Computação Gráfica: Fase Nº2

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Grupo 14

Flávio Martins
A65277

Gonçalo Costeira
A79799

José Ramos
A73855

Rafael Silva
A74264

Contents

List of Figures

1 Introdução

Na segunda fase do trabalho foi proposto ao grupo a continuação do trabalho realizado na primeira fase, agora utilizando algumas transformações geométricas.
Nesta fase os ficheiros XML irão estar sujeitos a um modelo hierárquico, em árvore, na qual cada nodo contém um conjunto de transformações, modelos e nodos filhos. As transformações num nodo afetam os nodos filhos do mesmo, demonstrando a hierárquia das transformações na árvore.
A demonstração desta fase será feita através dum modelo estático do sistema solar, incluindo o sol, planetas e luas, definido de modo hierárquico.

2 Gerador

Nesta fase o gerador é praticamente idendico ao da fase anterior. A unica primitiva utilizada da primeira fase é a esfera sendo que foi criado adicionalmente o anel que iremos utilizar tambem nesta fase.

2.1 Ring

Nesta fase já incluimos os aneis dos planetas, para tal foi necessário criar uma função capaz de os gerar.

Para desenhar o anel é necessário limitar o raio interior e exterior, assim como o número de slices a utilizar.

```
float slice = (360/slices)*(M_PI/180);
float doisPi = 2*M_PI;

if (fd!=NULL){
    for (float i = 0; i < doisPi; i += slice){
        fprintf(fd , "%f %f %f \n" ,sin(i)*in_r, 0.0, cos(i)*in_r);
        fprintf(fd , "%f %f %f \n" ,sin(i)*out_r, 0.0, cos(i)*out_r);
        fprintf(fd , "%f %f %f \n" ,sin(i+slice)*in_r, 0.0, cos(i+slice)*in_r);
        fprintf(fd , "%f %f %f \n" ,sin(i)*out_r, 0.0, cos(i)*out_r);
        fprintf(fd , "%f %f %f \n" ,sin(i+slice)*out_r, 0.0, cos(i+slice)*out_r);
        fprintf(fd , "%f %f %f \n" ,sin(i+slice)*in_r, 0.0, cos(i+slice)*in_r);
        fprintf(fd , "%f %f %f \n" ,sin(i)*out_r, 0.0, cos(i)*out_r);
        fprintf(fd , "%f %f %f \n" ,sin(i)*in_r, 0.0, cos(i)*in_r);
        fprintf(fd , "%f %f %f \n" ,sin(i+slice)*in_r, 0.0, cos(i+slice)*in_r);
        fprintf(fd , "%f %f %f \n" ,sin(i+slice)*out_r, 0.0, cos(i+slice)*out_r);
        fprintf(fd , "%f %f %f \n" ,sin(i)*out_r, 0.0, cos(i)*out_r);
        fprintf(fd , "%f %f %f \n" ,sin(i+slice)*in_r, 0.0, cos(i+slice)*in_r);
    }
}else{
    printf("Error creating 3D file...\n");
}
```

Figure 1: Função que cria um anel

3 Engine

3.1 Ficheiro XML de pontos

3.1.1 Elementos XML

Nesta secção iremos apresentar os diferentes elementos que iremos usar na nossa contrução do ficheiro XML com os diferentes pontos, formando for final uma cena. Para cada um dos elementos XML iremos usar diferentes nomes e atribuições aos mesmos. Estes elemtentos irão ser lidos pelo *Engine*, para o desenho da figura. Estes elemetos irão ser denotados de Nodos XML, pelo que estes termos serão usados ao longo do relatório, de seguida explicamos o conceito de cada nodo.

- <figure> ... </figure>
- <group> ... </group>
- <specs X=? angle=? axisX=? axisY=? axisW=? scale_x=? scale_y=? scale_z=? />
- <color R=? G=? B=? />
- <orbit radiusX=? radiusZ=? />
- <models> <model> file=? </model> ... </models>

3.1.1.1 Nodo *<figure>*

O elemento ***figure*** é o nodo pai de todos os outros, ou seja, de todo o documento. Todos os elementos criados no ficherio XML devem estar dentro destes parâmentros, começando por abrir a *tag* e no final fecha-la.

```
<figure>
    ...
    <!-- Groups XML -->
    ...
</figure>
```

Figure 2: Nodo *figure*

3.1.1.2 Nodo *<group>*

Este nodo consiste num nodo filho do apresentado em cima, e tem como função conter a informação todo precisa a construção de um dado planeta, bem como da sua órbita, ou seja, que um nodo ***group*** irá conter a descrição do que será preciso ser desenhado, tais como, as transfomaçoes ao objeto, a sua cor, se possuir ou nao orbita e que modelo irá usar nessas transformações.

```
<group>
   ...
   Specs, Color, Orbit Models
   ...
</group>
```

Figure 3: Nodo *group*

3.1.1.3 Nodo *<specs>*

Nodo filho de ***group*** que descreve as transformações que foram efetuadas ao um dado objeto, tais como, translações, rotações relativamente aos eixos e onde o objeto se encontra relativamente aos eixos.
Contem as seguintes especificações:

1 - Translações

- **X** - Componente x do vetor de translação. (Não é um valor obrigatório)
- **Y** - Componente y do vetor de translação. (Não é um valor obrigatório)
- **Z** - Componente z do vetor de translação. (Não é um valor obrigatório)

2 - Rotações

- **angle** - Descreve o angûlo de rotação em causa. (Não é um valor obrigatório)
- **axisX** - Componente x do vetor sobre o qual sera feita a rotação. (Não é um valor obrigatório)
- **axisY** - Componente y do vetor sobre o qual sera feita a rotação. (Não é um valor obrigatório)
- **axisZ** - Componente z do vetor sobre o qual sera feita a rotação. (Não é um valor obrigatório)

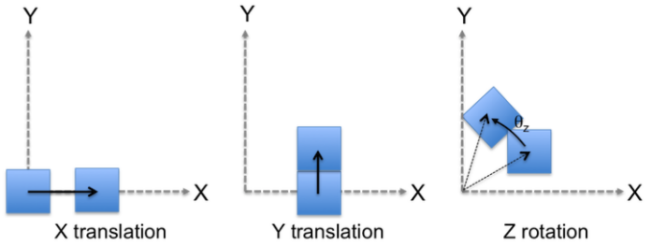

Figure 4: Diferença de transformações

3 - Escala

- **scale_x** - Indica a escala no eixo *oX*.
- **scale_y** - Indica a escala no eixo *oY*.
- **scale_z** - Indica a escala no eixo *oZ*.

```
<specs X=55 angle=177.36 axisZ=1 scale_x=0.13 scale_y=0.13 scale_z=0.13 />
```

Figure 5: Nodo *specs* (Exemplo)

3.1.1.4 Nodo *<color>*

Nodo responsável pela atribuição de uma determinada cor a um objeto, ele irá conter os seguintes elementos:

- **R** - Valor para a cor vermelha no sistema *RGB*, toma valor de 0 a 1.
- **G** - Valor para a cor verde no sistema *RGB*, toma valor de 0 a 1.
- **B** - Valor para a cor azul no sistema *RGB*, toma valor de 0 a 1.

```
<color R=1.0 G=1.0 B=0 />
```

Figure 6: Nodo *color* (Exemplo)

3.1.1.5 Nodo *<orbit>*

Este nodo é responsável pela criação de uma órbita definida nos seus parâmetros, que são os seguintes:

- **radiusX** - irá definir o raio da orbita perante o eixo *oX*.
- **radiusZ** - irá definir o raio da orbita perante o eixo *oZ*.

Esta informação é depois completada com a informação descrita em cima, ou seja, com nodos **specs** e/ou **color**.

```
<orbit radiusX=45 radiusZ=44.25 />
```

Figure 7: Nodo *orbit* (Exemplo)

3.1.1.6 Nodo *<models>*

Este nodo contém a figura sobre a qual as transformaçoes descritas acima irão fazer efeito. Terá o seguinte campo:

- *file* - Nome ou caminho para o ficheiro que contém a figura que pretendemos modificar.

```
<models>
    <model file="./SolarSystem/primitiveSphere.3d" />
</models>
```

Figure 8: Nodo *model* (Exemplo)

3.1.2 Exemplos XML

Neste secção iremos mostrar alguns exemplos de como estes nodos irão ser aplicados.

```
<!-- sol -->
  <group>
      <specs angle=7.25 axisZ=1 scale_x=3 scale_y=3 scale_z=3  />
      <color R=1.0 G=1.0 B=0 />
      <models>
          <model file="./SolarSystem/primitiveSphere.3d" />
      </models>
  </group>
```

Figure 9: Sol

```
<!-- orbit terra -->
<group>
    <specs angle=20 axisY=1 />

    <group>
        <specs X=0.999 />
        <color R=0.3 G=0.3 B=1 />
        <orbit radiusX=60 radiusZ=59.99 />

        <!-- terra -->
        <group>
            <specs X=60 angle=23.45 axisZ=1 scale_x=0.137 scale_y=0.137 scale_z=0.137 />
            <color R=0.3 G=0.3 B=1 />
            <models>
                <model file="./SolarSystem/primitiveSphere.3d" />
            </models>

                <!-- orbit lua -->
                <group>
                    <specs X=0.199 angle=5.14 axisZ=1 />
                    <color R=0.7 G=0.7 B=0.7 />
                    <orbit radiusX=10 radiusZ=10 />

                    <!-- lua -->
                    <group>
                        <specs X=10 angle=6.68 axisZ=1 scale_x=0.37 scale_y=0.37 scale_z=0.37 />
                        <color R=0.7 G=0.7 B=0.7 />
                        <models>
                            <model file="./SolarSystem/primitiveSphere.3d" />
                        </models>
                    </group>
                </group>
        </group>
    </group>
</group>
```

Figure 10: Terra e Lua e as suas órbitas

3.2 Estrutura de dados

Para realizar a nova fase do trabalho tivemos de reformular a nossa estrutura de dados, isto porque é necessário guardar um maior número de dados.

3.2.1 Point

O Point, ou Ponto, é constituido por três coordenadas (x, y, z) que definem um ponto no espaço 3D, criamos então a seguinte estrurura:

```
//Define um ponto no espaço 3d
class Point{
public:
    float coordx;
    float coordy;
    float coordz;
};
```

Figure 11: Point

3.2.2 Color

Esta estrutura foi criada para guardar as cores utilizadas nos diferentes objetos. A classe é constituida por três variáveis r, g e b, assim temos toda a informação necessária para a representação RGB usada no OpenGL.

```
//Define a cor do objeto
class Color{
public:
    float r,g,b;
};
```

Figure 12: Color

3.2.3 Orbit

Nesta fase do projeto é necessário criar a orbita de cada planeta, como todos têm uma orbita diferente foi necessário criar uma estrutura para guardar as respetivas orbitas. Geométricamente uma órbita é semelhante a uma elípse, decidimos então criar uma classe constituida por duas variáveis radiusX e radiusZ que definem os raios da elípse.

```
//Define os raios da orbita
class Orbit{
public:
    float radiusX,radiusZ;
};
```

Figure 13: Orbit

3.3 Model

Para armazenar uma figura geométrica (conjunto de pontos (Point)) temos a estrutura Model. Nesta classe guardamos então o nome da figura (file) e o conjunto de pontos (points) para a sua construção.

```
//Define uma lista de pontos
class Model{
public:
    string file;
    list<Point> points;
};
```

Figure 14: Model

3.3.1 Specs

No sistema solar temos três tipos de transformações (translação, rotação e escala), para isso criamos a estrura Specs que através das coordenadas x, y e z conseguimos reproduzir a translação. Para a rotação usamos o ângulo (angle) e a direção (eixos x, y e z). Por fim para recriar a escala usamos x, y e z para alterar as dimensões do objeto em questão.

```
//Transformações necessárias para o Sistema Solar: Translação, Rotação e Escala
class Specs{
public:
    float coordx, coordy, coordz;
    float angle, axisx, axisy, axisz;
    float scale_x, scale_y, scale_z;
};
```

Figure 15: Specs

3.3.2 Group

Group é a estrutura que reune todas as anteriores, além disso tem também a variável type que indica se é um planeta ou uma orbita.

```
//Reunião de todas as estruturas anteriores
class Group{
public:
    string type; // planeta ou orbita
    Specs specs;
    Color color;
    Orbit orbit;
    list<Model> models;
    list<Group> groups;
};
```

Figure 16: Group

3.3.3 Figure

Na estrutura Figure guardamos toda a informação que nos permite criar o sistema solar. Na variável groups temos todos os componentes do sistema solar e através da função load carregamos todos esses componentes.

```
//Reunião dos Grupos
class Figure{
public:
    list<Group> groups;
    void load(const char* pFilename);
};
```

Figure 17: Figure

3.4 Descrição do ciclo *Renderização*

Após ser feita a leitura do XML e armazenada a informação nas estruturas de dados, que permitirão desenhar o sistema solar, é feita, na função *renderScene()*, a travessia dos grupos onde estes elementos estão guardados.

Uma vez que os grupos estão guardados na variável global figura, o processo de desenho desses grupos é iniciado colocando um iterador nesta que vai processor todos os elementos recorrendo a uma função *renderGroup()*.

```
list<Group>::iterator itg = figure.groups.begin();
renderGroup(itg);
```

Esta função é responsável por iterar todos os grupos armazenados em *figure* e desenhar renderiza-los para apresenta-los no sistema que pretendemos criar.

Em primeiro lugar realizamos uma travessia em profundidade *(depth first search)* para permitir que todas as transformações que seja referenciadas a um grupo de hierarquia maior, se propaguem para todos os grupos filhos deste.

Assim, e de modo a permitir uma maior fieldade no sistema de coordenadas recorremos a métodos como pushMatrix() e popMatrix(), que nos permitem guardar um sistema de coordenadas numa stack de matrizes , permitindo-nos transformar este, trabalhar numa outra zona renderizando os grupos necessários nessas posições e depois ir buscar esse sistema de coordenadas guardado na stack de modo a termos uma origem a partir da qual trabalhar permitindo uma renderização das figuras muito mais eficaz e segura. Isto é, quando um grupo não tem mais filhos faz se popMatrix() que permite restaurar o sistema de coordenadas anterior.

Em regra estas funções encontram-se dentro da função *renderGroup()*, uma no inicio e outra no fim para que sempre que as transformações sejam realizadas e a função chamada garantimos o controlo da posição relativa das figuras.
O processo usado para o desenho de grupos segue, para cada um dos grupos, os seguintes passos:

- Em primeiro lugar, guarda-mos a matriz do sistema de coordenadas atual com *pushMatrix( )*.
- De seguida, percorremos o vector de instâncias de Transformação e para cada uma, de acordo com o seu tipo (seja ROTAÇÃO, TRANSLAÇÃO ou ESCALA), é aplicada a respetiva transformação.
- Após terem sido aplicadas as transformações, passamos a renderizar a *orbit* (apresentado na secção 3.2.3) que é desenhada segundo as definições fornecidas em *figure*.
- Assim, depois é percorrido o iterador de *Models* onde percorremos o vetos de *Point* e são desenhado os pontos segundo as definições de desenho.
- Com o objetivo de serem desenhados todos os grupos, são ainda percorridos todos os filhos do nodo atual e para cada filho é invocada recursivamente a função desenhaGrupo.
- Por fim, é invocamos a função popMatrix() para restaurar o sistema de coordenadas para o ponto antes das transformações do grupo terem sido aplicadas.

O pseudo-código da função renderGroup é o seguinte:

```
void renderGroup(tree<Grupo>::iterator itg)
glPushMatrix()
Aplica as Transformacoes
Desenha a Orbit
Desenha os Models de acordo com as definicoes de desenho
Para todos os filhos
Invoca a função renderGroup recursivamente
glPopMatrix()
```

3.5 Câmara

Um dos objetivos do projeto é permitir a visualização em primeira pessoa e permitir alterar tanto a posição do utilizador como a sua perspectiva.
A câmara utilizada é identica à câmara utilizada na primeira fase, uma vez que esta ja cumpria os requesitos visto que esta tem as funcionalidades de alterar a sua posição, alterar o ângulo de visão e de translação, isto é aumentar ou diminuir o valor tanto do eixo x como do eixo z em 1 unidade.

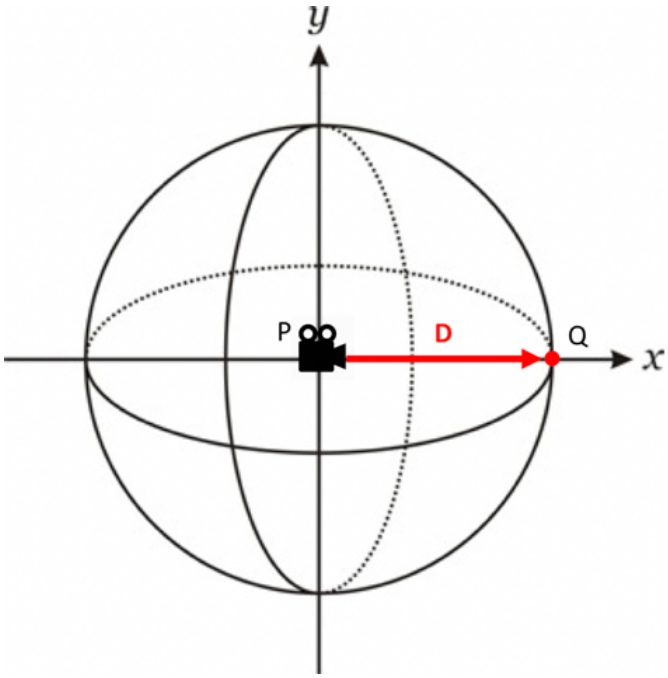

Figure 18: Esquema da posição da câmara (P) e do vetor direção D

3.6 Interacao com o utilizador

O utilizador pode interagir com o sistema através do teclado e do rato.

3.6.1 Teclado

- Tecla 'UP': Rotação de todos os componentes para cima, aumentando o ângulo beta em 0.1;
- Tecla 'DOWN': Rotação de todos os componentes para baixo, diminuindo o ângulo beta em 0.1;
- Tecla 'RIGHT': Rotação de todos os componentes para a direita, aumentando o ângulo alpha em 0.1;
- Tecla 'LEFT': Rotação de todos os componentes para a esquerda, diminuindo o ângulo alpha em 0.1;
- Tecla 'W': Aproxima a câmara;
- Tecla 'S': Afasta a câmara;
- Teclas 'G' e 'J': Permitem mover a câmara ao longo do eixo x;
- Teclas 'Y' e 'H': Permitem mover a câmara ao longo do eixo z;
- Tecla 'END': Termina o sistema.

3.6.2 Rato

Utilizando o botão esquerdo do rato, mantendo o mesmo premido, o utilizador pode movimentar a câmara na direção que pretender.
Com o botão direito o utilizador visualiza um menu com três opções, FILL, LINE e POINT, que alteram o aspeto das figuras.

3.7 Construção do Sistema Solar

O objetivo desta fase era representar o Sistema Solar, De seguida iremos demonstrar os resultados do nosso trabalho e algumas medidas que foram necessárias na sua realização.
Como o tamanho do Sol é extremamante maior do que o dos restantes planetas que precisavamos de representar este teve de ser ajustado, de forma a conseguirmos que o nosso sistema fosse mais equilibrado e de fácil visualização. A inclusão das orbitas, que não estão a uma escala correcta, proporcionou uma melhor visualização do sistema.
O sistema realizado constituido pelo Sol, os planetas Mercúrio, Vénus, Terra, Marte, Júpiter, Saturno, Urânio e Neptuno, e ainda a Lua, satélite da Terra, e Europa, Io, Ganimedes e Calisto, satélites de Júpiter. São ainda representadas as orbitas correspondentes a cada um deste objetos representados.

Nas imagens que são apresentadas de seguida será possivel visualizar o sistema final, assim como comprovar as capacidades de manipulção da câmara.

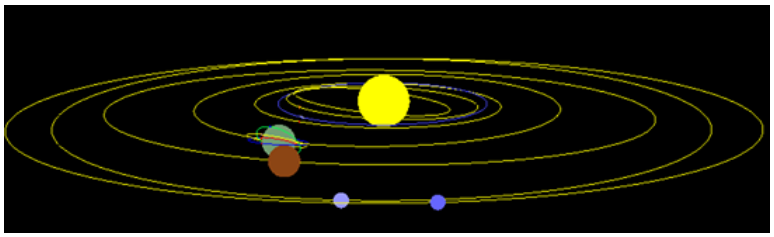

Figure 19: Sistema solar

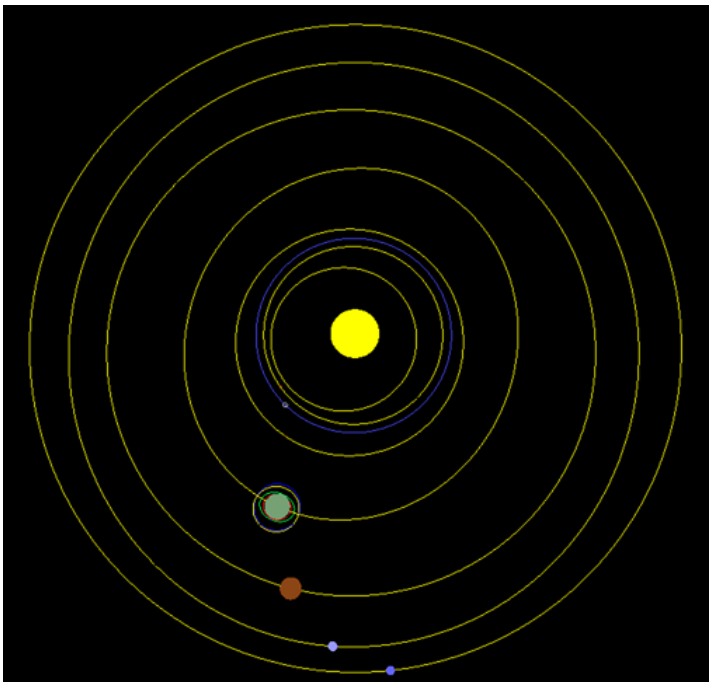

Figure 20: Sistema solar visto de outra perspectiva

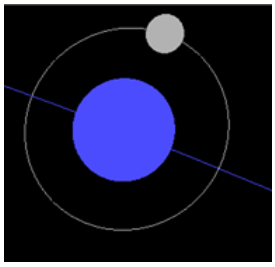

Figure 21: Terra e lua

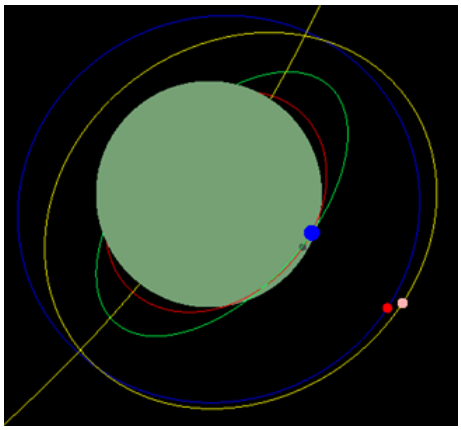

Figure 22: Júpiter e suas luas

4 Conclusão

No final desta fase o grupo sente que, tal como na primeira, a mesma foi bastante produtiva no que diz respeito a assimilar os conteúdos lecionados na Unidade Curricular de Computação Gráfica. Sentimos que conseguimos implementar uma representação do Sistema Solar correspondente ao que nos foi proposto.

A principal dificuldade que encontramos foi definir as escalas a utilizar, sendo este um sistema com distâncias tão grandes a escolha seria sempre algo importante. Como tal a nossa representação foi feita da forma que consideramos mais eficiente embora não tenha uma escala idêntica em todos os seus elementos.