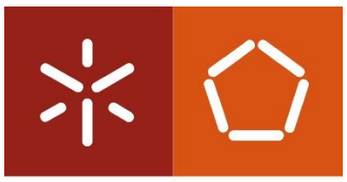

Universidade do Minho

Computação Gráfica

Fase 2

Benjamim Coelho, Henrique Neto, Leonardo Marreiros e
Júlio Alves

e-mail: {a89616,a89618,a89537,a89468}@alunos.uminho.pt

4 de abril 2021

1 Introdução

O objetivo desta fase é fazer uma melhoria ao *engine* de forma a ele passar a ler vários tipos de ficheiros *XML* constituídos por uma maior diversidade de elementos, nomeadamente translações, rotações e escalas. Desta forma torna-se muito mais fácil abstrair as transformações e modelos necessários para um dado modelo. Com esta nova funcionalidade podemos definir cenas mais complexas em formato xml e carregar essas mesmas cenas com o engine para serem apresentadas ao utilizador.

2 Parser XML

2.1 Transformações

As transformações são descritas pela classe abstracta *Transformation* que possui um único método virtual *void apply(void)* que aplica a transformação no esquema atual. Desta forma é possível aplicar a transformação independentemente da classe que seja derivada.

```
class Transformation {
public:
    virtual ~Transformation()=default;
    virtual void apply() = 0;
};
```

Figura 2.1

Desta classe derivam quatro classes, *Color*, *Rotation*, *Scale* e *Translation*.

2.1.1 Color

Esta classe instancia três floats que representam uma cor no sistema de cores *RGB*. Assim a função *apply* aplica a função *glColor3d* a partir dos termos instanciados.

```
class Color : public Transformation {
public:
    Color(float red, float green, float blue);
    float red, green, blue;
    void apply();
};
```

Figura 2.2

2.1.2 Rotation

Esta classe instancia quatro floats, sendo que três representam cada um dos eixos e a restante indica o ângulo. Os floats que representam os eixos indicam a direção da rotação e o ângulo indica qual foi o ângulo (em graus) de rotação da primitiva. Nesta caso a função *apply* aplica a função *glRotatef* a partir dos termos instanciados.

```
class Rotation : public Transformation {
public:
    Rotation(float x, float y, float z, float angle);
    float x, y, z, angle;
    void apply();
};
```

Figura 2.3

2.1.3 Scale

Esta classe instancia três floats, um para cada eixo. Cada float indica a escala do redimensionamento que é efetuado no respetivo eixo. Desta forma a função *apply* aplica a função *glScalef* a partir dos termos instanciados.

```
class Scale : public Transformation {
public:
    Scale(float x, float y, float z);
    float x, y, z;
    void apply();
};
```

Figura 2.4

2.1.4 Translation

Esta classe instancia três floats, um para cada eixo. Cada float indica o valor da translação que é efetuada relativamente ao seu eixo. Com isto a função *apply* aplica a função *glTranslatef* a partir dos termos instanciados.

```
class Translation : public Transformation {
public:
    Translation(float x, float y, float z);
    float x, y, z;
    void apply();
};
```

Figura 2.5

2.2 Grupos

Um grupo é constituído por transformações, modelos e outros grupos. Desta forma, sabe-se sempre quais são as ações indicadas em cada grupo e conseguimos hierarquizar a informação presente no ficheiro XML de forma a saber qual a ordem em que devemos fazer cada tarefa.

Para guardarmos a ordem e estrutura dos grupos que aparecem no ficheiro xml, utilizámos a classe Group para guardamos as informações numa árvore. Deste modo conseguimos preservar transformações feitas num grupo e aplicá-las aos grupos interiores (descendentes desse nodo). Para tal, ao ler o ficheiro xml, para cada grupo, criamos um nodo Group, onde guardamos as transformações que aparecem no inicio desse grupo, os modelos que têm de ser carregados e os grupos que aparecem dentro desse próprio grupo. Após a informação do ficheiro xml estar guardada nesta árvore, para criar a cena desejada temos de percorrer a árvore aplicando o seguinte algoritmo: chamar o método glPushMatrix(), aplicar as transformações guardadas nesse group, desenhar os modelos guardados, chamar o algoritmo para os grupos descendentes e de seguida chamar o método glPopMatrix(). Deste modo, com recurso aos métodos glPushMatrix() e glPopMatrix() garantimos que as transformações são aplicadas aos nodos descendentes mas não aos nodos exteriores, pelo que a cena é apresentada corretamente ao utilizador.

```
class Group {
 public:
     vector<Model> models;
     vector<Group> groups;
     vector<Transformation*> transformations;

     Group();
     void draw();

 };
```

Figura 2.6

2.3 Modelos

Os modelos referem-se às primitivas produzidas na primeira fase, em que estão guardados todos os vértices necessários para a construção da mesma. Cada nodo model apenas pode indicar um ficheiro.

2.4 Ciclo de Rendering

Após a leitura do ficheiro *XML*, caso não ocorram erros no ficheiro de leitura, a cena lida é instanciada na variável global *mainScene*. Com isto a função *renderScene* (responsável por exibir a cena na janela) chama o método *draw* ao objeto *mainScene* que procederá por chamar o método *draw* a todos os grupos que a constituem. Estes começam por aplicar as suas transformações. De seguida prosseguem para desenhar os seus modelos e terminam por fim ao aplicar o método *draw* aos seu subgrupos de forma recursiva. No final, quando todas as chamadas recursivas terminarem, a cena foi desenhada. Adicionalmente se o utilizador tiver adicionado a flag *-axis* ao chamar o programa a função *renderScene* irá também desenhar os eixos do referencial na cena através da função built-in *DEBUG_draw_axis*.

3 Demo

3.1 Sistema Solar

Para demonstrar o sucesso da execução desta fase do projeto criamos um modelo do sistema solar em XML, com os seus planetas e o satélite natural da Terra, a Lua.

```
(...)
<group>
        <translate x="0" y="0" z="-50" />
        <scale x="2" y="2" z="2" />
        <color r="0.8" g="0.6" b="0.1" />
        <models>
                <model file="sphere.3d" />
        </models>
</group>
<group>
        <translate x="0" y="0" z="-60" />
        <scale x="2.2" y="2.2" z="2.2" />
        <color r="0.1" g="0.1" b="0.6" />
        <models>
                <model file="sphere.3d" />
        </models>
        <group>
                <rotate x="0" y="1" z="0" angle="90"/>
                <translate x="0" y="0" z="-2"/>
                <scale x="0.5" y="0.5" z="0.5"/>
                <color r="255" g="255" b="255"/>
                <models>
                        <model file="sphere.3d"/>
                </models>
        </group>
</group>
<group>
        <translate x="0" y="0" z="-70" />
        <scale x="2.0" y="2.0" z="2.0" />
        <color r="0.8" g="0.4" b="0.1" />
        <models>
                <model file="sphere.3d" />
        </models>
</group>
(...)
```

Figura 3.1: Excerto ficheiro XML da representação do sistema solar

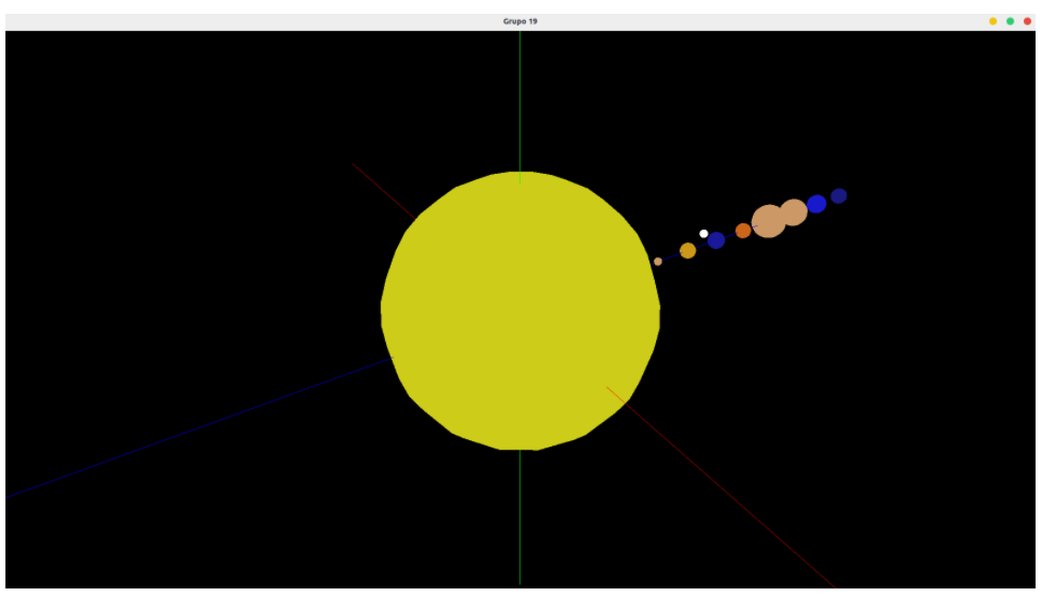

Figura 3.2: Representação do sistema solar

3.2 Árvore de Natal

Para além da cena pedida pela equipa docente, decidimos também criar uma árvore de natal decorada e com presentes. Escolhemos fazer esta cena porque esta exige que utilizemos todas as funcionalidades que o parser do XML consegue interpretar.

```
 (...)
<group>
        <translate y="54" z="2"/>
        <color r="0.8" g="0.8" b="0.1" />
        <rotate angle="+135" x="1" />
        <models>
                <model file="stretchedcone.3d" />
        </models>
 </group>
 <group>
        <translate y="54" z="-2"/>
        <color r="0.8" g="0.8" b="0.1" />
        <rotate angle="-135" x="1" />
        <models>
                <model file="stretchedcone.3d" />
        </models>
 </group>
 <group>
        <translate y="58" z="-2"/>
        <color r="0.8" g="0.8" b="0.1" />
        <rotate angle="315" x="1" />
        <models>
                <model file="stretchedcone.3d" />
        </models>
 </group>
 <group>
        <translate y="58" z="2"/>
        <color r="0.8" g="0.8" b="0.1" />
        <rotate angle="-315" x="1" />
        <models>
            (...)
 (...)
```

Figura 3.3: Excerto ficheiro XML da representação da árvore de Natal

Figura 3.4: Representação da árvore de natal decorada e com presentes

4 Conclusão

Este fase do projeto foi importante na medida em que pudemos melhorar o parsing do ficheiro XML que havia sido feito na fase anterior, o que nos permite criar e reproduzir cenas mais complexas devido a podermos utilizar várias primitivas e cada uma delas pode ser sujeita a transformações geométricas.

Pudemos também dar uso ao polimorfismo do C++ que nos permitiu uma maior flexibilidade na hora de criar as transformações.