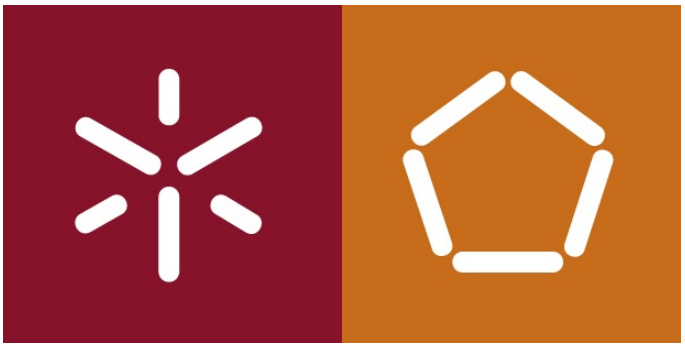


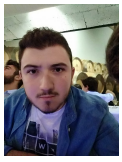
Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Computação Gráfica

Relatório do Projeto Prático - Parte 2

Autores :
Diana Costa (A78985)

José Oliveira (A78806)

Marcos Pereira(A79116)

Vitor Castro(A77870)


8 de Abril de 2018

Resumo

Perante a proposta de realizar cenas hierárquicas usando transformações geométricas, houve um impasse inicial devido à necessidade de uma boa estruturação do problema, nomeadamente, da organização da informação (acerca de translações, rotações e escalas de figuras) numa árvore e respetivos nodos. No fim, o grupo definiu que deveria ser capaz de fazer uma cena estática de um modelo de um sistema solar, incluíndo o sol, planetas, respetivas luas e a cintura de asteróides, definidos numa hierarquia.

Depois de algum tempo e trabalho, o resultado encontrado foi satisfatório, e os objetivos e respostas às questões do enunciado proposto cumpridos.

Conteúdo

1 Introdução

A continuação deste projeto surge no âmbito da unidade curricular de Computação Gráfica, do Mestrado Integrado em Engenharia Informática da Universidade do Minho. Era requerido, nesta segunda fase, que fosse produzido um molde de um sistema solar, em 3D, com o desenho hierárquico dos vários planetas, respectivas luas, cintura de asteróides e o sol. Para isto, foi necessário especificar um ficheiro XML, e alterar o *Engine* da primeira fase do trabalho prático.

O ficheiro XML é onde o grupo especifica as transformações válidas para uma figura (rotações, translações e escalas). Este ficheiro segue a seguinte estrutura:

```
<scene>
	<group>
		<translate X=1 />
		<models>
			<model file="sphere.3d" />
		</models>
		<group>
			<translate Y=1 />
			<models>
				<model file="cone.3d" />
			</models>
		</group>
	</group>
</scene>
```

Figura 1: Estrutura exemplificativa do ficheiro XML

O *Engine*, que tinha como objetivo ler um ficheiro XML, que continha as referências para os documentos gerados pelo *Generator* e dar como output os modelos pretendidos, vê-se agora alterado para ler, também, o novo ficheiro e dar como resultado o sistema solar ou qualquer outra figura e transformações.

Ao longo do presente relatório será explicado todo o raciocínio envolvente à realização das duas partes acima mencionadas, através dos algoritmos utilizados, imagens ilucidativas e excertos de código.

Em suma, a secção 2 descreve o processo de leitura do ficheiro XML, começando com uma secção que fala do ficheiro em si e a sua estrutura (2.1). Esta secção termina com a explicação do sistema de leitura do ficheiro, em 2.2.

Segue-se o esclarecimento e raciocínio envolvente das estruturas de dados utilizadas na secção 3, onde se demonstra, em primeiro lugar, uma representação da árvore utilizada para armazenar informação, seguida das classes que a constituem.

De forma a explicar o rendering das figuras, surge a secção 4, que explica o algoritmo utilizado para busca na árvore. Segue-se uma secção de *debug*, em 5, onde são explicados os mecanismos utilizados pelo grupo, por forma a validar os resultados obtidos nesta fase do projeto.

O relatório termina com uma breve conclusão na Secção 6, onde é feito um balanço do trabalho realizado, tendo em conta as dificuldades ao longo do desenvolvimento do mesmo. Também em "Anexos", na Secção 7, apresenta-se o resultado final deste trabalho prático.

2 Processo de Leitura

2.1 Ficheiro XML

O ficheiro XML criado tem como o objetivo representar o sistema solar proposto, contendo os respetivos planetas e satélites naturais. A *scene* é composta por um *group* que contém as várias translações, rotações e escalas a ser aplicadas à respetiva figura indicada na tag *model*. Cada *group* por sua vez pode ter outros *group* como filhos.

Em baixo, segue-se um excerto do ficheiro XML criado pelo grupo.

```
<scene>
        <group name="Sun">
                <models>
                        <model file="sun.3d" />
                </models>
                <group name="Mercury">
                        <translate X="1.3" Y="0" Z="0" />
                        <rotate angle="-0.1" axisX="0" axisY="0" axisZ="1"/>
                        <scale X="0.07" Y="0.07" Z="0.07" />
                        <models>
                                <model file="planet.3d" />
                        </models>
                </group>
                ...
        </group>
</scene>
```

Considera-se, assim, que todos os astros serão como "satélites"uns dos outros. Tem-se que o sol será a estrela principal, que terá na sua orbita como satélites todos os planetas, que por sua vez terão, cada um, as suas luas em órbita.

2.2 Sistema de Leitura

Após serem feitas as alterações no ficheio XML, foi necessário alterar o sistema de leitura já criado na fase anterior. Foram construídas três funções para o efeito, declaradas no ficheiro *xml-loader*.

A função `engine::modelTree xmlLoader::loadSceneXML(const char* path)`, que recebe como único parâmetro o caminho até ao ficheiro XML, é a responsável por iniciar este processo.

Esta função faz o seguinte:

- Começa por tentar carregar o documento no caminho recebido, terminando com um erro se não o tiver encontrado ou se tiver ocorrido um erro.
- Procura o elemento pai *<scene>*.
- Chama a função `getModelTreeFromXML(scene)`

A função `engine::modelTree xmlLoader::getModelTreeFromXML(XMLElement* root)` recebe como argumento um nó de XML de um *group*, por exemplo *<group name="Sun">* e retorna uma árvore de modelos que se explicará mais à frente.

Esta função faz o seguinte:

- Inicializa uma árvore de modelos *tree*;
- Cria um elemento *child*, que corresponde ao primeiro filho do elemento recebido;
- Itera até ao último filho do elemento recebido;
  - Obtém o valor da tag;
  - Caso seja um `translate`, coloca-se no nó da árvore *tree* os respetivos valores;
  - Caso seja um `rotate`, coloca-se no nó da árvore *tree* os respetivos valores;
  - Caso seja um `scale`, coloca-se no nó da árvore *tree* os respetivos valores;
  - Caso seja um *models*, itera-se sobre os existentes e através da função `loadModel(filename)` obtendo-se a figura, adicionando ao nó da árvore.
  - Caso seja um *group*, faz-se uma chamada recursiva da função `getModelTreeFromXML(child)` usando *child* como parâmetro. Por fim adiciona-se a nova árvore de modelos como subárvore de *tree*.
- Retorna a árvore *tree*.

A função `engine::model xmlLoader::loadModel(string path)` recebe como parâmetro um path para um ficheiro .3d, por exemplo planet.3d, e faz load do mesmo criando um *model* que será no fim retornado.

3 Estruturas de Dados

A estrutura de dados desenhada para a resolução deste problema passa pela implementação de uma árvore, em que cada nó pode ter múltiplos filhos. Tomando como exemplo o sistema solar, representa-se abaixo um modelo que demostra esta situação:

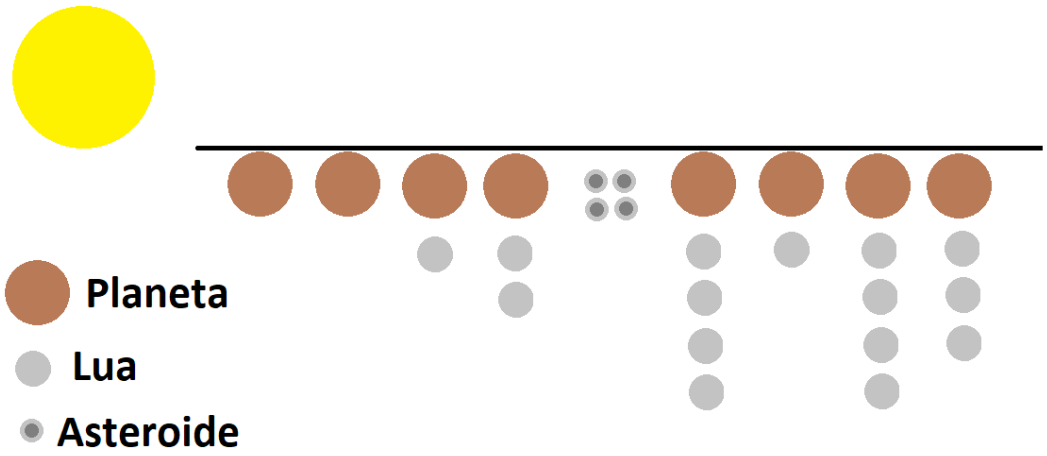


Figura 2: Árvore de modelos

O primeiro nó da árvore é o Sol, e os respetivos filhos são os planetas, que por sua vez têm como filhos os seus satélites naturais. Entre o quarto e o quinto planeta tem-se a cintura de asteroides que é também um "filho"do Sol.

A implementação do desenho anterior é feita com auxílio de um conjunto de *structs*.

```
// A vertex has x y and z coordinates
struct vertex {
            float x, y, z;
};

// A model is a collection of vertices
struct model {
    std::vector<vertex> vertices;
};

struct modelTree {
            std::string name;
            std::vector<model> models;
            bool applyTranslate = false;
            bool applyRotate = false;
            bool applyScale = false;
            vertex translate;
            vertex rotate;
            float rotateAngle;
            vertex scale;
            std::vector<modelTree> branches;
};
```

A estrutura principal é a *modelTree* introduzida nesta fase dois, cada nó desta árvore possui:

- Uma string `name` - que contém o valor especificado no ficheiro XML pelo atributo "name" (exemplo: <group name="AsteroidBelt">);
- Um vetor `models` - contém os *model* que representam as figuras;
- Três booleanos `applyTranslate`, `applyRotate`, `applyScale` que determinam se é necessário aplicar alguma destas operações neste nó e aos respetivos filhos;
- Um vertex `translate`, `rotate`, `scale` que caso não sejam null, devem ser aplicados a todos os models (incluindo os filhos). E caso `rotate` não seja null, é esperado um float `rotateAngle`;
- Um vetor `branches` que contém os respetivos filhos;

4 Ciclo de Rendering

De maneira a fazer o *rendering* do sistema solar, foi usada uma pesquisa em profundidade da árvore, de modo a representar as figuras e efetuar as translações, rotações e escalas necessárias.

O processo começa quando a função `renderScene()` é chamada no *main.cpp* que por sua vez chama a função `drawFrame()` que chama a função `drawScene(modelTree tree)` recebendo como argumento a árvore de modelos.

O algoritmo da função `drawScene(modelTree tree)` é descrito de seguida:

- É efetuado glPushMatrix() (para assegurar que as transformações efetuadas em chamadas recursivas anteriores serão igualmente aplicadas);
- Verifica se existem transformações a serem aplicadas aos *models* do *group* atual, bem como aos *models* das respetivas sub-árvores. Deste modo, se `applyTranslate`, `applyRotate` ou `applyScale` forem `true` são efetuadas as transformações através da `glTranslate()`, `glRotatef()`, `glScalef()` respetivamente;
- Itera pelo vetor dos *models* e através da função `drawModel()` desenha as figuras correspondentes a este nó da árvore;
- Itera pelas subárvores, chamando recursivamente a função `drawScene()`;
- É efetuado glPopMatrix() (para assegurar que voltamos ao estado das transformações do `group` anterior);

5 Debug

De maneira a verificar os resultados obtidos, foram adicionadas algumas funcionalidades à interface do programa. A tecla F1 foi definida para renderizar apenas as arestas das figuras, enquanto que a F2 define o modo de renderização que inclui as faces dos modelos. As teclas W, A, S e D rodam os modelos, enquanto que as setas movem-nos no espaço.

Com a ajuda destas funcionalidades foram descobertos alguns problemas ao longo do desenvolvimento do programa que, por terem sido identificados a tempo, foram rapidamente resolvidos.

6 Conclusão

Esta segunda fase do projeto foi bastante importante e enriquecedora, pois permitiu aos membros do grupo perceber e interiorizar melhor os conceitos abordados nas aulas práticas da Unidade Curricular de Computação Gráfica. Deste modo, foram adquiridos conhecimentos básicos acerca da biblioteca de funcionalidades GLUT para OpenGL, bem como conhecimento relativo a um conjunto de primitivas gráficas, que certamente serão úteis para as fases seguintes do projeto prático.

No que diz respeito à continuação do desenvolvimento da aplicação Engine, a principal dificuldade encontrada foi a realização de algumas funções de *parsing* da árvore do modelo do sistema solar e chegar ao algoritmo de pesquisa em profundidade do mesmo. Posto isto, com algum trabalho e aproveitando as bases adquiridas nas aulas,a equipa foi capaz de produzir com sucesso os algoritmos responsáveis pela estruturação dos modelos.

Quanto ao desenvolvimento do ficheiro XML, um dos grandes obstáculos foi a divisão do problema em partes de maneira a seguir a abordagem mais simples e eficaz. Depois de desenhada a solução, através de uma estrutura de dados em árvore, a implementação foi relativamente simples.

Também uma das partes mais interessantes, complexas, e recompensadoras deste trabalho foi perceber o funcionamento da stack de transformações, e como o uso desta facilita enormemente a aplicação hierárquica das translações, rotações, e redimensionações dos modelos.

Em suma, é feita uma apreciação positiva relativamente ao trabalho realizado, visto que a implementação de todas as funcionalidades propostas foram conseguidas com sucesso. O grupo conseguiu tirar partido dos conhecimentos adquiridos neste projeto, sentido-se capaz de, num contexto futuro, aplicar os conceitos subjacentes de forma eficaz.

7 Anexos

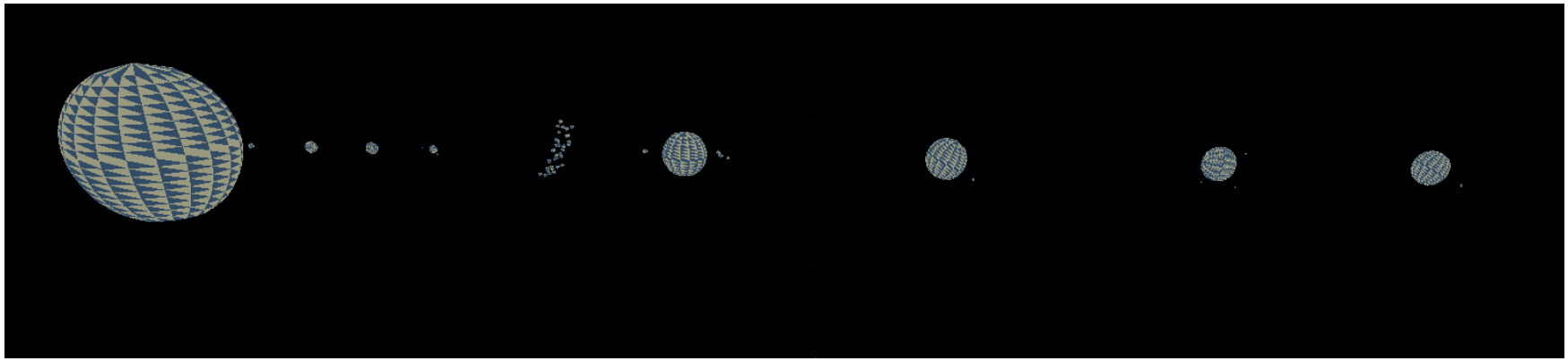


Figura 3: Sistema solar - Resultado final, utilizando as esferas da fase 1 do projeto