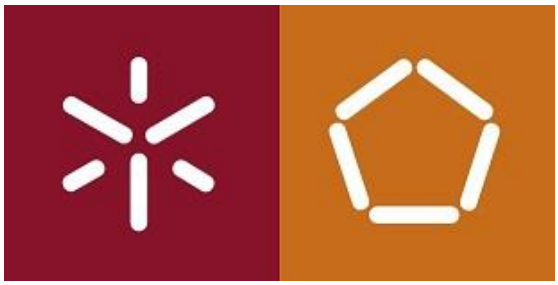

Universidade do Minho
Engenharia Informática

Trabalho Prático
Computação Gráfica

2º Semestre - 2021/2022

RELATÓRIO CG – Fase 2

Março 2022

Diogo da Costa e Silva Lima Rebelo (A93180)

Diogo Miguel da Silva Araújo (A93313)

Joel Costa Araújo (A76603)

Francisco Peixoto (A84668)

Resumo

O presente documento serve como base de acompanhamento à realização da segunda fase do projeto de computação gráfica, onde o objetivo principal é introduzir no projeto a noção de transformações geométricas. De forma a conseguir demonstrar o trabalho realizado, criou-se um modelo estático do sistema solar, incluindo o sol, os planetas e as suas luas, utilizando transformações geométricas.

Índice

Resumo 2
Índice 3
Índice de Ilustrações 3
Fase 2 – Hierarquia 4
Corpo Celeste 4
Sol 5
Corpos celestes com luas 5
Leitura do sistema solar 6
Primitivas utilizadas 7
Processo de renderização 8
Câmara 9
Conclusão 10
Anexos 11

Índice de Ilustrações

Figura 1 - Exemplo xml de um corpo celeste (Mercúrio) 4
Figura 2 – Exemplo xml do Sol 5
Figura 3 – Exemplo xml de um satélite natural 5
Figura 4 - Definição da classe CelestialBody 6
Figura 5 - Definição da classe Orbit 6
Figura 6 - Definição da classe Settings 6
Figura 7 - Definição da classe Primitive 7
Figura 8 - Definição da classe Figures 7
Figura 9 – Renderização do Sistema Solar 8
Figura 10 - Representação visual do SolarSystem.xml 11
Figura 11 - Planeta Terra e a lua 11

Fase 2 – Hierarquia

Como primeiro passo para construir um ficheiro xml, com uma hierarquia bem definida e que nos auxiliasse no processo de construção da representação gráfica com o sistema solar, tentamos definir uma estrutura para um qualquer corpo celeste, nomenclatura que utilizamos para nos referir ao sol, aos planetas e mesmo às suas luas.

Corpo Celeste

Um corpo celeste é constituído por dois componentes importantes, que se encontram definidos dentro de um grupo. O primeiro é referente às informações necessárias para configurar a órbita do corpo e, de seguida, é utilizado um novo grupo onde se encontram as definições relativas à configuração propriamente dita da figura em si, assim como o nome do ficheiro que contém os dados sobre a primitiva utilizada para a sua construção. Os valores escolhidos têm como referência a origem do sistema de eixos, ou seja, o sol.

```
<!-- Mercurio T = 0.25 Distancia(d)= 13   -->
<group>
    <!-- Orbita -->
    <rotate angle = 0 axisX= 0 axisY = 0 axisZ = 0 />
    <scale scaleX = 0  scaleY = 0 scaleZ = 0 />
    <orbit radX = 13 radZ = 13 /> <!-- Valores para obter a orbita do planeta-->

    <!-- Planeta -->
    <group>
        <translate X = 9.1923885  Y = 0 Z = 9.1923885 />
        <rotate angle = 5 axisX = 1 axisY = 0 axisZ = 0/>
        <scale scaleX = 0.25 scaleY = 0.25 scaleZ = 0.25 />
        <color R = 0.6667 G = 0.6627 B = 0.6784 />
        <!-- Silver Gray <color R = 192 G = 192 B = 192/> -->
        <models>
            <model file = "sphere.3d" />
        </models>
    </group>

</group>
```

Figura 1 - Exemplo xml de um corpo celeste (Mercúrio)

Sol

Apresentada a definição de um corpo celeste, e relembrando que o sol se encaixa nessa definição, temos de atentar que este é um dos dois casos especiais de um corpo celeste. Esta especificação deve-se ao facto de o sol ser o ponto central do nosso sistema de eixos e, como tal, definimo-lo como não tendo informações sobre a sua órbita. Com isto, apenas as informações sobre a posição e configuração no espaço são especificadas, assim como o modelo utilizado na primitiva.

```
<!-- Sol (Origem) -->
<group>
    <translate X = 0 Y = 0 Z = 0 />
    <rotate angle = 0 axisX = 0 axisY = 0 axisZ = 0 />
    <scale scaleX = 7  scaleY = 7  scaleZ = 7 />
    <color R = 1 G = 0.549 B = 0/> <!-- Dark Orange -->
    <!-- <color R = 253 G = 184 B = 19 />  Sun / Yellow Orange -->

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

</group>
```

Figura 2 – Exemplo xml do Sol

Corpos celestes com luas

De modo a poder incluir satélites naturais neste projeto (e visto que alguns planetas têm cerca de 100), foi escolhido pelo grupo apresentar apenas as 10 maiores luas, havendo assim, no projeto, vários planetas com esta distribuição. Dentro do grupo do planeta, existem *n* subgrupos, em que *n* é o número de luas que giram em torno do mesmo, de modo a permitir o desenho dos satélites em relação ao próprio planeta ao invés do sol, ficando assim, deste modo, a órbita das luas centradas no próprio planeta.

```
<!-- Lua distancia da terra 0.03844 -->
<group>
    <rotate angle = 25 axisX= 1 axisY = 0 axisZ = 0 />
    <scale scaleX = 0  scaleY = 0 scaleZ = 0  />
    <orbit radX = 1.6 radZ = 1.6 />

    <group>
        <translate X = 1.6 Y = 0 Z = 0/>
        <rotate angle = 0 axisX = 0 axisY = 1 axisZ = 0/>
        <scale scaleX =  0.2 scaleY =  0.2 scaleZ =  0.2 />
        <color R = 0.502 G = 0.502 B = 0.502 />
        <!-- Gray <color R = 255 G = 0 B = 0/> -->
        <models>
            <model file = "sphere.3d" />
        </models>
    </group>
</group>
```

Figura 3 – Exemplo xml de um satélite natural

Leitura do sistema solar

De forma a realizar a leitura do ficheiro *xml* que constitui o sistema solar, SolarSystem.xml, demos uso à biblioteca externa denominada de "tinyxml", a qual preparamos para conseguir efetuar a leitura da hierarquia que definimos.

Contudo, foi preciso preparar uma estrutura de dados que conseguisse persistir os dados retirados, para que uma consequente e contínua abertura do ficheiro em questão não persistisse com a execução do programa. Para tal, definimos uma classe, *CelestialBody*, cuja informação se divide em três campos: um objeto *Orbit*, outro *Settings* e, por fim, uma lista de *CelestialBodies*, referentes à possibilidade de um planeta ter um determinado número de luas associadas.

```
class CelestialBody{

    private:

        Orbit bodyOrbit;
        Settings bodySettings;
        vector<CelestialBody> moons;
```

Figura 4 - Definição da classe CelestialBody

Ao atentar a configuração xml de cada corpo celeste, tal como anteriormente explicado, encontram-se as definições da órbita e do corpo celeste em si, sendo esses os atributos utilizados para constituir, respetivamente, um objeto *Orbit* e *Settings.*

```
class Orbit {

    private:
        float rotateAngle, rotateX, rotateY, rotateZ;
        float scaleX, scaleY, scaleZ;
        float radX, radZ;
```

Figura 5 - Definição da classe Orbit

```
class Settings{

    private:
        float translX, translY, translZ;
        float rotateAngle, rotateX, rotateY, rotateZ;
        float scaleX, scaleY, scaleZ;
        float r,g,b;
        string primitiveModel;
```

Figura 6 - Definição da classe Settings

Primitivas utilizadas

Como seria de esperar, a única primitiva que utilizamos para configurar o sistema solar é a esfera.

Contudo, e atentando na possibilidade de necessitar de mais primitivas, decidimos criar uma outra estrutura auxiliar que persistisse as informações referentes às primitivas, de maneira a não precisar de consultar o ficheiro .3d associado, de cada vez que fosse necessário desenhá-la.

Assim, apresentamos a definição de uma classe *Primitive*, cujo conteúdo assenta no *path* do ficheiro associado à primitiva (.3d), o nome da primitiva (atentando nos nomes fixos “plane”, “box”, “cone” e “sphere”), para mais facilmente a identificar, um *vector* com os pontos todos que a definem e, por último, valores referentes à cor com a qual se pretende desenhar a primitiva.

Deste modo, ao ler um ficheiro .3d, conseguimos criar um objeto com todas as informações necessárias sobre a primitiva. Neste ponto, com uma bem definida, podemos utilizar os seus métodos para as desenhar, confirmando sempre que se está a utilizar a correta primitiva para o desenho, isto é, não permitir que se tente utilizar um *drawSphere* numa primitiva pré-definida como um plano. Tal processo é possível porque, relembrando uma decisão efetuada na primeira fase do projeto, atribuímos um valor como identificador a cada uma das primitivas, o qual vai explícito no ficheiro 3d associado.

```
class Primitive {

    private:
        string filename;
        string primitiveName;
        vector<Point_3D> points;
        float r,g,b;
```

Figura 7 - Definição da classe Primitive

Complementarmente a esta temos a classe *Figures*, que se resume a um *vector* com várias primitivas, ao qual se podem adicionar, remover, ou recolher qualquer uma das primitivas a ela associada.

```
class Figures {

    private:
        vector<Primitive> primitives;
```

Figura 8 - Definição da classe Figures

Processo de renderização

Chegados a um ponto onde temos a informação toda pronta, resta-nos transfigurá-la para a sua representação gráfica.

Para tal, e assumindo que temos uma lista de corpos celestes já preenchidos, passamos à parte de os desenhar, iterando-os um a um.

O processo resume-se a extrair, numa primeira fase, as informações relativas à configuração da órbita e, de seguida, do objeto (posição, ângulo, escalas, etc.), aplicando-as com a seguinte sequência de passos:

- para a órbita
  - plPushMatrix();
  - glRotate();
  - drawOrbit();
  - glPopMatrix():
- para o corpo celeste
  - glPushMatrix();
  - glTranslatef():
  - glRotatef();
  - glScalef();
  - drawSphere();

Contudo, um corpo celeste pode conter uma lista de luas e, para cobrir esse caso, se essa lista tiver valores, o que fazemos é chamar recursivamente a mesma função para qualquer lua, apenas aplicando o **glPopMatrix()** no final de cada execução.

```
glPushMatrix();
glRotatef(rOrbit, orbX, orbY, orbZ);
drawOrbit(planetOrbit.getRadX(), planetOrbit.getRadZ(), colorRGB);
glPopMatrix();

glPushMatrix();
glTranslatef(posX, posY, posZ);
glRotatef(rAngle, rotateX, rotateY, rotateZ);
glScalef(scaleX, scaleY, scaleZ);

currentPrimitive.drawSphere();
vector<CelestialBody> moons = cb.getMoons();

if (moons.size() != 0) {

    for (CelestialBody moon : moons) {

        drawCelestialBody(moon);

    }
}
glPopMatrix();
```

Figura 9 – Renderização do Sistema Solar

Câmara

De forma a desenvolver uma versão completa representativa da câmara para utilizar no *engine*, abordamos não só a vertente em primeira pessoa, mas também a vertente em terceira pessoa. Ambas as versões modificam o comportamento e as definições de como a câmara se comporta, e pode-se alternar usando a tecla 'v'.

Os comandos de controlo da câmara estão definidos pelo método *keyFunc* e funcionam da seguinte forma:

- , - modifica a formação dos objetos para visão cheia
- . - modifica a formação dos objetos para visão por arestas

Existem comandos para o controlo da câmara em primeira pessoa:

- l - na vista de primeira pessoa aponta para a direita
- h - na vista de primeira pessoa aponta para a esquerda
- k - na vista de primeira pessoa aponta para cima
- j - na vista de primeira pessoa aponta para baixo

Existem ainda comandos gerais que movimentam a câmara em ambos os modos:

- w - movimenta a câmara para a frente
- s - movimenta a câmara para trás
- q - movimenta a câmara para cima
- z - movimenta a câmara para baixo
- a - movimenta a câmara para a esquerda
- d - movimenta a câmara para a direita
- v - muda o modo da câmara para o modo seguinte

A câmara funciona ainda com controlo pelo rato, onde podemos modificar o angulo de visão da câmara arrastando o rato.

Conclusão

Dada a fase como terminada, consideramos ter obtido uma representação fiel e esperada do sistema solar, embora ainda fixa. As nossas dificuldades passaram pela escolha da ordem pela qual aplicávamos as transformações necessárias à configuração dos planetas e em definir os valores a ser usados no xml, visto que tentamos usar escalas reais, as quais tivemos de ir alterando para facilitar a visualização.

Como aspetos a melhorar, pretendemos tornar a câmara mais fácil de utilizar. Reparamos também que acabamos por focar demasiado a ler um xml com formato do sistema solar e não para ficheiros xml no geral, daí ser um aspeto que pretendemos melhorar na próxima fase.

Anexos

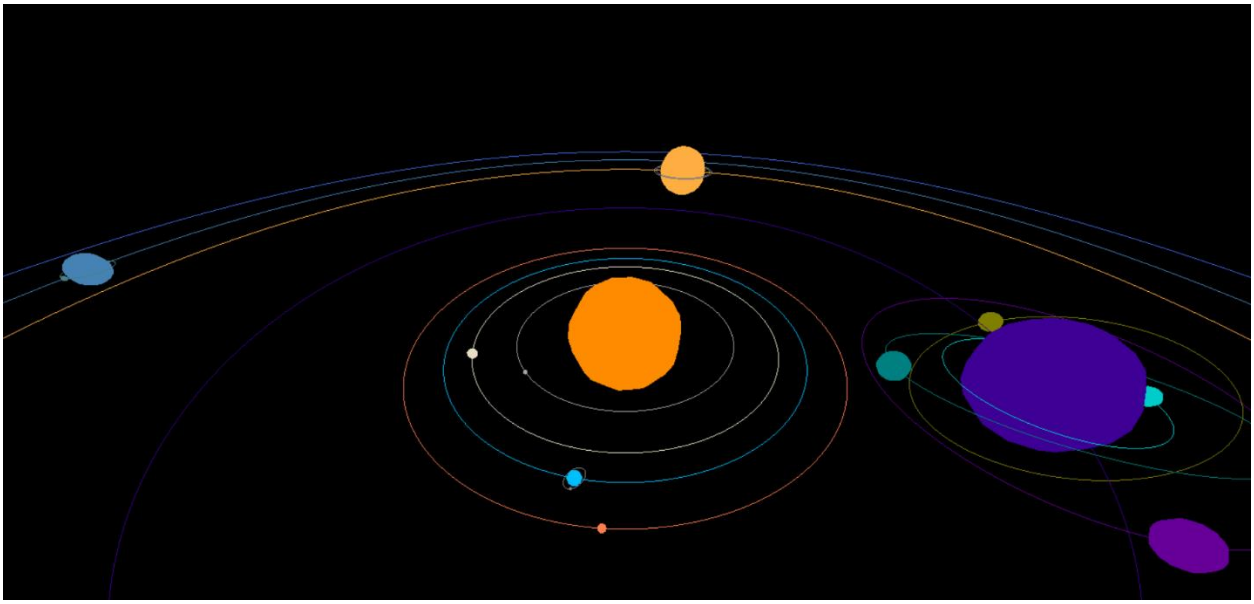

Figura 10 - Representação visual do SolarSystem.xml

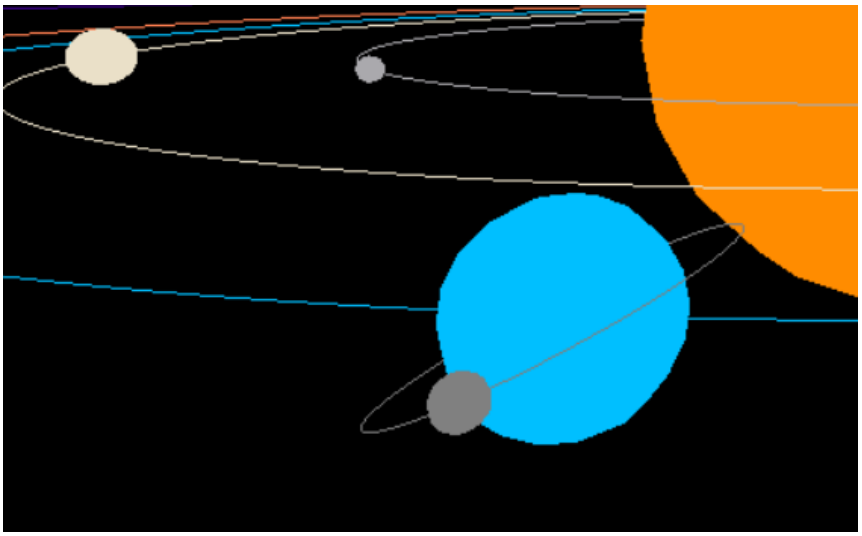

Figura 11 - Planeta Terra e a lua