \times \cos(\alpha 1)$
- $x4 = outter\_radious \times \sin(\alpha 2)$
- $z4 = outter\_radious \times \sin(\alpha 2)$

Sendo que os valores de α são calculados com as equações:

- $\alpha 1 = \text{j} \times \frac{2\pi}{slices}$
- $\alpha 2 = (\text{j}+1) \times \frac{2\pi}{slices}$

Usando a regra leccionada nas aulas de Computação Gráfica, a "regra da Mão Direita", podemos desenhar o anel, tanto no sentido positivo, como no sentido negativo do eixo y. Para o sentido positivo, conseguiu-se desenhar os triângulos formados (pelos pontos representados na figura 2) usando as ordens 1,2,4 e 2,3 4. respectivamente. Para obter o anel orientado no sentido negativo, desenhamos os triângulos formados pelos pontos 2,1,4 e 4,3,2.

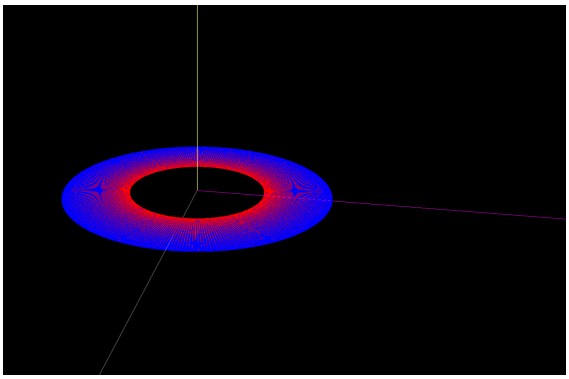

Figura 3: *Representação do anel (visto de cima) utilizando o engine*

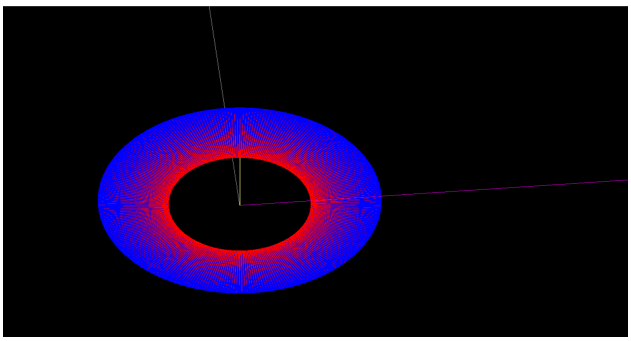

Figura 4: *Representação do anel (visto de baixo) utilizando o engine*

3.2 Motor

Visto que nesta fase o ficheiro XML possuí uma constituição diferente da fase anterior, foi necessário adaptar o método de parsing ao ficheiro. Para além disto, foi necessário armazenar mais informações para além dos vértices e dos modelos, o que desencadeou a criação de novas estruturas, tais como *groupData* que permite o armazenamento dos valores atribuídos às coordenadas nas transformações,

```
typedef struct groupData{
    float traX, traY, traZ;
    float angle, rotX, rotY, rotZ;
    float scaleX, scaleY, scaleZ;

}Group;
```

pathInfo que guarda o *path* para o ficheiro *3d* do *model* associando assim as alterações que irão ser feitas ao modelo em questão,

```
typedef struct pathInfo{
    Path path;
    float traX=0, traY=0, traZ=0;
    float angle=0, rotX=0, rotY=0, rotZ=0;
    float scaleX=1, scaleY=1, scaleZ=1;
} Paths;
```

e *modelData* que armazena os dados necessários para desenhar o respectivo modelo com as transformações existentes no XML,

```
typedef struct modelData{
    Model model;
    float traX=0, traY=0,traZ=0;
    float angle=0, rotX=0, rotY=0, rotZ=0;
    float scaleX=1, scaleY=1, scaleZ=1;
}ModelData;
```

3.2.1 Leitura do ficheiro XML

A biblioteca usada para ler os ficheiros XML continua a ser a *TinyXML-2*. Sendo assim, foi novamente usada a função implementada na fase anterior *readXML* com as alterações necessárias.

Nesta fase é necessário retirar a informação contida no *group* existente no ficheiro XML, e para isso foi implementada uma função auxiliar *groupAux* que armamezena todos os dados contidos na estrutura *groupData* criada para o efeito. Esta função, caso encontre o campo *models*, chama uma outra função auxiliar *readModels* que guarda os dados na estrutura *pathInfo*. Caso encontre o campo *group* novamente ao ler o ficheiro XML, vai ser chamada a *groupAux* recursivamente.

3.2.2 Desenho das Primitivas

Para o desenho das primitivas, foi previamente usada a função *loadXML* explicada na fase anterior com pequenas alterações, sendo que armazena os dados na estrutura *modelData*. Após a inserção de todos os modelos contidos no ficheiro XML, é precorrido o vetor *modelz* pela função *drawTheFiles* que faz *push* e as devidas alterações com o auxílio das funções *glPushMatrix*, *glTranslatef*, *glRotatef* e *glScalef*. Tendo

concluido o desenho através dos vértices que constituem o triângulo com o auxílio da função *glVertex3f*, é usada a função *glPopMatrix* para retirar a matriz antiga sem transformações.

3.2.3 Câmara

Foi implementada uma câmara FPS que movimenta-se segundo ângulos proporcionais às posições horizontal/vertical do cursor na janela. Tendo em conta a figura abaixo, supondo que a câmara encontra-se no ponto P e que o seu ponto de referência esteja em R, a direção do movimento da câmara é calculada segundo os ângulos α e β aplicados do vetor D.

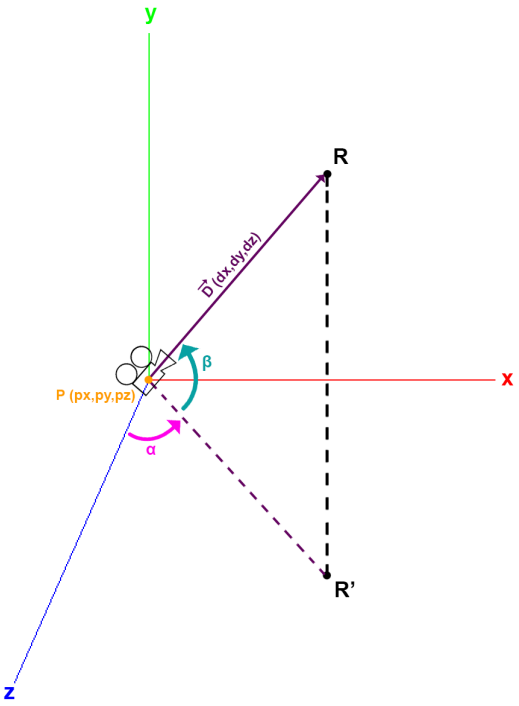

Figura 5: *Disposição do ponto de referência relativamente à câmara*

A função *gluLookAt* permite definir vários parâmetros, dentro destes , dado o contexto, destacam-se:

- Posição da câmara
- Posição do ponto de referência do ponto focado pela câmara
- Direção do vector up

A posição da câmara é dada pelas coordenadas de P (px,py,pz) enquanto que as coordenadas de R (ponto de referência) são obtidas segundo a direção do vetor D. Sendo assim, a função *glutLookAt* receberá os seguintes parâmetros:

```
gluLookAt(px,py,pz,
                  px + dx,py + dy,pz + dz,
                  0.0f  ,  1.f  ,  0.0f)
```

Os valores iniciais das coordenadas P e D foram definidos, respetivamente, da seguinte forma:

```
float px = 0.0;
float py = 0.0;
float pz = 20.0;
float dx = 0.0;
float dy = 0.0;
float dz = -1;
```

Tendo em conta a figura 5 assim como os ângulos α e β conclui-se que as coordenadas do vetor D são dadas por :

dx = $\sin(\alpha)$, $\alpha = [0, 2\pi]$
dy = - $\cos(\alpha)$, α =$[0, 2\pi]$
dz = $\sin(\beta)$, β =$[\frac{-\pi}{2}, \frac{\pi}{2}]$

De realçar que dy é simétrico de $\cos(\alpha)$ de forma a permitir o movimento a coincidir com o sentido desejado. Como foi referido anteriormente, o vetor D indica apenas a direção do movimento da câmara porém também deve ser possível a esta aproximar/afastar dos objectos. Sendo assim e como D é um vetor unitário, para deslocar a câmara k unidades segue-se a fórmula indicada abaixo.

P' = P + $k \times D$

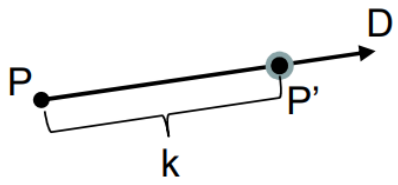

Figura 6: *Deslocação da câmara k unidades segundo D*

Tendo isto em conta, a nova posição de P será dada por:

px = px + $dx \times k$
py = py + $dy \times k$
pz = pz + $dz \times k$

Com as fórmulas apresentadas acima é possível mover a camara em qualquer direção de acordo com o vetor D. Porém o grupo optou por adicionar a possibilidade de movimento perpendicular da câmara numa dada direção. Isto permitirá mover a câmara para a esquerda/direita relativamente ao ponto de referência.

Posteriormente foram criadas mais duas variáveis (dxP e dzP) que representam as coordenadas do movimento perpendicular. Estas coordenadas e as novas coordenadas px e py são dadas por :

$\mathbf{dxP} = \sin(\alpha + \frac{\pi}{2})$

$\mathbf{dzP} = -\cos(\alpha + \frac{\pi}{2})$

$\mathbf{py} = py + dxP \times k$

$\mathbf{pz} = pz + dzP \times k$

3.2.4 Teclado

A função *gluLookAt* possibilita o movimento da câmara sendo que a posição da câmara e do ponto de referência baseiam-se no valor das variáveis globais correspondentes às coordenadas do ponto P (px,py,pz) e do vector D(dx,dy,dz). Para alterar a posição da câmara usam se as teclas W que movimenta k(=1) unidades de acordo com o vector D e S movimenta a câmara k unidades na direção contrária. Como foi referido, disponibilizou-se a opção de movimento perpendicular à direcção de D , que é feito com as teclas A e D. À semelhança da fase anterior, é permitido afastar ou aproximar de um ponto sendo que a tecla + serve para aproximar enquanto que a tecla - serve para distanciar. Também foram adicionadas as funcionalidades de ativar/desativar os eixos ao premir a tecla F e ver as figuras geométricas desenhadas por linhas apenas, sendo que para isso basta premir a tecla L.

A função *process_keys* trata da interação do teclado sendo que posteriormente esta função é passada como argumento à função *glutKeyboardFunc*.

3.2.5 Rato

O rato também será utilizado na movimentação da câmara nomeadamente para alterar o vetor D. O direcionamento horizontal diz respeito ao ângulo α enquanto que o direcionamento vertical diz respeito ao ângulo β.

Supondo que a origem da janela coincide com o seu centro, ao efetuar-se a comparação entre o posicionamento do cursor com os valores de α e β, obteve-se:

- x $\in [\frac{-width}{2}, \frac{width}{2}]$
- y $\in [\frac{-height}{2}, \frac{height}{2}]$

Assim sendo pode-se assumir que à medida x aumenta/diminui na janela α também aumenta/diminui na mesma proporção. O mesmo acontece com y e β sendo que sempre que y aumenta/diminui na janela β irá fazer o mesmo em igual proporção.

Posto isto, concluiu-se que as frações dos ângulos α e β são dadas por : $\mathbf{aStep} = \frac{x}{width}$, $\mathbf{bStep} = \frac{y}{height}$

No entanto como $\alpha \in [0, 2\pi]$ e $\beta \in [\frac{-\pi}{2}, \frac{\pi}{2}]$ temos que : $\alpha = aStep \times 2\pi$, $\beta = bStep \times \pi$

A função *mouseMove* irá ser responsável por lidar com o movimento do rato, sendo que esta é invocada nas funções *glutMotionFunc* e *glutPassiveMotionFunc* do *GLUT*.

De modo a não interferir com a principal função do rato, a funcionalidade de alterar a câmara só será disponibilizada quando o botão esquerdo estiver premido. Isto conseguiu-se recorrendo a uma variável global *clicked* que indica se o botão está premido. Esta funcionalidade é implementada na função *mouseClick* que posteriormente é passada como argumento à função do *GLUT glutMouseFunc*.

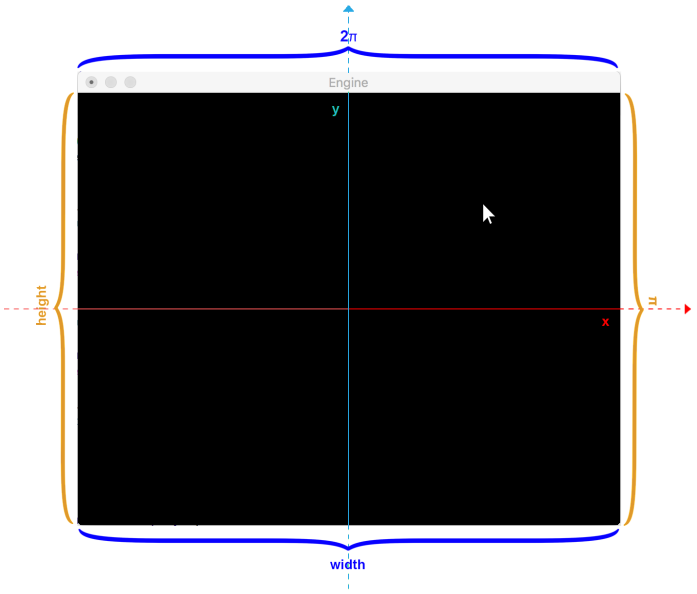

Figura 7: *Sistema de coordenadas da janela*

3.3 Modelo do Sistema Solar

No modelo do sistema solar, descrito no ficheiro XML, serão representados o Sol, os 8 planetas (Mercúrio, Vênus, Terra, Marte, Júpiter, Saturno, Urano e Neptuno) e os maiores satélites de cada planeta (que possuam satélites). De forma a tornal o modelo mais realista, será representado os aneis de saturno. Para esta tarefa foram utilizadas as primitavas da esfera e do anel.

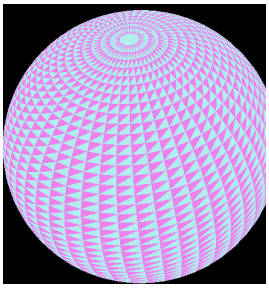

Figura 8: Esfera com raio 1, 50 slices e 50 stacks

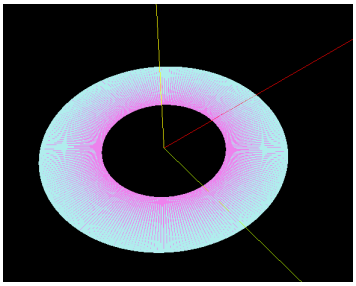

Figura 9: Anel com raio interior 1, raio exterior 2 e 50 slices

Visto que as primitivas tem uma dimensão e posição fixas, é necessario recorrer a transformações,ou seja, escalas, rotações e translações, de modo a desenhar as figuras o mais semelhante à realidade. Como existe uma discrepância entre as dimensões reais dos diferentes elementos do sistema não foi possivel utilizar uma escala completamente de acordo com o modelo real, mas sim aproximações de modo a ter um sistema o mais parecido com a realidade. Ao desenvolver o modelo tivemos de ter atenção ao facto de um subgrupo herdar todas as tranformações do grupo que está inserido.

- o translate de um grupo seja somado ao translate do subgrupo;
- o scale de um grupo multiplique o valor do scale do subgrupo;
- o valor do rotate do grupo seja somado ao rotate do subgrupo.

Para começar vamos definir o valor de rotação de cada planeta e de cada satélite de acordo com o eixo de rotação e da rotação orbital. Para isso aplicamos um rotate sobre o eixo dos Z com o valor do ângulo de inclinação. Como este ângulo é positivo para o lado esquerdo, é necessário aplicar o valor negativo ao ângulo para que a inclinação seja para o lado direito. Como os satelites herdam as tranformações dos seus planetas é neceseário anular estas rotações de forma a ter o ângulo correto.

Nome	**Inclinação (graus)**	***Rotate***
Mercúrio	0.1	-0.1
Vénus	177	-177
Terra	23.5	-23.5
Lua	5.15	18.35
Marte	25	-25
Fobos	1	24
Júpiter	3	-3
Ganimedes	0.2	2.8
Saturno	27	-27
Titã	0.33	26.67
Urano	98	-98
Titânia	0.14	97.86
Neptuno	30	-30
Tritão	157.35	-127.35

Tabela 1: Rotações utilizadas para cada Planeta e Satélite do modelo do sistema Solar.

De seguida, definimos a dimensão de cada Planeta e Satélite. Inicialmente estabelecemos uma escala de 200 que representa a distancia entre o Sol e o planeta mais distante(Neptuno), no entanto, como o valor era muito pequeno multiplicamos esse mesmo valor por 2000. Apesar de termos feito essa multiplicação existiam planetas ou satélites que ou continuavam muito pequenos ou demasiado grandes,como por exemplo, o Fobos e o Sol, nesses casos fizemos alguns ajustes.

Tipo	**Nome**	**Raio (km)**	**Escala**	**Escala*2000**	***Scale***
Estrela	Sol	69 600 000	3.10165	6203	8
Planeta	Mercúrio	2 440	0.000109	0.218	0.218
Planeta	Vénus	6 052	0.00027	0.54	0.54
Planeta	Terra	6371	0.000284	0.568	0.568
Satélite	Lua	1 737	0.000077	0.154	0.4
Planeta	Marte	3 390	0.000151	0.302	0.302
Satélite	Fobos	11.1	0.000000494	0.00099	0.5
Planeta	Júpiter	69 911	0.0031	6.2	2.2
Satélite	Ganimedes	2 634	0.000117	0.234	0.2
Planeta	Saturno	58 232	0.002595	5.19	2
Satélite	Titã	2 575	0.000115	0.23	0.12
Planeta	Urano	25 362	0.00113	2.26	1.2
Satélite	Titânia	789	0.000035	0.07	0.15
Planeta	Neptuno	24 622	0.001097	2.194	1.1
Satélite	Tritão	1 353	0.00006	0.12	0.2

Tabela 1: Escalas utilizadas para cada Estrela,Planeta e Satélite do modelo do sistema Solar, tendo em conta o seu raio.

Inicialmente para os valores das distâncias entre os Planetas e o Sol utilizamos a escala descrita em cima, no entanto, devido a estas distancias serem demasiado grandes e não conseguirmos visualizar bem o modelo a escala foi reduzida para 75(Distancia ente o Sol e neptuno). Para a distancia entre os Satélites e os Planetas foi necessario aumentar a distancia entre eles de maneira a tornar os Satélites perceptíveis. Como estes herdam a translação do eixo dos X apenas foi necessario adicionar uma translação no eixo dos Y, essa translação tinha como consideração a distancia entre o Satélite e o Planeta mas como referido em cima foi necessário aumentar essa escala.

Nome	**Distância(km)**	**Escala**	**Translate**
Sol	-	-	-
Mercúrio-Sol	56 847 190	1.265	X=1.265
Vénus-Sol	107 710 466	2.4	X= 2.4
Terra-Sol	149 597 870	4.2	X= 4.2
Lua-Terra	394 400	2	Y=2
Marte-Sol	227 388 763	6.065	X= 6.065
Fobos-Marte	9 380	1.85	Y=1.85
Júpiter-Sol	777 908 927	17.335	X= 17.335
Ganimedes-Júpiter	1 070 400	2.43	Y=2.43
Saturno-Sol	1 421 179 771	31.65	X=31.65
Titã-Saturno	1 221 870	3.84	Y=3.84
Urano-Sol	2 872 279 117	50.995	X= 50.995
Titânia-Urano	436 300	2.28	Y=-2.28
Neptuno-Sol	4 487 936 121	75	X= 75
Tritão-Neptuno	354 800	2.23	Y=-2.23

Tabela 3: Translações utilizadas para cada Planeta e Satélite do modelo do sistema solar, tendo em conta as distâncias entre o Sol.

Finalmente, para o anel de Saturno, como este está no grupo do planeta vai herdar a translação do mesmo e vai estar centrado. Sendo assim, é apenas necessário aplicar uma scale de modo a que este permaneça ligeiramente afastado de Saturno. Para este valor da escala aplicada ao anel foi também preciso ter em

atenção o valor da escala aplicada ao planeta.

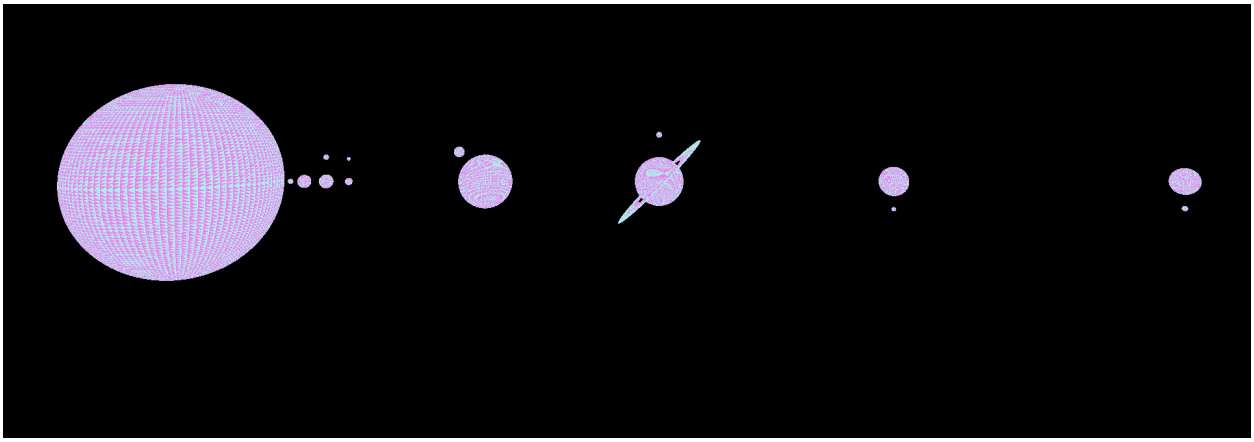

Tabela 1: Representação do Sistema Solar.

4 Conclusões

De uma forma geral, o grupo considerou a realização desta segunda fase do projecto satisfatória, pois foram cumpridos todos os pré-requesitos necessários.
Nesta fase conseguiu-se realizar as modificações necessárias ao Motor para este processar as várias tranformações geográficas pretendidas (translações, escalas e rotações), estas descritas num ficheiro de configuração em XML.
Foi desenhado um modelo estático do Sistema Solar como pretendido e, para uma navegação e análise dos resultados obtidos ao longo do projecto, foi desenvolvida a possibilidade da utilização do rato para orientação no modelo. Ainda na primeira fase do projecto, foi necessária a utilização de caminhos absolutos para a leitura do ficheiro *Config.xml*, entrave que foi ultrapassado sendo que agora é possível utilizar caminhos relativos.
Resumindo, a realização desta segunda fase será imprescindível para o contínuo desenvolvimento deste projecto.