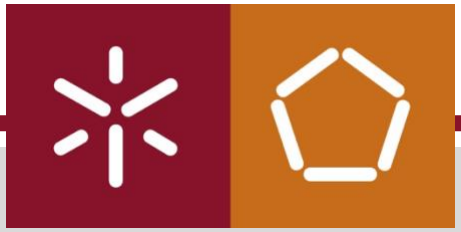
UNIVERSIDADE DO MINHO



Mestrado Integrado em Engenharia Informática

PROJETO DE COMPUTAÇÃO GRÁFICA FASE 2 : TRANSFORMAÇÕES GEOMÉTRICAS

março, 2020




GRUPO 10
Carolina Cunha, A80142 | Hugo Faria, A81283 | João Diogo Mota, A80791 | Rodolfo Silva, A81716

ÍNDICE

ÍNDICE DE FIGURAS

BREVE DESCRIÇÃO DO ENUNCIADO PROPOSTO

Prevê-se, com este trabalho, o desenvolvimento de um mecanismo 3D baseado em mini gráficos de cenas e fornecer exemplos de uso que mostrem o seu potencial. Este trabalho será dividido em quatro fases distintas, de modo a facilitar a elaboração do mesmo. A conceção e desenvolvimento do trabalho seguirá uma abordagem suportada pela ferramenta *OpenGL* e a linguagem C++.

A segunda fase do projeto consistirá na criação de cenas hierárquicas usando transformações geométricas. Cada cena é definida como uma árvore em que cada nodo contém um conjunto de transformações geométricas (translação, rotação e escala) e, opcionalmente, um conjunto de modelos. Cada nó pode também ter nós filhos.

A cena demo requerida para esta fase é um modelo estático do sistema solar, incluindo o sol, planetas e luas definidas numa hierarquia.

TRANSFORMAÇÕES GEOMÉTRICAS

TRANSLAÇÃO

A translação move um ponto, ou um conjunto de pontos, numa determinada direção e comprimento. As translações provam-se úteis para a movimentação de objetos, construção de cenas 3D, "posicionamento" da câmara ou execução de animações. [1]

Na Figura 1 é visível a representação da translação do ponto p através do vetor $\vec{v}$.

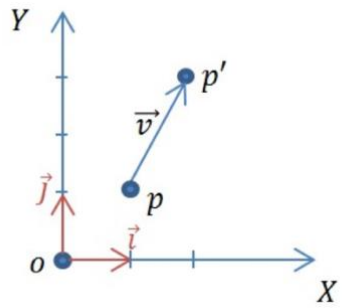


Figura 1: Translação do ponto p

Matematicamente, esta translação pode ser representada por [1]:

$$p' = p + \vec{v} \quad (1)$$

Intuitivamente se transforma esta equação na que se segue [1]:

$$p' = (p_x + v_x\,, p_y + v_y\,, 1) \quad (2)$$

Seguidamente, e decompondo a equação anterior em elementos de um sistema de coordenadas, versores e origem, tem-se [1] :

$$p' = p + \vec{v} = p_x\vec{\imath} + p_y\vec{\jmath} + o + v_x\vec{\imath} + v_y\vec{\jmath} \quad (3)$$

Agrupando os últimos três elementos em o', assumindo que se trata da soma da origem com o vetor $\vec{v}$, tem-se [1]:

$$o' = o + v_x + v_y \tag{4}$$

Reescrevendo a equação (3) na forma matricial obtém-se [1]:

$$p' = [\vec{\imath}\ \vec{\jmath}\ o'] \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} \tag{5}$$

Considerando $\vec{\imath}(1,0,0)$, $\vec{\jmath}(0,1,0)$ e $o(0,0,1)$, o' pode ser escrito da seguinte forma [1]:

$$o' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ 1 \end{bmatrix} \tag{6}$$

Consequentemente, p' pode ser escrito da forma [1] :

$$p' = \begin{bmatrix} 1 & 0 & v_x \\ 0 & 1 & v_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ 1 \end{bmatrix} \tag{7}$$

Deste modo, confirma-se que, quando a um ponto p com coordenadas homogéneas $(\mathrm{px}, \mathrm{py}, 1)$ é realizada uma translação com o vetor $\vec{v}$ com coordenadas $(v_x, v_y, 0)$, o ponto resultante tem as coordenadas $(p_x + v_x\ , p_y + v_y\ , 1)$.

Um resultado importante é que uma matriz 3x3 pode ser usada para a translação num ponto cartesiano 2D .

No que diz respeito à forma matricial de uma translação por um vetor $\vec{v}$ sobre um ponto 3D, esta pode ser representada do seguinte modo [1]:

$$T = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8)$$

Dada a matriz da equação (8), o ponto p em coordenadas homogéneas é dado por [1]:

$$p' = Tp \quad (9)$$

Se for utilizado o vetor $\vec{v}$ para construir a matriz de translação, então -$\vec{v}$ irá permitir a obtenção da translação inversa T = -T [1].

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -v_x \\ 0 & 1 & 0 & -v_y \\ 0 & 0 & 1 & -v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (10)$$

Como esperado, a seguinte equação é verdadeira [1]:

$$TT^{-1} = I \quad (11)$$

A aplicação prática destes conceitos foi aplicada através da função *glTranslatef* da biblioteca *Glut* do seguinte modo, em que nodo é a estrutura de dados xmlData.

```
glTranslatef(nodo.getTranslateX(),nodo.getTranslateY(),nodo.getTranslateZ());
```

Sirva de exemplo a seguinte condição de translação:

```
<translate X="5" Y="10" Z="15"/>
```

No exemplo apresentado, o centro do objeto vai ser deslocado 5 unidades no eixo dos xx, 10 unidades no eixo dos yy e 15 unidades no eixo dos zz.

A escala é a transformação que altera o tamanho ou forma de um objeto.

Esta é necessária para lidar com modelos construídos em unidades diferentes. Pode ainda ser usada quando se pretende utilizar o mesmo modelo para situações distintas.

Escalar implica multiplicar as coordenadas de cada ponto que define o modelo por um certo valor [1].

$$p' = (s_x, s_y) \times p = (s_x \times p_x, s_y \times p_y \quad (12)$$

A equação (12) define a escala 2D em coordenadas cartesianas. Geometricamente, esta equação pode ser representada da seguinte forma:

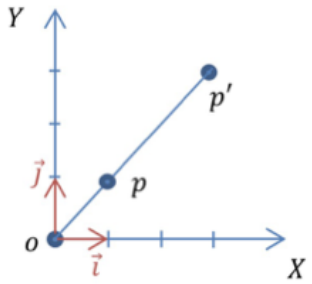


Figura 2: Escala do ponto p

A matriz para esta transformação é representada por [1]:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (13)$$

Consequentemente, escalar um ponto é definido por [1]:

$$p' = Sp = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x \times p_x \\ s_y + p_y \\ 1 \end{bmatrix} \quad (14)$$

De notar que, no caso de $s_z \neq 1$, a matriz S é dada por [1]:

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \quad (15)$$

Desta forma, a matriz S para coordenadas 3D é representada por [1]:

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (16)$$

A aplicação prática destes conceitos foi aplicada através da função *glScalef* da biblioteca *Glut* do seguinte modo.

```
glScalef(nodo.getScaleX(),nodo.getScaleY(),nodo.getScaleZ());
```

Sirva de exemplo a seguinte condição de escala:

```
<scale X="20" Y="20" Z="20"/>
```

No exemplo apresentado, o objeto vai ser redimensionado 20 unidades nos eixos dos xx, yy e zz.

A rotação é outra transformação muito útil, uma vez que os modelos geralmente exigem que uma rotação seja exibida adequadamente em uma cena 3D [1].

ROTAÇÃO SOBRE OS EIXOS PRINCIPAIS

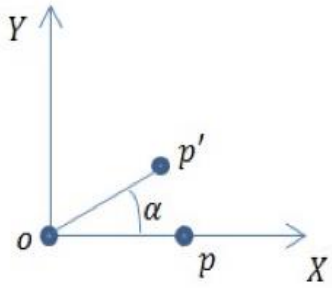


Figura 3: Rotação do ponto p

Na Figura 3 constata-se que as rotações em sistemas de 2D são realizadas no sentido contrarrelógio. No entanto, quando sobre um cenário 3D, as rotações denominam-se eixo de rotação. Por este motivo, uma rotação no plano XY é realizada sobre o eixo Z. A utilização da regra da mão direita permite saber a orientação da rotação [1].

Recorrendo à trigonometria, e considerando $p(p_x, 0)$, a rotação deste ponto por um ângulo α é dada por [1]:

$$p' = (p_x \times \cos(\alpha), p_x \times \sin(\alpha)) \quad (17)$$

Considerando agora uma rotação arbitrária, realizando uma rotação do ponto p' por β, tem-se a seguinte equação, visível na Figura 4 [1]

$$p'' = (p_x \times \cos(\alpha + \beta), p_x \times \sin(\alpha + \beta)) \quad (18)$$

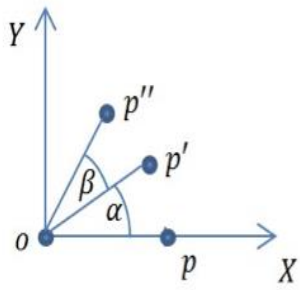


Figura 4: Rotação arbitrária

Através das regras da trigonometria, as componentes x e y do ponto p'' são dadas por:

$$p''_x = p'_x \times \cos(\beta) - p'_y \times \sin(\beta) \quad (19)$$

$$p''_y = p'_x \times sin(\beta) + p'_y \times cos(\beta) \quad (20)$$

Desta forma, o ponto p'' pode ser representado na forma matricial do seguinte modo [1]:

$$p'' = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p'_x \\ p'_y \\ 1 \end{bmatrix} \quad (21)$$

No que diz respeito a rotações 3D no eixo Z, estas são semelhantes a rotações 2D, uma vez que as componentes Z e a origem não serão afetadas pela rotação, bem como não influenciam os novos valores de x e y. Deste modo, a rotação no eixo Z é representada pela seguinte matriz [1].

$$R_{Z,\beta} = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 & 0 \\ \sin(\beta) & \cos(\beta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (22)$$

Pelo mesmo motivo se obtêm as matrizes para as rotações no eixo X e Y.

$$R_{X,\beta} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) & 0 \\ 0 & \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (23)$$

$$R_{Y,\beta} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (24)$$

ROTAÇÃO SOBRE EIXOS ARBITRÁRIOS

Considerando que, em vez de um eixo principal, se pretende executar uma rotação sobre um eixo arbitrário com direção definida por $\vec{v}$.

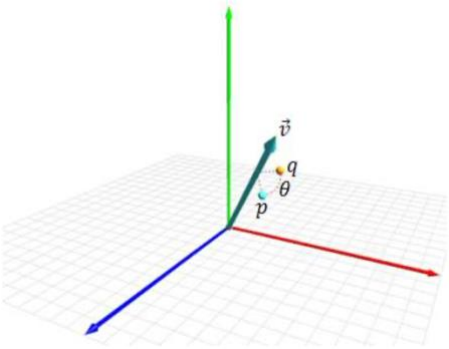


Figura 5: Rotação sobre eixos arbitrários

Analisando a Figura 5, verifica-se que o ponto q é obtido através da rotação do ponto p sobre o vetor $\vec{v}$ pelo ângulo θ. Deste modo, pretende-se uma matriz de rotação tal que [1]:

$$q = R_{v,\theta} p \quad (25)$$

Assim sendo, a obtenção da matriz da rotação do ponto p sobre o vetor $\vec{v}$, pelo ângulo θ, pode ser concretizada segundo:

1. Rodar o vetor $\vec{v}$ de modo a que este fique alinhado pelos eixos principais;
2. Rodar o ponto sobre o eixo selecionado, por θ;
3. Desfazer as rotações do passo 1.

Caso o vetor $\vec{v}$ não coincida ainda com um dos eixos principais, a primeira etapa requer pelo menos duas fases. Considere-se este caso, visível na Figura 6.

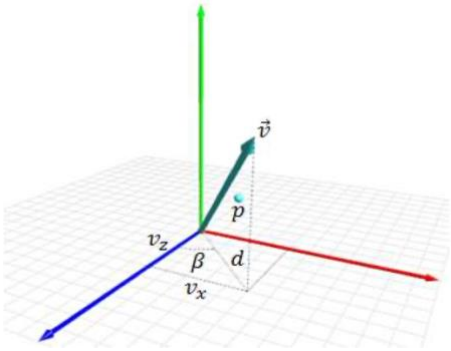


Figura 6: Rotação de v, para alinhar com o plano YZ

O primeiro passo passa pela rotação, de modo a que o vetor $\vec{v}$ fique no plano YZ ou XZ. Assumindo que se pretende colocar $\vec{v}$ no plano YZ, é necessário executar uma rotação em torno do eixo Y. Para determinar a matriz, é crucial analisar o ângulo β [1].

Considerando que $\vec{v} = (v_x, v_y, v_z, 0)$. A projeção deste vetor no plano XZ é $\vec{v}_{xz} = (v_x, 0, v_y, 0)$, pelo que a sua norma é dada por $d = \sqrt{{v_x}^2 + {v_z}^2}$. Observando a Figura 6, conclui-se que:

$$\cos(\beta) = \frac{v_z}{d}$$
$$\sin(\beta) = \frac{v_x}{d} \qquad (26)$$

No entanto, note-se que, com base da regra da mão direita, as rotações são realizadas no sentido contrarrelógio. A rotação em questão é aplicada no sentido dos ponteiros do relógio e, por este motivo, é necessário considerar a rotação do ângulo como sendo $-\beta$ [1].

$$\cos(-\beta) = \frac{v_z}{d}$$
$$\sin(-\beta) = -\frac{v_x}{d} \tag{27}$$

A matriz R_y e a sua inversa podem ser apresentadas do seguinte modo [1]:

$$R_y = \begin{bmatrix} \frac{v_z}{d} & 0 & -\frac{v_x}{d} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{v_x}{d} & 0 & \frac{v_z}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad {R_y}^{-1} = \begin{bmatrix} \frac{v_z}{d} & 0 & \frac{v_x}{d} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{v_x}{d} & 0 & \frac{v_z}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{28}$$

Após realizada a rotação, obtém-se a equação 29, representada graficamente na Figura 7:

$$\vec{v}' = R_y \vec{v} \tag{29}$$

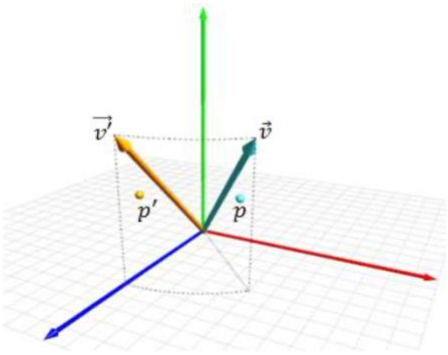


Figura 7: Vetor v alinhado com o plano YZ

De seguida, é necessária a rotação sobre o eixo X, de forma a alinhar o eixo de rotação com o eixo Z. Como o ângulo de rotação é α, tem-se $\cos(\alpha) = \frac{d}{v}$ e $\sin(\alpha) = \frac{v_y}{v}$.

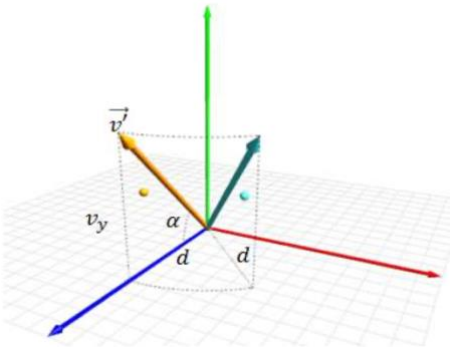


Figura 8: Rotação para alinhar v com o eixo Z

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & d & -v_y & 0 \\ 0 & v_y & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad {R_x}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & d & v_y & 0 \\ 0 & -v_y & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{30}$$

Após realizadas as transformações da equação 30, obtém-se $\vec{v}''$ alinhado com o eixo Z, e p''. Deste modo, dá-se por terminada a fase 1 do processo [1].

$$\vec{v}'' = R_x \vec{v}' = R_x R_y \vec{v} \tag{31}$$

$$p'' = R_x p = R_x R_y p \tag{32}$$

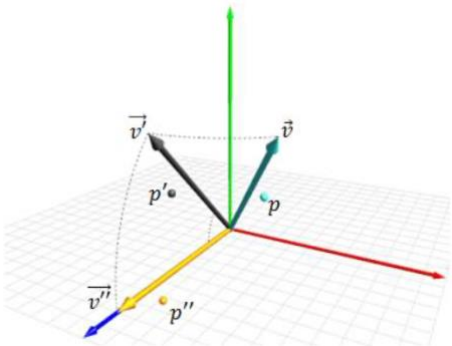


Figura 9: v alinhado com o eixo Z

Concluída a primeira fase, é precisa a rotação em torno do eixo Z por um ângulo θ, deduzida através da equação 22, visível na Figura 10 [1].

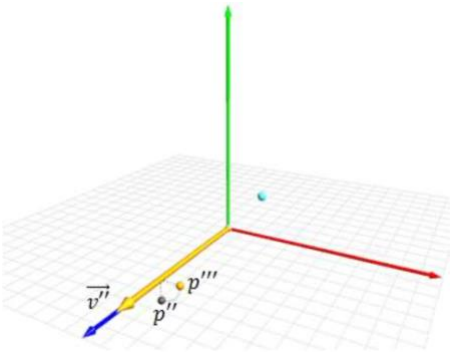


Figura 10: Final do passo 2

O ponto p''' pode ser obtido através da equação 33:

$$p''' = R_{Z,\beta}p'' = \begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 & 0 \\ \sin(\beta) & \cos(\beta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} p'' \qquad (33)$$

O último passo consiste em aplicar, pela ordem inversa, as matrizes inversas calculadas nas equações 28 e 30, isto é, ${R_x}^{-1}$ e ${R_y}^{-1}$.

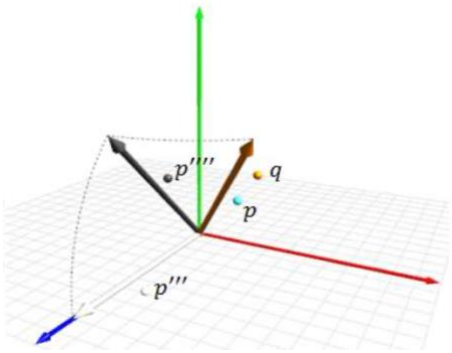


Figura 11: Final do passo 3

Na Figura 11, o ponto p'''' é obtido por:

$$p'''' = {R_x}^{-1}p''' \qquad (34)$$

Por outro lado, o ponto q é obtido através da equação 35:

$$q = {R_y}^{-1} p'''' \tag{35}$$

Aglomerando todos os passos, a matriz final para a rotação de ângulo θ sobre o vetor $\vec{v}$ é dada por:

$$R_{v,\theta} = {R_y}^{-1} \times {R_x}^{-1} \times R_{z,\theta} \times R_x \times R_y \tag{36}$$

Em termos práticos, são aplicados ambos os tipos de rotação. Uma vez que o Sol se encontra no centro dos eixos, os planetas vão realizar uma rotação sobre os eixos principais, permitindo que orbitem em torno do Sol. Por sua vez, os satélites naturais dos vários planetas vão realizar uma rotação sobre eixos arbitrários, dado que as suas órbitras são em torno do respetivo planeta.

A aplicação prática destes conceitos foi aplicada através da função *glRotatef* da biblioteca *Glut* do seguinte modo.

```
glRotatef(nodo.getRotateAngulo(),nodo.getRotateX(),nodo.getRotateY(),nodo.getRotateZ());
```

Sirva de exemplo a seguinte condição de rotação:

```
<rotate angle="60" axisX="1" axisY="1" axisZ="0"/>
```

No exemplo apresentado, o objeto vai realizar uma rotação de 60° sobre os eixos X e Y.

GENERATOR

Para a realização da segunda fase do projeto, não foram necessárias quaisquer alterações no gerador elaborado na primeira fase do mesmo. Por este motivo, não será apresentada uma descrição sobre este.

ENGINE

ESTRUTURA DE DADOS

Na classe XMLData encontram-se as estruturas de dados necessárias para a realização deste projeto. Desde modo, o vetor modelos vai armazenar todos os ficheiros a ler num determinado "nodo", de modo a permitir o desenho dos mesmos.

```
std::vector<std::string> modelos
```

Por sua vez, o vetor pontos é responsável por armazenar todos os pontos presentes nesses ficheiros, agrupando-os em conjuntos de 9 valores, correspondentes a um triângulo.

```
std::vector<triangulo> pontos
```

Uma vez que o modelo XML necessita de suportar a ideia de "filhos", no exemplo prático, a existência de luas e anéis, e sabendo que estes têm que sofrer as mesmas transformações geométricas que os respetivos pais, foi criado o vetor filhos, que recorre à recursividade da própria estrutura de dados.

```
std::vector<xmlData> filhos
```

Por fim, a estrutura de dados será guardada num outro vetor, que é retornado e passado às várias funções que precisam da informação, sendo o seu tamanho igual ao número total de grupos "pai" existentes..

```
std::vector<xmlData> ficheiros
```

```
<scene>
  <group> <!-- SOL -->
    <scale X="20" Y="20" Z="20"/>
    <models>
      <model file="planeta.3d"/>
    </models>
  </group>
  <group> <!-- MERCÚRIO -->
    <translate X="26"/>
    <scale X="0.28" Y="0.28" Z="0.28"/>
    <models>
      <model file="planeta.3d"/>
    </models>
  </group>
  <group> <!-- VÉNUS -->
    <translate X="-30.5"/>
    <scale X="0.7" Y="0.7" Z="0.7"/>
    <models>
      <model file="planeta.3d"/>
    </models>
  </group>
  <group> <!-- TERRA -->
    <translate Z="35"/>
    <scale X="0.73" Y="0.73" Z="0.73"/>
    <models>
      <model file="planeta.3d"/>
    </models>
    <group> <!-- LUA -->
      <translate X="1.5"/>
      <scale X="0.16" Y="0.16" Z="0.16"/>
      <models>
        <model file="planeta.3d"/>
      </models>
    </group>
  </group>
```

Figura 12: Exemplo do ficheiro XML

PARSER

Após criadas as estruturas de dados que irão dar suporte à aplicação, segue-se para o ficheiro responsável pelo *parsing*. Assim sendo, tal como na primeira fase do projeto recorremos à API do *tinyxml2* para realizar esta tarefa. Nesta fase, foi necessária a alteração ao modo como o *parsing* foi efetuado, uma vez que, para além da construção das primitivas gráficas, existe também informação relativa às transformações geométricas.

```
std::vector<xmlData> initParsing(std::string ficheiro)
```

Esta função é usada para percorrer os nodos "pai". Vai ser auxiliada com as funções dadosXML e getModelosFilhos, que permitem obter os dados inclusive sobre as transformações geométricas a realizar, e percorrer todos os nodos “filhos” de modo recursivo, respetivamente.

LOAD

De seguida, e dada como concluída a análise do ficheiro *XML*, é imprescindível o preenchimento de parte da estrutura de dados.

```
std::vector<triangulo> pontos
```

De modo a preencher este vetor, é necessário efetuar o carregamento dos ficheiros que se encontram dentro do vetor

```
std::vector<std::string> modelos
```

Assim, são utilizadas as funções initLoad e loadFilhos. Estas funções seguem o mesmo conceito previamente apresentado nas funções initParsing e getModelosFilhos. No entanto, estas funções distinguem-se pelo facto de iterarem a estrutura de dados, ao invés de iterar o ficheiro *XML* fornecido.

Por sua vez, a função loadFicheiros é responsável por recorrer ao vetor

```
std::vector<std::string> ficheiros
```

na tentativa de abrir o ficheiro com nome correspondente à *string* e ler a informação que nela se encontra, isto é, lê nove floats correspondentes a um triângulo, que serão posteriormente guardados no vetor

```
std::vector<triangulo> pontos;
```

REPRESENTAÇÃO DO SISTEMA SOLAR

Por fim, é crucial a possibilidade de representação do Sistema Solar, pelo que essa responsabilidade cai sobre a função startDrawing.

Contrariamente ao realizado na fase anterior, é indispensável ter em atenção a possibilidade de existência de transformações geométricas na cena a apresentar. Por este motivo, a função desenharFiguras irá analisar o nodo armazenado em globalnfo, realizando as transformações necessárias, na sua respetiva ordem. De notar que, uma vez que serão efetuadas alterações aos eixos principais, através das transformações geométricas, é importante recorrer às funções glPushMatrix() e glPopMatrix(), permitindo guardar o estado inicial da matriz e, posteriormente, substituir a matriz atual novamente pelo estado inicial da mesma.

SISTEMA SOLAR

Para a construção do Sistema Solar, foi tida em conta a constituição do mesmo. Deste modo, foram acrescentados os oito planetas principais e Plutão, bem como alguns dos satélites naturais de cada planeta. A atribuição dos anéis aos planetas Saturno e Úrano, bem como a construção da cintura de asteroides serão realizados numa fase posterior.

Assim, teve-se como base do Sistema Solar o posicionamento e tamanho do Sol, que se encontra na posição (0,0,0) e com raio = 20 unidades. As dimensões e distâncias dos planetas foram definidas com base no Sol. Deste modo, o cálculo do raio de cada planeta teve em consideração a razão entre a dimensão deste relativamente ao sol, com um raio de 20 unidades, tendo sido adicionado um fator multiplicativo de 4 unidades, uma vez que, caso contrário, as dimensões de cada planeta seriam muito pequenas. Por sua vez, a distância de cada planeta ao sol regeu-se à escala de 10 milhões de quilómetros por unidade.

Deste modo, tem-se:

$$raioP' = \frac{raioP * 20}{696340} * 4$$

$$dP' = \frac{d}{10000000}$$

Planeta	Raio	Distância ao Sol
Mercúrio	0.28	6
Vénus	0.70	10.5
Terra	0.73	15
Marte	0.39	22.5
Júpiter	8.03	75
Saturno	6.69	142.5
Úrano	2.91	285
Neptuno	2.82	450
Plutão	0.14	581.68

No que diz respeito aos satélites naturais, numa primeira fase, os seus raios e distância ao centro do respetivo planeta são dados por:

$$raioS = 0.1 * raioP$$

$$dS = 1.5 * raioP$$

Isto deve-se ao facto de que alguns satélites naturais têm poucos quilómetros de diâmetro, pelo que a atribuição de uma escala real modificada iria destabilizar a tentativa de manter o Sistema Solar com dimensões e distâncias o mais corretas possíveis.

Satélite Natural	Raio	Distância ao Planeta
Lua	0.07	1.1
Phobos	0.04	0.59
Deimos	0.04	0.59
Io	0.80	12.05
Europa	0.80	12.05
Calisto	0.80	12.05
Genimedes	0.80	12.05
Mimes	0.67	10.04
Encélado	0.67	10.04
Tétis	0.67	10.04
Ariel	0.29	4.37
Puck	0.29	4.37
Tritão	0.28	4.23
Larissa	0.28	4.23

MOVIMENTO DA CÂMARA

De modo a facilitar a verificação da concretização das transformações geométricas, foram alterados os parâmetros relativos à posição da câmara.

Para este efeito, e uma vez que é necessária a utilização de coordenadas esféricas, foi criada a função spherical2cartesian.

```
void spherical2cartesian() {
    camX = radius * sin(alfa) * cos(beta);
    camZ = radius * cos(alfa) * cos(beta);
    camY = radius * sin(beta);
}
```

Assim, tem-se

```
gluLookAt(camX + refX , camY + refY , camZ + refZ,
          refX , refY , refZ,
          0.0f , 1.0f , 0.0f);
```

Desta forma, é possível, através de teclas especiais, alterar a posição da câmara, modificando os valores de alfa e de beta.

```
case(GLUT_KEY_RIGHT):
    alfa -= 0.01; break;
case(GLUT_KEY_LEFT):
    alfa += 0.01; break;
case(GLUT_KEY_UP):
    beta += 0.01;
    if (beta > 1.5f)
        beta = 1.5f;
    break;
case(GLUT_KEY_DOWN):
    beta -= 0.01;
    if (beta < -1.5f)
        beta = -1.5f;
    break;

case GLUT_KEY_PAGE_DOWN: radius -= 5.0f;
    if (radius < 1.0f)
        radius = 1.0f;
    break;

case GLUT_KEY_PAGE_UP: radius += 5.0f; break;
```

Por outro lado, através das teclas regulares, é possível alterar o ponto para onde a câmara está a olhar, modificando os valores de refX, refY e refZ.

```
case 'w': {
    refZ -= 5 * cos(alfa);
    refX -= 5 * sin(alfa);
} break;
case 's': {
    refZ += 5 * cos(alfa);
    refX += 5 * sin(alfa);
} break;

case 'd': {
    refX += 5 * cos(alfa);
    refZ += 5 * (-sin(alfa));
} break;
case 'a': {
    refX -= 5 * cos(alfa);
    refZ -= 5 * (-sin(alfa));
} break;
case 'q': {
    refY += 3;
} break;
case 'e': {
    refY -= 3;
} break;
case '-': {
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
} break;
case '.': {
    glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
} break;
case ',': {
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
} break;
```

CONCLUSÕES

A realização da segunda fase do projeto do Sistema Solar permitiu que o grupo solidificasse conhecimento sobre transformações geométricas. A inclusão de cada transformação geométrica possibilitou o melhor entendimento sobre a relevância da ordem pela qual estas são realizadas, assim como a forma como estas devem ser implementadas.

A implementação do Sistema Solar estático, bem como da hierarquia na representação de cenas possibilitou o que o grupo considera ser o sucesso da segunda fase deste projeto.

ANEXOS

Apresentam-se em seguida a representação gráfica do Sistema Solar estático, com a ausência dos anéis e cintura de asteroides, a concluir na fase seguinte.

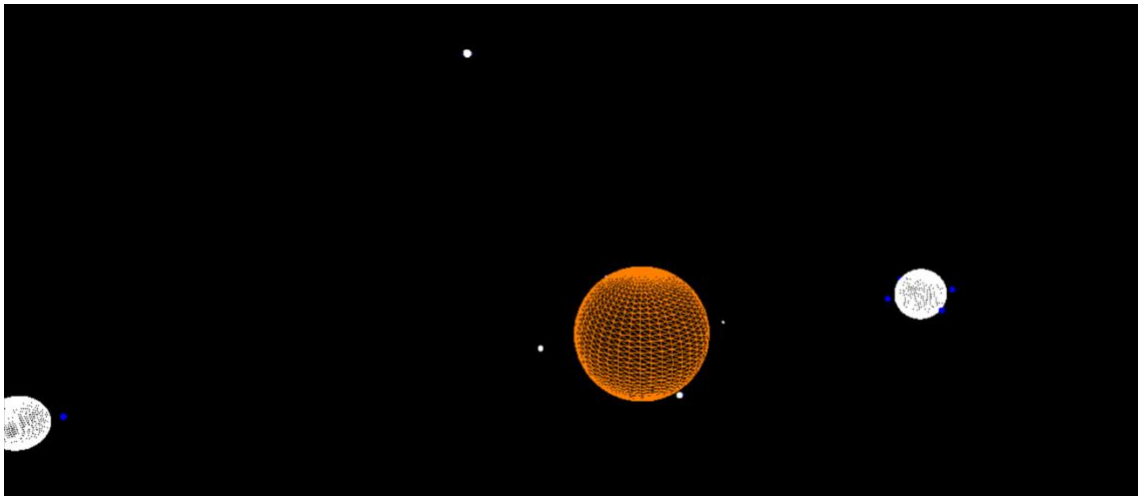


Figura 15: Sistema Solar Estático 1

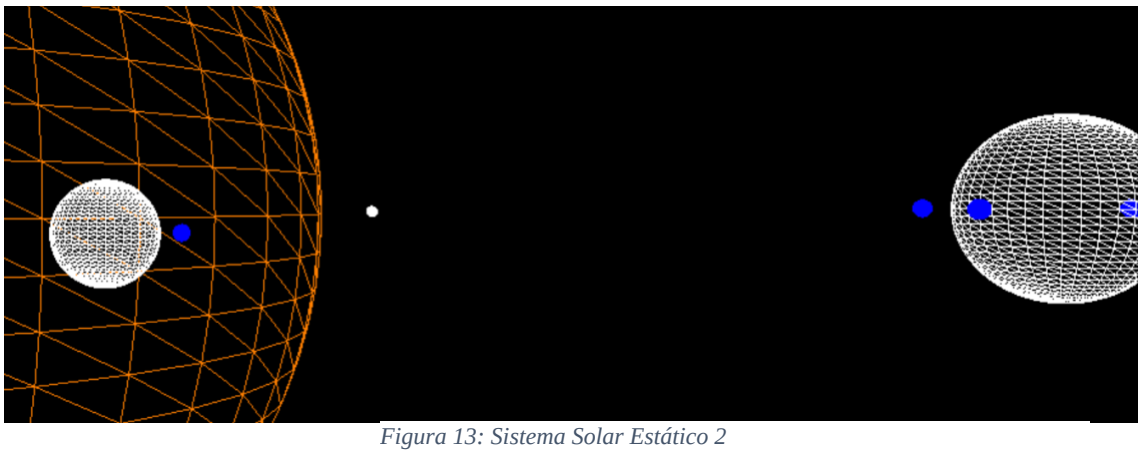


Figura 13: Sistema Solar Estático 2

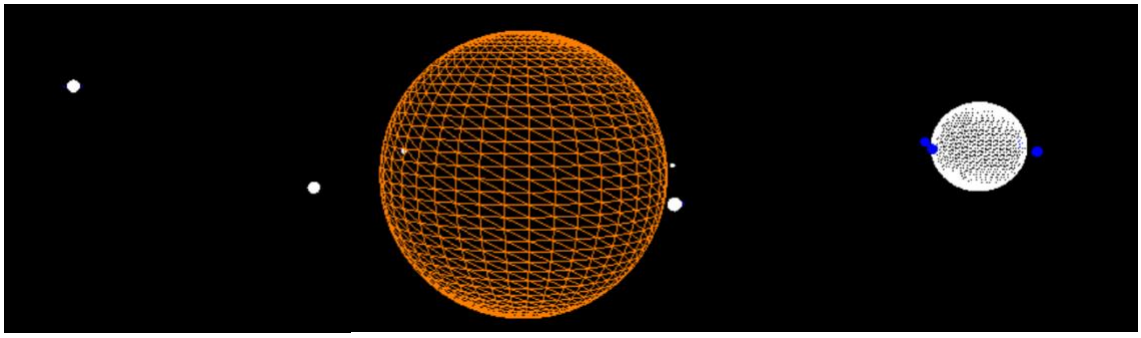


Figura 14: Sistema Solar Estático 3