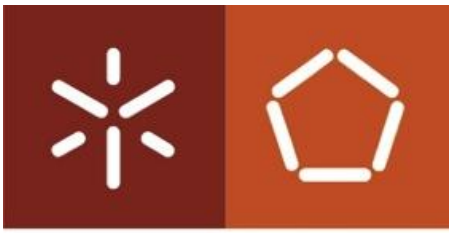

Universidade do Minho

Computação Gráfica

Relatório Projeto Prático – Parte II

Mestrado Integrado em Engenharia Informática

Ano Letivo 2016/2017

Docente: António José Borba Ramires Fernandes

Elementos do Grupo:

Daniel Teixeira Militão – A74557

Hugo Alves Carvalho – A74219

João Ismael Barros Dos Reis – A75372

Luís Miguel Da Cunha Lima – A74260

Índice

Índice 2
1. Introdução 3
2. Processo de leitura 4
   2.1. Ficheiro XML 4
   2.2. Sistema de leitura 5
3. Estruturas de Dados 6
4. Ciclo de rendering 8
5. Debug 9

1. Introdução

O presente relatório descreve a segunda etapa do projeto prático da unidade curricular de Computação Gráfica. Nesta etapa foi pedido que representássemos em 3D o sistema solar, especificado num ficheiro XML.

No ficheiro XML estará presente as várias translações, rotações e escalas para cada figura, bem como a discriminação dos ficheiros a usar que foram previamente criados pelo gerador.

A *engine* terá a função de ler esse ficheiro e representar de acordo com o que foi especificado no ficheiro XML.

Neste relatório vamos explicar, através de figuras e algoritmos em pseudocódigo, o nosso raciocínio para a realização das duas partes acima mencionadas.

2. Processo de leitura

2.1. Ficheiro XML

Previamente à aprimoração da *engine*, foi estabelecido um ficheiro XML com a função de representar o sistema solar, com os respetivos planetas e satélites naturais, que, neste contexto serão todos tratados como luas. A sua organização passa por um conjunto de grupos e subgrupos – nesta situação o grupo principal trata-se do Sol, que contem vários grupos correspondentes aos seus planetas, que por si contêm outros grupos, as luas.

Ainda neste ficheiro é possível representar as translações, rotações e escalas de cada um dos ficheiros a ser lidos.

```
<scene>
	<group a="Sun">
		<models>

			(...)

		<group a="Mars">
			<translate X="3" Y="0" Z="0" />
			<rotate angle="-25.19" axisX="0" axisY="0" axisZ="1" />
			<scale X="0.1" Y="0.1" Z="0.1" />
			<models>
				<model file="planet.3d" />
			</models>
			<group a="Moon">
				<translate X="-1" Y="0" Z="1" />
				<scale X="0.2" Y="0.2" Z="0.2" />
				<models>
					<model file="moon.3d" />
				</models>
			</group>
			<group a="Moon">
				<translate X="1" Y="0" Z="1" />
				<scale X="0.2" Y="0.2" Z="0.2" />
				<models>
					<model file="moon.3d" />
				</models>
			</group>
		</group>

			(...)

	</group>
</scene>
```

Figure 1 - Exemplo ficheiro XML

2.2. Sistema de leitura

Assim que acabado o ficheiro XML, partiu-se para a edição do sistema de leitura já criado, que recorre à função `void getFigures()`. Resume-se o seu funcionamento a uma leitura recursiva de cada grupo, como é possível observar nesta explicação detalhada:

1. Abre ficheiro "scene.xml";
    a. Se não obtiver sucesso, termina execução.
2. Procura primeiro elemento, "scene";
    a. Se não existir, termina execução.
3. Usando esse elemento como parâmetro para `getGroup()`, este devolve a arvore que contêm a informação do ficheiro XML.

Ainda a função `getGroup()`:

1. Cria arvore `t`;
2. Cria elemento `child`, que corresponderá ao primeiro filho do elemento recebido;
3. Itera ao último filho do elemento recebido;
    a. Se se tratar de um `translate`, coloca-se no nodo da árvore `t` tais valores;
    b. Se se tratar de um `rotate`, coloca-se no nodo da árvore `t` tais valores;
    c. Se se tratar de um `scale`, coloca-se no nodo da árvore `t` tais valores;
    d. Se se tratar de um `models`, itera-se sobre os existente `t` e adiciona-se ao nodo da árvore tais figuras;
    e. Se se tratar de um `group`, recursivamente usa-se a função `getGroup()` usando child como parâmetro, e coloca-se como nome no nodo arvore recebida o nome do tal `group`. Adiciona a árvore recebida como subárvore de `t`.
4. Retorna a árvore `t`.

3. Estruturas de Dados

A estrutura que entendemos que melhor representaria o que foi pedido passa por uma árvore que cada nodo poderá ter vários filhos.

Tomando em conta o exemplo do sistema solar, está representado abaixo em árvore que lhe corresponderia:

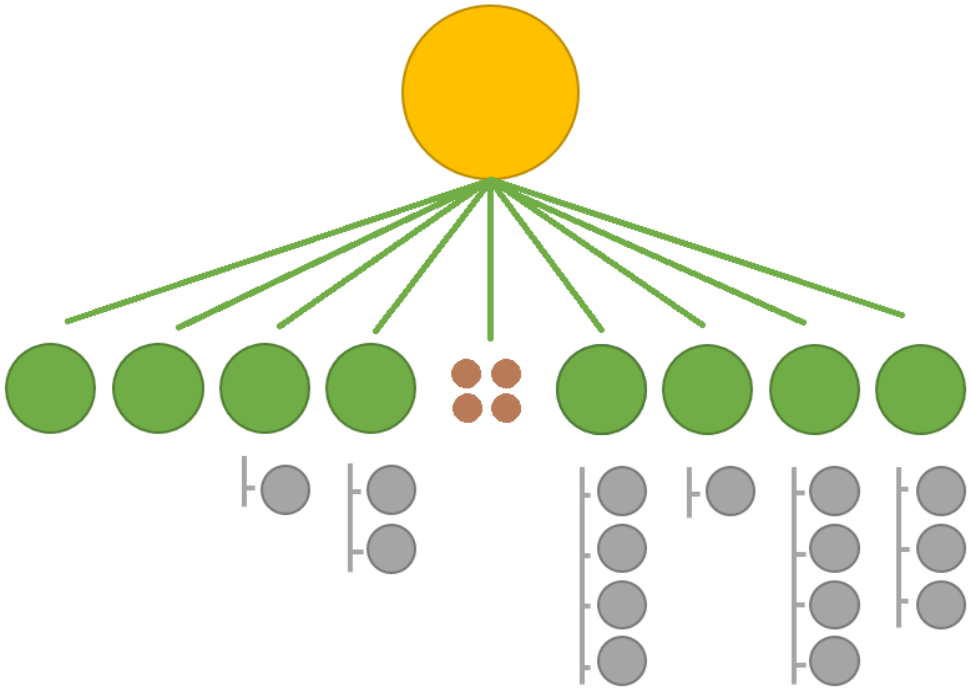

Figura 1 - Árvore

Temos como primeiro nodo da árvore o sol, e como respetivos filhos estão os planetas que por sua vez contêm as luas como filhos. A cintura de asteroides, presente entre o quarto e quinto planeta, está também assinalada como filho do sol.

De modo a criar a árvore foram criadas três classes:

```
class Coordinate {
	public:
		Coordinate() :empty(true) {}
		bool empty;
		float x, y, z, angle;
};
```

Esta classe representará um ponto com x, y, z. Terá vários propósitos:

1. Representar um ponto de um triângulo;
2. Representar uma translação;
3. Representar uma escala;
4. Representar uma rotação. Nesta situação, será também dado uso a `angle`.

O booleano `empty` será `true` se uma dada figura tiver translação, escala ou rotação, e usar-se-á na altura da representação das figuras.

```
class Figure {
public:
	string name;
	std::vector<std::vector<Coordinate>> figura;
	Coordinate rotate, translate, scale;
};
```

A classe Figure será utilizada para representar uma figura, que pode ser um conjunto de figuras 3D.

- `name` será o nome da figura, como “Sol”, “Marte” ou “Lua”;
- `figura` será vetor que conte um vetor com os vários pontos de um ficheiro 3D, ou seja, pode conter várias figuras.
- `rotate`, `translate`, `scale` corresponde às rotações, translações e escalas que a figurar sofrerá.

```
class Tree {
	public:
		Figure figure;
		std::vector<Tree> subtrees;
};
```

Finalmente, a classe `Tree` que corresponde à árvore terá como seu nodo uma `figure` e as respetivas subárvores.

4. Ciclo de *rendering*

De modo a fazer *rendering* ao projeto final usa-se uma pesquisa em profundidade à árvore de modo a representar e efetuar as translações, rotações e escalas.

A função `renderScane()` chamada a função `drawScene()`, que efectua a representação das figuras. De maneira mais detalhada, de modo a desenhar a cena usa-se a função `drawScene()`, que recebe como parâmetro uma arvore:

1. Efetua `glPushMatrix();`
2. Verifica se o a figura do nodo atual tem translação;
    a. Se tiver, efetua recorrendo a `glTranslatef();`
3. Verifica se o a figura do nodo atual tem rotação;
    a. Se tiver, efetua recorrendo a `glRotatef();`
4. Verifica se o a figura do nodo atual tem escala;
    a. Se tiver, efetua recorrendo a `glScalef();`
5. Recorre a `drawFigure()` com o vetor de figuras como parâmetro de modo a representar a figura do nodo atual;
6. Itera pelas subárvores;
    a. Recursivamente usará `drawScene()` usando cada elemento como paramento;
7. Efetua `glPopMatrix()`

A função `drawFigure()`, que recebe como parâmetro um vetor de vetores:

1. Itera sobre o vetor ;
    a. Iniciar desenho com `glBegin(GL_TRIANGLES);`
    b. Itera sobre o vetor que contem os pontos da figura;
        i. Representar o ponto com recurso a `glVertex3f(it->x, it->y, it->z);`
    c. Terminar desenho com `glEnd();`

A cada 3 pontos está também configurado para alternar entre duas cores.

5. Debug

De modo a ajudar no *Debug* das figuras foi programado uma câmara para melhor visualização e deteção de erros. Com recurso às teclas W, A, S e D é possível mover a figura para cima, esquerda, baixo ou direita, respetivamente. É possível ainda girar usando as setas cima, baixo, direita e esquerda e fazer *zoom* com as teclas I e O.

Finalmente, foi também incluída a possibilidade de ver as figuras por linhas, pontos ou preenchida usando as teclas Z, X e C, respetivamente.