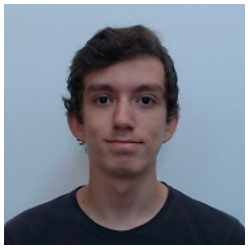
Trabalho Prático - fase 2
Primitivas gráficas

Catarina Cruz
A84011

Jorge Mota
A85272



Maria Silva
A83840

Mariana Marques
A85171

27 de Março de 2020

Universidade do Minho

Conteúdo

1 Resumo

Nesta segunda fase, segundo o que foi pedido pelo enunciado estruturamos uma cena (scene) para o renderizador interpretar os dados transcritos em xml e construir os devidos modelos e aplicar, de forma hierárquica, as devidas transformações a esses modelos.

Como tal, foram feitos alguns modelos, incluindo um protótipo do Sistema Solar para serem processados e interpretados.

Juntamente, algumas mudanças estruturais foram feitas ao projeto.

2 Cena

A cena é uma árvore de componentes que descrevem o necessário para compor a mesma. Esta pode ter 'n' grupos e 1 câmera.

Cada grupo pode ter 'n' modelos , transformações ou grupos tornando-se assim numa estrutura recursiva de grupos. Uma Transformação pode ser translação, rotação ou escala.

Para desenhar a cena há o método `scene.compose()` que compõe as transformações, modelos e câmera.

2.1 Grupos

Para desenhar o grupo há o método `group.draw()` que compõe as transformações, modelos e grupos que este contém.

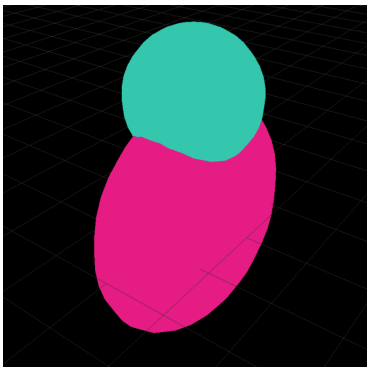
As transformações de um grupo aplicam-se aos modelos que nele estão contido e a todos os subgrupos deste, mas não prolongam para grupos vizinhos. Para demonstração tomemos como exemplo as seguintes cenas, com a respetiva representação :

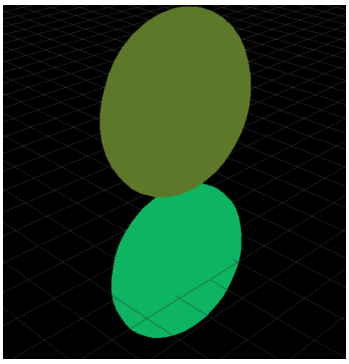
```
<scene>
   <group>
      <scale X="1" Y="2" Z="2"/>
      <model file="models/sphere.3d"/>
   </group>

    <group>
      <translate Y="2"/>
      <model file="models/sphere.3d"/>
   </group>
</scene>
```

```
<scene>
   <group>
      <scale X="1" Y="2" Z="2"/>
      <model file="models/sphere.3d"/>

      <group>
         <translate Y="2"/>
         <model file="models/sphere.3d"/>
      </group>
   </group>
</scene>
```





2.2 Transformações

A biblioteca gráfica OpenGL permite executar as tranformações translação, rotação e escala. Uma das principais vantagens é que estas transformações são acumulativas, ou seja, podem ser aplicadas umas sobre as outras com praticamente o mesmo custo de aplicação aos vértices no caso de haver muitas transformações envolvidas. Estas transformações geométrias são armazenadas (a partir de cálculos) internamente numa matriz e a cada nova transformação, esta matriz é alterada e usada para desenhar os objetos a partir daquele momento, até que seja novamente alterada.

Para poder-se guardar a informação das tranformações criamos as devidas classes 'transformation', 'translation' , 'rotation' e 'scale' que disponibilizam o método `apply()` para executar a respetiva instrução do OpenGL. A 'transformation' é construída a partir de um 'translation' , 'rotation' ou 'scale' , permitindo assim agregar todas as transformações da mesma forma numa coleção.

2.3 Modelos

Os modelos permanecem como na primeira fase, contêm os vértices de todos os triângulos necessários para desenhar esse modelo na cena.

Podem ser adicionados modelos a um grupo usando o método `add_model(model)`

3 Render Engine

3.1 Controles

O desenvolvimento dos controles da câmera ainda se encontram em progresso, e , para já, temos a rotação esférica da câmera em torno da origem e o zoom in e out da mesma para se poder ver o resto da cena.

Para a rotação esférica da câmera os controles são as SETAS do teclado.

Para o zoom in e out são o '+' e o '-' respetivamente.

3.2 XML Parser

3.2.1 Câmera

Para facilitar a colocação da câmera agora é possivel inserir (dentro de `<scene>`)

```
<scene>
    <camera posX="5" posY="5" posZ="5"/>
    <group>
        <model file="models/example.3d"/>
    </group>
</scene>
```

3.2.2 Translação

A sintaxe equivalente em xml é `<translate K="m»`, sendo m o valor da translação e K representa um dos eixos X, Y e/ou Z.

Ou seja, `<translate X="1»` move o objeto uma unidade segundo o eixo X, e corresponde ao comando glTranslatef(1,0,0).

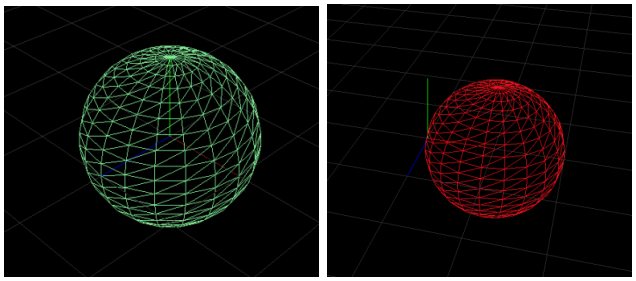


Figura 1: Resultado da transformação `<translate X="1"/>`

3.2.3 Rotação

Com o comando glRotatef(angle, x, y, z) efetuamos uma rotação do objeto em relação aos diferentes eixos(x, y, z), com um ângulo em graus e no sentido anti-horário. O comando equivalente em xml é `<rotate angle="a"axisK="m"/>`, sendo a o ângulo em graus, K representa os eixos X, Y e/ou Z e m toma o valor de 0 ou 1.

Por exemplo, `<rotate angle="90"axisX="1"/>` roda o objeto 90 graus, ao redor do eixo Y, no sentido anti-horário.

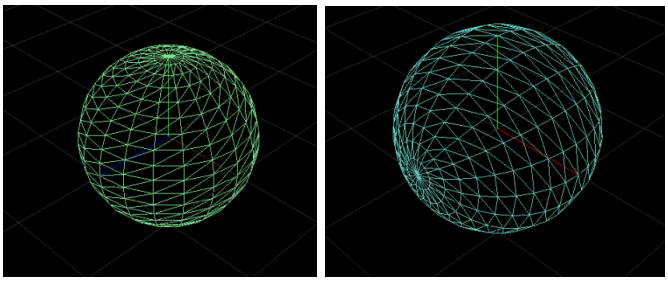


Figura 2: Resultado da transformação `<rotate angle="90"axisX="1"/>`

3.2.4 Escala

Para efetuar uma escala utilizamos o comando `glScalef(ex, ey, ez)` que altera a escala do objeto ao longo dos eixos coordenados.

O comando equivalente em xml é `<scale X="ex"Y="ey"Z="ez"/>`, sendo ex, ey e ez os valores da escala em relação aos respetivos eixos.

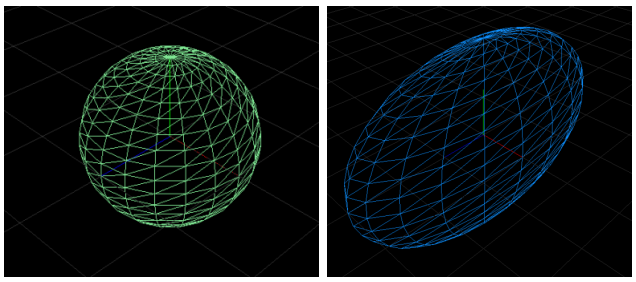


Figura 3: Resultado da transformação `<scale X="1"Y="2" Z="3"/>`

3.2.5 Cores

A fim de poder colorir os modelos 3D decidimos implementar um atributo nos mesmos que lhes atribui cor de acordo com cores predefinidas numa nova secção.

Essa nova secção é o nodo `<colors>`, que permite definir cores com nomes da seguinte forma:

```
<colors>
    <color name="red" r="1"/>
</colors>

<scene>
    <model file="models/example.3d" color="red"/>
</scene>
```

O nome 'random' é reservado, visto que se trata de uma cor que não necessita de estar definida, pois é atribuída aleatoriamente.

```
<scene>
    <model file="models/example.3d" color="random"/>
</scene>
```

Caso nenhuma cor seja fornecida o modelo será de cor branca.

4 Cenas Exemplo

Após a realização desta segunda fase construímos algumas cenas com a devida sintaxe xml incluindo a pedida cena inicial do sistema solar.

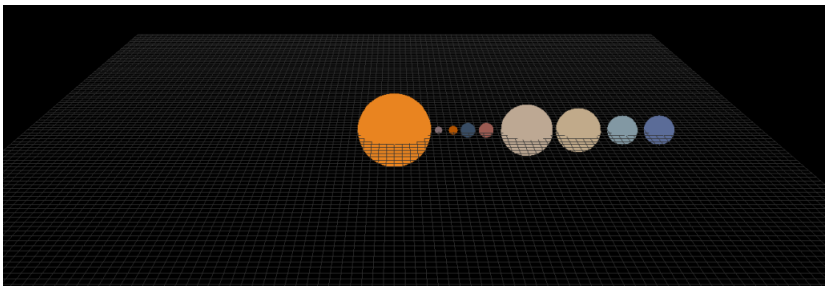


Figura 4: solar_system.xml

A cena que se segue é um teste de sistema solar mais completo com luas

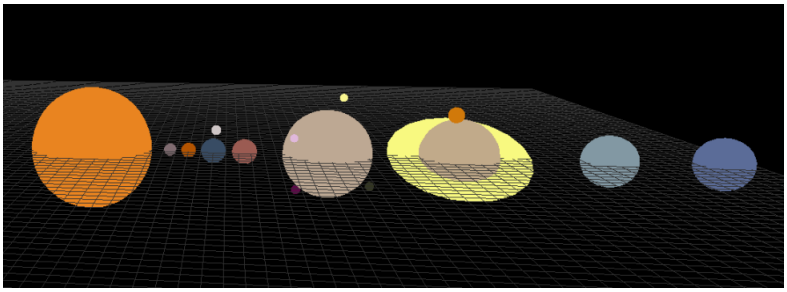


Figura 5: complete_solar_system.xml

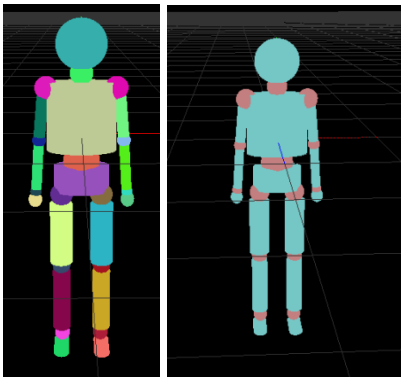


Figura 6: humanoid.xml

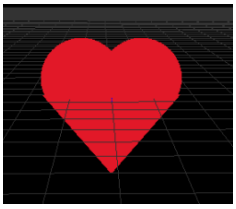


Figura 7: heart.xml

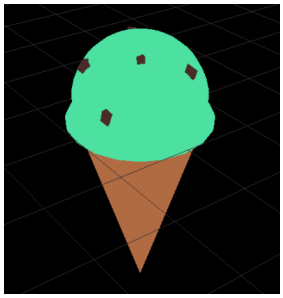


Figura 8: Another figure