Universidade do Minho
Mestrado Integrado em Engenharia Informática

Computação Gráfica

Fase 2 - Grupo 51
Abril 2020

Figure 1: A89536
Ana Gomes

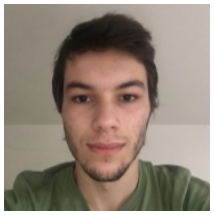

Figure 2: A89536
Francisco Franco

Figure 3: A89525
Maria Barros

Figure 4: A89535
Beatriz Lacerda

1 Introdução e Objetivos

Nesta Unidade Curricular de Computação Gráfica, foi-nos dada uma sugestão do desenvolvimento de modelos 3D, utilizando OpenGL com base na biblioteca GLUT e todo o projeto foi feito na linguagem C++.
Este foi dividido em 4 fases distintas, sendo que nesta 2ª fase o principal objetivo era, partindo de transformações geométricas, criarmos cenários hierárquicos de modo a termos um Sistema Solar estático, incluindo o sol, os planetas e também as luas.

2 Resumo

Para esta segunda fase do projeto, ainda que naturalmente utilizemos funcionalidades criadas na primeira fase, tivemos a necessidade de proceder à criação de uma nova estrutura XML, de modo a que fosse possível a utilização das primitivas geométricas, previamente representadas, em modelos mais complexos.
Assim, com estas novas alterações, o uso de transformações geométricas como translações, rotações e à escala torna-se possível, ficando estas assim responsáveis pelo atual modo de exibição de cada uma das primitivas.
Concluindo, todas as modificações que efetuamos nesta segunda fase são com o intuito de gerar as primitivas geométricas de modo a formar um modelo estático do Sistema Solar.

3 Estrutura do projeto

3.1 Módulos

Anteriormente na primeira fase do projeto, já tinhamos explicado a ultilidade de cada módulo e como cada um destes funcionava, no entanto para esta segunda fase houve algumas alterações, nomeadamente a criação de uma nova estrutura XML, a serem feitas e por isso é imprescindível falarmos novamente dos mesmos.

3.1.1 Generator

O **generator.cpp**, à semelhança do que foi explicado na primeira fase, é o módulo onde estão determinadas as estruturas para cada uma das formas geométricas a representar, com o objetivo de gerar os vértices das figuras. Nesta fase, mantivemos idênticas as restantes classes que já tinhamos desenvolvido e decidimos que seria útil adicionarmos a primitiva geométrica **Torus**. Por esta razão, criamos uma classe no Generator com as novas características que dessem oportunidade de gerar também esta nova primitiva, tal como o acontece com as restantes.

3.1.2 Engine

O **engine.cpp** contém as funcionalidades principais do projeto, como por exemplo a interpretação e leitura dos ficheiros XML.
Após a criação de uma nova estrutura XML, tivemos a necessidade de alterar o método usado para efetuar o *parsing*. Para além disso, como agora temos também grupos de figuras geométricas, cada um com a informação necessária para a sua definição, achamos por bem termos um armazenamento da informação diferente, que a organiza hierarquicamente, e posteriormente, o GLUT vai processá-la de maneira distinta também.

3.2 Classes

Como já foi anteriormente dito, para esta segunda fase do projeto, necessitamos de interpretar uma estrutura XML diferente uma vez que foi criada uma nova para esta fase. Assim, procedemos à criação de novas classes que nos irão ajudar nesse sentido.
Para além das classes já existentes **Ponto** e **Shape**, temos agora também as classes **Scale**, **Rotate**, **Transformation**, **Translate** e **Group**. Todas estas apresentam variáveis de instância, construtores, getters e setters, como iremos explicar de seguida.

3.2.1 Translação

A classe **Translate** contém tudo que é necessário para efetuar uma translação, ou seja, variáveis de instância **x**, **y** e **z**, que irão descrever o vetor usado para a execução desta.

```
#ifndef Engine_Translate_H
#define Engine_Translate_H

#include <stdio.h>
#include <string.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <stdlib.h>
#include <cstring>
#include <vector>

using  namespace std;

class Translate{
        private:
        float x;
        float y;
        float z;

        public:
        Translate();
        Translate (float x, float y, float z);
        float getX();
        float getY();
        float getZ();
        void setX(float a);
        void setY(float a);
        void setZ(float a);
};

#endif //Engine_Translate_H
```

Figure 5: Translate.h

3.2.2 Rotação

A classe **Rotate** contém o necessário para efetuar uma rotação, isto é, tal como na translação apresenta as variáveis de instância **x**, **y**, e **z**, que ajudam a representar o vetor usado para a execução desta, e ainda uma variável **angle** que consiste no ângulo a aplicar sobre o vetor.

```
#ifndef Engine_Rotate_H
#define Engine_Rotate_H

#include <stdio.h>
#include <string.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <stdlib.h>
#include <cstring>
#include <vector>

using  namespace std;

class Rotate{
        private:
        float angle;
        float x;
        float y;
        float z;

        public:
        Rotate();
        Rotate (float angle, float x, float y, float z);
        float getAngle();
        float getX();
        float getY();
        float getZ();
        void setAngle(float a);
        void setX(float a);
        void setY(float a);
        void setZ(float a);
};

#endif //Engine_Rotate_H
```

Figure 6: Rotate.h

3.2.3 Escala

A classe **Scale** contém o necessário para uma alteração de dimensões a uma figura, ou seja, precisamos de uma relação entre as 3 variáveis de instância **x**, **y** e **z** originais e as mesmas após o redimensionamento.

```
#ifndef Engine_Scale_H
#define Engine_Scale_H

#include <stdio.h>
#include <string.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <stdlib.h>
#include <cstring>
#include <vector>

using  namespace std;

class Scale{
        private:
        float x;
        float y;
        float z;

        public:
        Scale();
        Scale (float x, float y, float z);
        float getX();
        float getY();
        float getZ();
        void setX(float a);
        void setY(float b);
        void setZ(float c);
};

#endif //Engine_Scale_H
```

Figure 7: Scale.h

3.2.4 Cor

A classe **Cor** tem toda a informação essencial a que seja possível atribuir uma cor a um dado planeta. Esta é processada em formato RGB e, por isso, temos 3 variáveis de instância r (red), g (green) e b (blue).

```
#ifndef Engine_Cor_H
#define Engine_Cor_H

#include <stdio.h>
#include <string.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <stdlib.h>
#include <cstring>
#include <vector>

using  namespace std;

class Cor{
        private:
        float r;
        float g;
        float b;

        public:
        Cor();
        Cor (float r, float g, float b);
        float getR();
        float getG();
        float getB();
        void setR(float a);
        void setG(float b);
        void setB(float c);
};

#endif //Engine_Cor_H
```

Figure 8: Cor.h

3.2.5 Transformação

A classe **Transformation** contém toda a informação para se tornar possível uma transformação geométrica tal como rotação, translação ou escala. Como variáveis de instância possui uma rotação rt, uma translação tr e uma escala sc.

```
#ifndef Engine_Transformation_H
#define Engine_Transformation_H

#include <stdio.h>
#include <string.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <stdlib.h>
#include <cstring>
#include <vector>
#include "Rotate.h"
#include "Scale.h"
#include "Translate.h"
#include "Cor.h"

using  namespace std;

class Transformation{
        private:
        Rotate rt;
        Scale sc;
        Translate tr;
        Cor cor;

        public:
        Transformation();
        Transformation (Rotate x, Scale y, Translate z, Cor c);
        Rotate getRotate();
        Scale getScale();
        Translate getTranslate();
        Cor getCor();
};

#endif //Engine_Transformation_H
```

Figure 9: Transformation.h

3.2.6 Grupo

A classe **Group** é a classe que agrupa a informação necessária a um dado grupo, ou seja, sempre que o programa ler os ficheiros input em formato XML, divide-os em grupos e temos assim a informação respetiva a cada grupo. Assim, sabemos o seu **nome** (*nome*), a transformação efetuada (*t*), que inclui translação, rotação ou escala, um vetor de pontos (*pontos*) e ainda os grupos inseridos nesse grupo (*filhos*).

```
#ifndef Engine_Group_H
#define Engine_Group_H

#include <stdio.h>
#include <string.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <stdlib.h>
#include <cstring>
#include <vector>
#include "Transformation.h"
#include "Ponto.h"

using  namespace std;

class Group{
        private:
        string nome;
        Transformation t;
        vector<Group> filhos =vector<Group>();
        vector<Ponto> pontos = vector<Ponto>();

        public:
        Group(string n, Transformation tf, vector<Group> g, vector<Ponto> p);
        vector<Ponto> getPontos();
        vector<Group> getFilhos();
        Transformation getTrans();
        void imprime();
};

#endif //Engine_Group_H
```

Figure 10: Group.h

3.3 Auxiliares

3.3.1 Parse

A classe **Parse** apresenta os métodos necessários para ser efetuado o *parsing* dos ficheiros XML. Assim, estes métodos, aliados aos da classe **tinyxml2**, tornam possível a leitura e interpretação de um ficheiro XML e, consequentemente, o armazenamento em estruturas já definidas da informação mais relevante.
Mais à frente, iremos explicar mais detalhadamente todo o processo de leitura de ficheiros que envolve o *parse*.

```
#ifndef Engine_Parse_H
#define Engine_Parse_H

#include <stdio.h>
#include <string.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include <stdlib.h>
#include <cstring>
#include <vector>
#include "tinyxml2.h"
#include "Group.h"

using namespace tinyxml2;
using  namespace std;

void parseXML(XMLElement* group, vector<Group> *g);
void parseScale(XMLElement* elemento, Scale *scale);
void parseRotate(XMLElement* elemento, Rotate *rotate);
void parseCor(XMLElement* elemento, Cor *cor);
void parseTranslate(XMLElement* elemento, Translate *translate);
void readFile(string fich);
vector<Group> lerXML(string ficheiro);

#endif //Engine_Parse_H
```

Figure 11: Parse.h

3.3.2 tinyxml2

A classe **tinyxml2** explora o conteúdo dos ficheiros XML, axiliando assim no parsing.

4 Primitivas Geométricas

4.1 Generator

4.1.1 Torus

O **Torus** é uma figura geométrica que se aproxima a uma câmara de pneu. Para o definirmos, necessitámos de um raio interior *r1*, um raio exterior *r2*, o número de divisões de torus *divd* e o número de divisões da circunferência que dão expessura ao torus *divc*.

As amplitudes necessárias são definidas por:
double alpha = (2 * MPI) / divc;
double beta = (2 * MPI) / divd;

Posteriormente, com a ajuda das funções *sin* e *cos*, partindo dos pontos iniciais, chegamos aos vários pontos das circunferências.

- **Pontos**

```
float x1 = cos(j) * (cos(i) * r1 + r2);
float y1 = sin(j) * (cos(i) * r1 + r2);
float z1 = r1 * sin(i);

float x2 = cos(j + beta) * (cos(i) * r1 + r2);
float y2 = sin(j + beta) * (cos(i) * r1 + r2);
float z2 = r1 * sin(i);

float x3 = cos(j + beta) * (cos(i + alpha) * r1 + r2);
float y3 = sin(j + beta) * (cos(i + alpha) * r1 + r2);
float z3 = r1 * sin(i + alpha);

float x4 = cos(j) * (cos(i + alpha) * r1 + r2);
float y4 = sin(j) * (cos(i + alpha) * r1 + r2);
float z4 = r1 * sin(i + alpha);
```

5 Engine

5.1 Processo de leitura

Depois de termos todas as estruturas de dados definidas de acordo com os novos requisitos da segunda fase do projeto, conseguimos ler e guardar a informação que considerarmos necessária do novo ficheiro XML.
Com o auxílio de todos os métodos que criamos para ser possível o *parse*, percorre-se às várias transformações, verificando o tipo de cada um e, seguindo esse parâmetro, guarda-se a informação nas respetivas classes já criadas.
Posteriormente, vemos quais os modelos que sofreram alterações e guardamos a informação na classe **Group**, de modo a que posteriormente, as figuras possam ser representadas com as respetivas transformações.
Para finalizar o processo, percorre os filhos ou os irmãos e, caso falte realizar o *parse* a algum, o processo é efetuado mais uma vez.

5.2 Ciclo de Rendering

Este ciclo foi criado com o objetivo de desenhar as figuras e sabe-se que, a cada iteração do ciclo, é feito um **getTrans()** de modo a conseguirmos a transformação efetuada. Posteriormente, após obtermos esses dados, utilizamos as funções **glRotatef()**, **glTranslatef()**, **glScalef()** e **glColor3f()** de modo a efetuarmos todas as transformações.
Para finalizar, apenas nos falta percorrer o dado grupo e, para cada figura inserida neste, desenhá-la sequencialmente, de modo a formarmos o pretendido.
Este processo ocorre para cada grupo-filho de forma recursiva, com o objetivo de renderizar toda a informação que vem no vetor de *Group*.

6 Análise de resultados

Refletindo sobre o trabalho realizado, o grupo está satisfeito com os resultados visto que todos os elementos do Sistema Solar estão representados de um modo muito semelhante ao que eles são na realidade.

6.1 Visualização geral

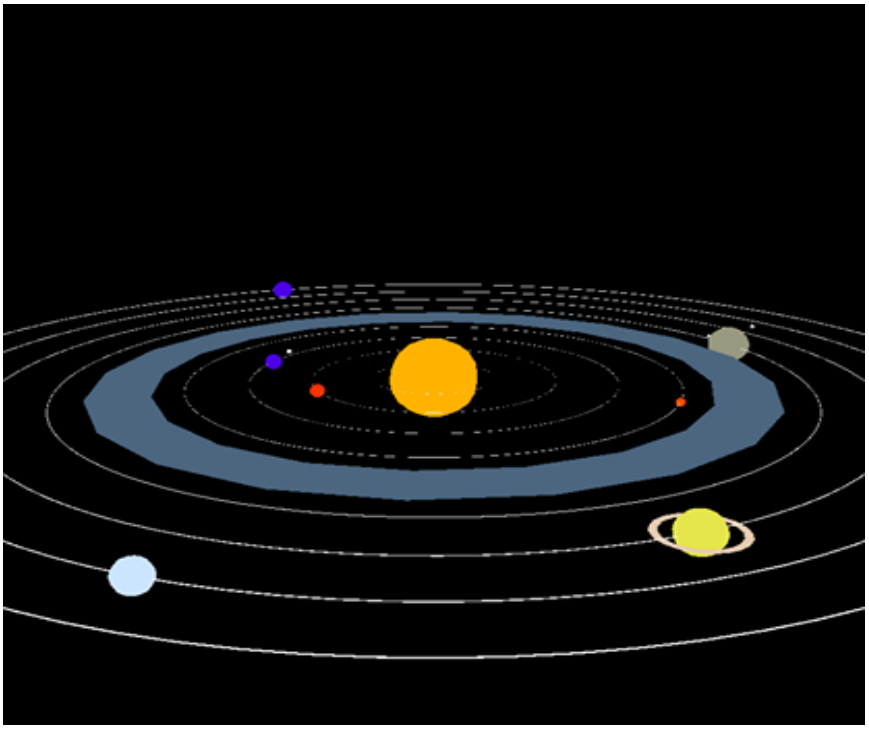

Figure 12: Perspetiva geral do Sistema Solar

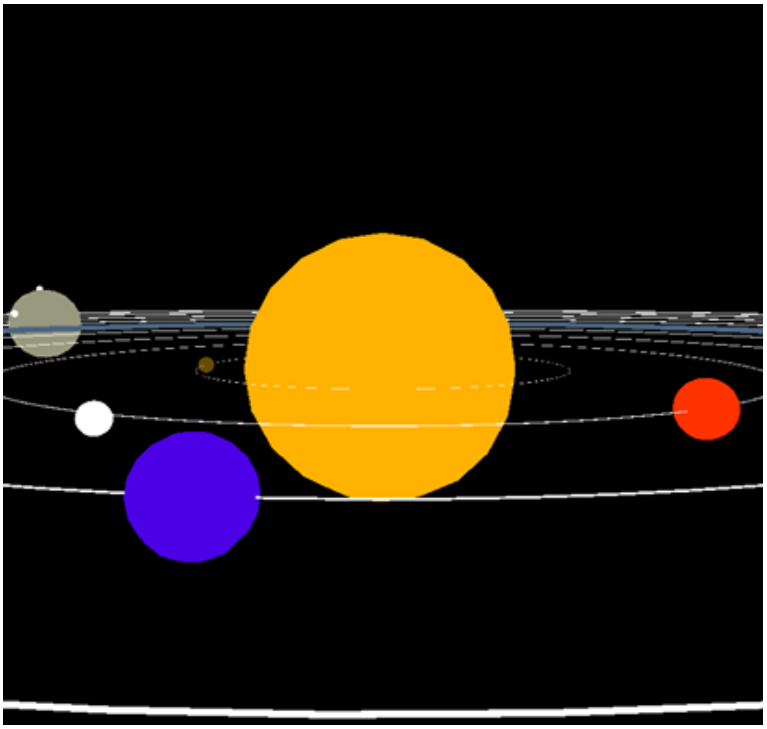

Figure 13: Outra prespetiva do Sistema Solar

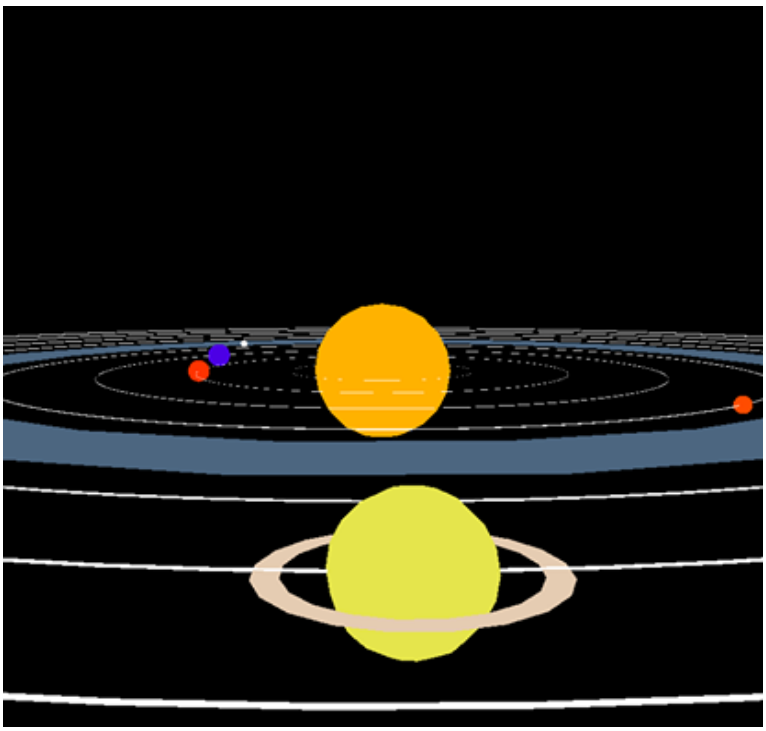

Figure 14: Prespetiva aproximada de Saturno

7 Conclusão e Trabalho Futuro

Primeiramente, consideramos que o nosso desempenho nesta segunda fase do projeto se refletiu no facto de conseguirmos atingir todos os objetivos propostos e até mesmo conteúdo extra. Tudo o que foi desenvolvido por nós correspondeu às expectativas, isto é, chegamos a um produto final que em pouco se distanciava da realidade.
Para além disso, concluímos que esta fase foi um pouco mais rápida, talvez porque o grupo começa a estar cada vez mais à vontade com os conteúdos abordados na UC e com as ferramentas utilizadas na mesma. No entanto, foi mais complexa no sentido em que tivemos de criar uma nova estrutura de leitura de ficheiros XMl e, por isso mesmo, obrigou a mais alterações no processo de *parsing*.
Em suma, o nosso desejo é que, com as restantes fases que ainda nos restam até que o projeto esteja concluído, tenhamos a oportunidade de melhorar e aperfeiçoar ainda mais o nosso produto final de modo a que este chegue o mais próximo possível da realidade.