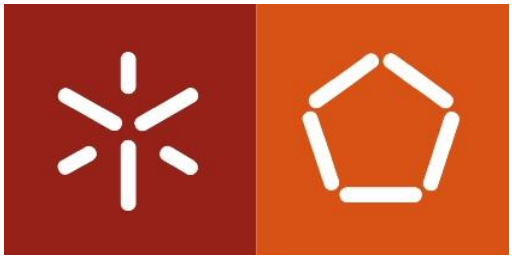
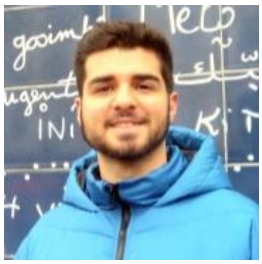


Computação Gráfica - CG 2019/2020

FASE 2

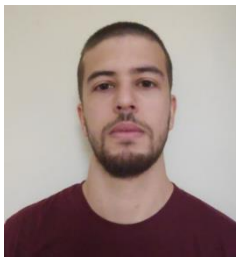
Mestrado Integrado em Engenharia Informática

José Pinto
A84590



Eduardo Costa
A85735

Luís Lopes
A85367



Ricardo Carvalho
A84261

Índice

Introdução

No âmbito da Unidade Curricular de Computação Gráfica, foi-nos proposto o desenvolvimento de um mecanismo 3D baseado num cenário gráfico com o auxílio de duas ferramentas utilizadas, também, nas aulas práticas, sendo elas o OpenGL e a linguagem C++.

Este relatório, relativo à segunda fase do projeto, foca-se na criação de cenários hierárquicos usando transformações geométricas. Para tal foi necessário alterar o ficheiro XML, bem como a forma como lemos e interpretamos a sua informação. Para além disto foi também requisitada uma representação do sistema solar.

Todo o processo apresenta-se, então, neste relatório, desde as alterações feitas ao que havia sido desenvolvido anteriormente, na primeira fase, às implementações feitas para estas segundas tarefas.

1. Reimplementação do *parser XML*

Um dos pontos mais importantes deste projeto é o desafio de ler, de forma correta, a informação contida nos ficheiros XML. Na primeira fase, como era apenas necessário guardar os nomes dos ficheiros .3d, a implementação do *parser XML* foi bastante simples e utilizamos *tinyxml* para o fazer. Nesta fase, para além disso, foi também necessário processar a informação relativa a operações dos campos *translate*, *rotate* e *scale* em ficheiros mais complexos, com subgrupos de outros grupos. Após alguma pesquisa, e como o *parser* teria de ser completamente reimplementado, decidimos fazê-lo, desta vez, em *tinyxml2.* A nosso ver, esta mudança permitiu-nos que a implementação fosse mais rápida e sucinta, facilitando-nos o processo. Para o efeito, necessitamos de criar e utilizar estruturas auxiliares, que serão apresentadas mais à frente, no capítulo dedicado a esse tema.

Quanto à implementação propriamente dita, esta é bastante simples e trivial, como já foi referido. O *parserXML* lê o ficheiro a partir da sua raiz, atribuindo níveis de hierarquia, especificados como índice do *array Group* por nós criado. Quando é lida a primeira *tag* da hierarquia (grupo principal), é feita uma leitura por uma função auxiliar, que lhe atribui hierarquia 0. Assim, sempre que o *parser* encontra uma operação *openGL,* esta é inserida na estrutura auxiliar na posição relativa à sua hierarquia, de modo a que, quando se encontra um *"model file"*, se consigam colocar na estrutura principal as operações relativas a esse ficheiro. Quando são encontrados subgrupos, usamos recursividade de forma a aumentar o nível da hierarquia até ao fim desse grupo e seus subgrupos. Quando saímos de um grupo, apagamos a informação na estrutura auxiliar relativa ao grupo em que nos encontramos.

2. Estruturas de dados

Para a reimplementação do *parser* necessitamos de duas estruturas de dados auxiliares.

- ***OperAux***

Estrutura de dados auxiliar ao *parser,* que foi necessário implementar aquando da abordagem dos subgrupos existentes no ficheiro XML. Criamos, então, a estrutura *OperAux* que não é mais do que uma lista ligada lista ligada que contém a informação sobre as diferentes operações que vão sendo encontradas ao longo do ficheiro, assim como as variáveis a si associadas. Assim sendo, poderíamos ter uma lista ligada associada a cada nível de hierarquia.

```
typedef struct operationAux {
    char* operation;
    double x, y, z, angle;
    struct operationAux* next;
} OperAux;
```

Figura 1 - Estrutura OperAux

- ***Group***

Para tratar o problema das hierarquias dos grupos dos ficheiros XML, criamos uma estrutura chamada "*Group*", que consiste num *array*, em que cada posição contém um apontador para uma lista ligada *OperAux,* associando, assim, posição do *array* ao nível da hierarquia do ficheiro XML. Como explicado anteriormente, este nível é aumentado recursivamente, à medida que se encontravam subgrupos. Sempre que este volta a aumentar, uma nova lista ligada *OperAux* é criada nesse nível, mas os níveis anteriores não são apagados, pois é necessário que as transformações de um grupo principal sejam também aplicadas aos grupos subjacentes.

```
typedef OperAux* Group[10];
```

Figura 2 - Estrutura Group.

- ***Oper***

Tipo de dados que contém o nome de uma operação e os valores a si associados (que dizem respeito a cada eixo do sistema de coordenadas e o ângulo). Esta estrutura contém praticamente a mesma informação que a *OperAux*, pelo que poderíamos tê-la utilizado na definição da lista ligada, mas decidimos manter as coisas simples e separadas, de forma a que a sua perceção fosse direta, sem ter qualquer impacto na performance. Por esse motivo, o objetivo desta estrutura é simplesmente armazenar informação relativa a uma determinada operação e suas variáveis, como já foi referido.

```
typedef struct oper {
    char* operation;
    double x, y, z, angle;
} Oper;
```

Figura 3 - Estrutura Oper

- ***OperFile***

Após a definição da estrutura anterior, faltava, então, associar a cada ficheiro .3d as operações a efetuar sobre ele. Assim sendo, criamos a estrutura *OperFile*, que, como o nome indica, associa operações (*vector<Oper*> operations*) a um ficheiro (*char* fileName*). Estas operações são guardadas sobre a forma de um vetor, pois é uma estrutura simples de trabalhar e nunca serão necessárias guardar operações neste vetor ao ponto de ter impacto na sua performance.

```
typedef struct operFile {
    char* fileName;
    vector<Oper*> operations;
} OperFile;
```

Figura 4 - Estrutura OperFile

3. Engine

Nesta fase do projeto, foram introduzidas as operações de transformação associadas a cada objeto a desenhar, pelo que o algoritmo de desenho (função *desenhar*) dos mesmos teve que sofrer alterações, essencialmente pelos factos de estas terem que ser aplicadas antes do seu desenho e as estruturas a partir das quais é retirada essa informação são, agora, diferentes.

O processo de desenho começa pela leitura do vetor das operações (*vector<OperFile*> files)* criado nesta fase e do vetor que contém os pontos dos triângulos a desenhar para cada ficheiro (*vector<Ponto> triangles*), já criado na primeira fase do projeto.

Como estas estruturas são vetores, utilizamos *iterators* para os percorrer, começando por identificar os triângulos a desenhar para cada ficheiro e, de seguida, aplicar as operações a ele associadas aos pontos de cada triângulo, como podemos ver nas alterações feitas aos dois ciclos principais da função *desenhar*, apresentados na figura abaixo.

```
while (it != triangles.end()) {

    glPushMatrix();
    std::vector<Oper*>::iterator it2;
    it2 = op->operations.begin();
    while(it2 != op->operations.end())
    {
        Oper* oper = *it2;
        it2++;
        if (strcmp(oper->operation, "translate")==0)
        {
            glTranslatef(oper->x, oper->y, oper->z);
        }
        if (strcmp(oper->operation, "rotate") == 0)
        {
            glRotatef(oper->angle, oper->x, oper->y, oper->z);
        }
        if (strcmp(oper->operation, "scale") == 0) {
            glScalef(oper->x, oper->y, oper->z) ;
        }
    }
    Ponto aux_1 = *it; it++;
    Ponto aux_2 = *it; it++;
    Ponto aux_3 = *it; it++;

    glBegin(GL_TRIANGLES);
    //glColor3f(0.3, 0.1, 0);
    glColor3f(1, 1, 1);
    glVertex3d(aux_1.x, aux_1.y, aux_1.z);
    glVertex3d(aux_2.x, aux_2.y, aux_2.z);
    glVertex3d(aux_3.x, aux_3.y, aux_3.z);
    glEnd();

    glPopMatrix();
}
```

Figura 5 – Alterações na engine

4. Sistema Solar

Um dos objetivos desta fase do trabalho é implementar uma representação do sistema solar. Para isto foi realizado um ficheiro XML onde todas os componentes do sistema solar são representados através dos pontos guardados num ficheiro chamado sphere.3d. Desta forma a única coisa que varia entres todas as componentes são as translações, rotações e a escala relativa a cada componente.

De seguida encontra-se um excerto do nosso ficheiro XML:

```
<group>
    <!--terra -->
    <translate X="46.22" Y="0" Z="0" />
    <scale X="0.6450" Y="0.6450" Z="0.6450" />
    <models>
        <model file ="sphere.3d" />
    </models>
    <group>
     <!-- lua -->
        <translate X="10.5" Y="10.5" Z="-5"/>
        <scale X="0.1737" Y="0.1737" Z="0.1737" />
        <models>
            <model file = "sphere.3d"/>
        </models>
    </group>
</group>
<group>
```

Figura 6 - excerto do ficheiro XML com o sistema solar.

A estratégia utilizada para gerar a cena demo consiste em começar por gerar o Sol que fica com as dimensões da esfera guardada no ficheiro sphere.3d, já o resto das componentes sofrem a operação escale e ***translates*** sucessivos.

De forma a tirar partido das operações geométricas herdadas em grupos intrínsecos decidimos que todas as componentes pertencem ao grupo do Sol e que para além disso a cada planeta estão intrínsecos os grupos dos seus satélites naturais.

Devido a alguns planetas terem um grande número de satélites naturais optou-se por nesses casos representar apenas os maiores.

A lista de satélites naturais implementados é a seguinte:

- Terra
  - Lua
- Júpiter
  - Ganimedes
  - Calisto
- Saturno
  - Titã
- Urano
  - Titânia
  - Oberon
- Neptuno
  - Tritão

Quanto à escala utilizada começamos por usar a escala 1-1000 milhões de Km com as distâncias reais, mas tivemos de reduzir esta escala para casos como o Sol, Júpiter, Saturno, Úrano e Neptuno e de forma a obter uma melhor representação.

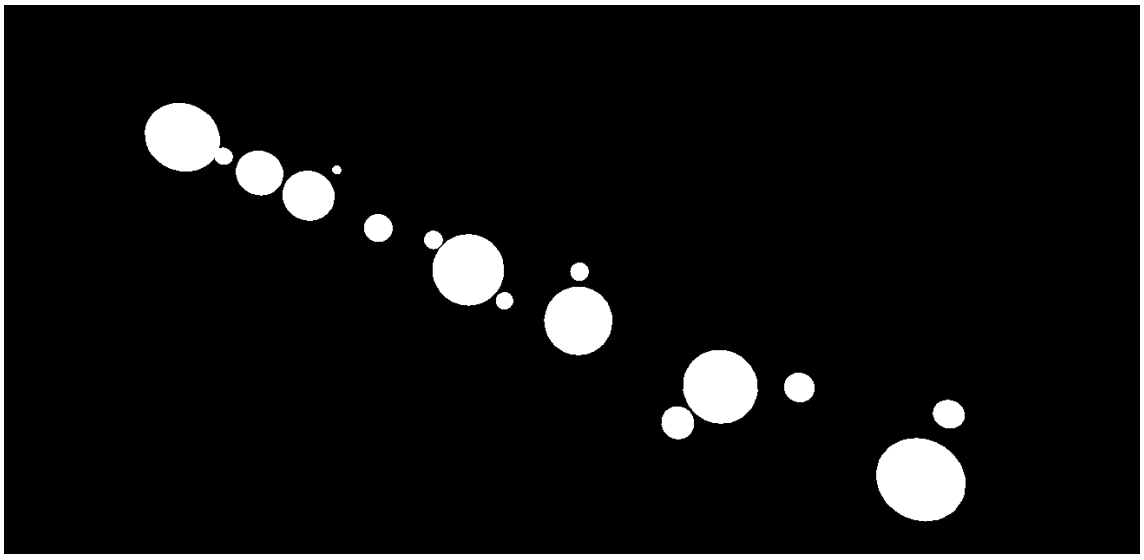


Figura 7 – resultados do desenho do sistema solar.

Conclusão

Esta segunda fase do trabalho prático foi importante, pois conseguimos aplicar os conhecimentos sobre as transformações geométricas adquiridos nas aulas, assim como aprofundar os nossos conhecimentos em *OpenGL* e na linguagem C++, que, de outra forma, não seria possível.

De uma forma geral, consideramos que os objetivos para esta segunda fase foram alcançados com sucesso, pois pensamos ter cumprido com todos os requisitos, embora tenha sido necessária muita pesquisa antes de passarmos à sua implementação propriamente dita, pesquisa essa que esteve na base das decisões tomadas durante todo o processo.

No entanto, como extra, poderiam ter sido adicionadas outras figuras geométricas, tais como o *torus*, de forma a fazer uma melhor recriação do sistema solar.