Universidade do Minho

Computação gráfica

Departamento de Informática

Fase 2

Grupo 40:
João Abreu — A84802
Tiago Magalhães — A84485
Hugo Matias — A85370
Sérgio Gomes — A67645

29 de Março de 2020

Conteúdo

1. Fase 2 - Transformações geométricas

O objetivo da segunda fase do trabalho foi do de melhorar o motor gráfico iniciado na primeira fase. Esta melhoria foi feita através da especificação e consequente interpretação de cenas hierárquicas em XML.

1.1 Generator

1.1.1 Torus

Ao generator que tínhamos da fase 1 apenas tivemos de adicionar a possibilidade de gerar um torus que servirá para representar os anéis dos planetas que os possuem.

Para desenhar o torus são necessários dois raios, o raio que vai do centro do torus até ao meio do tubo e o raio do tubo. São também necessárias stacks(em número considerável) para darem a forma redonda ao torus, e slices para darem volume ao mesmo.

De modo a desenvolver o processo de construção do torus foram implementados dois ciclos, um que percorre o número de stacks e outro, inserido no primeiro, que percorre os slices. O ciclo de slices vai desenhando os lados do torus recorrendo à construção de quadrados(por sua vez recorrem a triângulos), sendo por isso necessários 4 pontos para construir tais quadrados tanto do lado esquerdo como do direito. Quanto maior o número de slices maior será o "volume"do torus. O ciclo de stacks vai desenhando a forma redonda do torus recorrendo também a construção de quadrados sendo assim necessários 4 pontos para tal feito. Quanto maior o número de stacks, mais redondo irá parecer o torus.

Note-se que o número de stacks e slices não deve ser muito pequeno, assim como o raio do tubo não deve ser maior que o raio do centro até ao meio do tubo (modelo deixa de ser um torus).

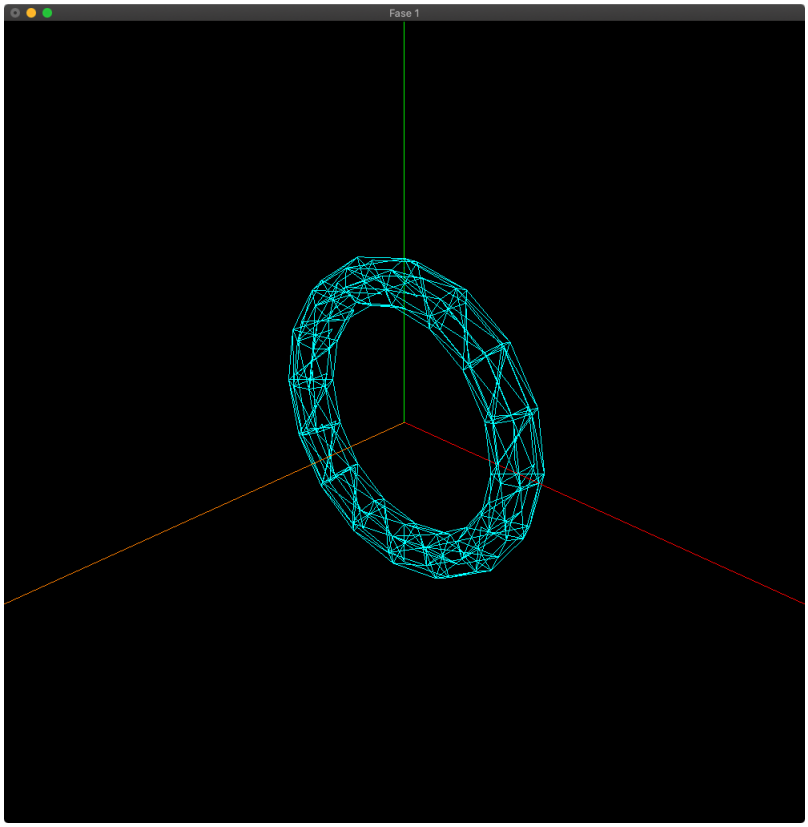

Figura 1.1: Torus.

1.2 Engine

Tal como na última fase, começamos por chamar uma função que vai tentar abrir o ficheiro argumento em formato XML, bem como identificar o primeiro grupo, esta tag grupo será fundamental nesta fase uma vez que é dentro dela que estão as transformações aplicar, bem como as primitivas a desenhar e grupos que irão herdar características dos anteriores, assim, esta estrutura grupo tornar-se-á fundamental no desenvolvimento desta fase.

```
    void readXML(string file){

    XMLDocument doc;
    if(!doc.LoadFile(file.c_str())){
        cout << "Ficheiro XML lido" << endl;
        XMLElement * root = doc.RootElement();  //obtem a tag scene
        XMLElement * group = root->FirstChildElement(); //obtem o primeiro grupo
        parseGroup(g,group); /* chamada da função parser com g (variável global Grupo)
                             e com o primeiro grupo do ficheiro */
    } else {
        cout << "Ficheiro XML não lido" << endl;
    }
}
```

Tivemos de criar um parser de ficheiros XML (chamado na função acima) e que este fosse capaz de guardar cada valor lido numa estrutura de dados, sendo a assinatura do método a seguinte:

```
void parseGroup(Grupo &grupo, XMLElement *group);
```

Para o parsing é utilizada a ferramenta *tinyXml2*,este parser vai identificar tag's xml,e verá se correspondem a uma rotação,translação, etc., aplicando a criação de um objeto translação por exemplo se a tag for translação e retira do XML os atributos necessários para o criar, fará o mesmo para a rotação e a cor, no caso de encontrar um grupo, este será filho pois a função do parser vai iterando pelos grupos principais, após encontrar um grupo filho irá aplicar recursividade e chamar-se a si própria, guardando estes grupos filhos no array childgroups da classe grupo, esta classe encontra-se abaixo explicada mais detalhadamente.

1.2.1 Grupo

Nesta fase, criamos uma classe chamada Grupo que se trata de uma estrutura de dados onde guardamos toda a informação de um grupo(<group>)que fomos carregando ao ler o ficheiro XML. É constituída pelas seguintes variáveis:

```
vector<Transformacao*> transformations;
vector<vector<Ponto>> models;
vector<Grupo> childgroups;
```

Um grupo terá, assim guardado os models a desenhar, as transformações a aplicar e os grupos filhos.

A definição da classe Transformacao está presente na sub-secção 1.2.2 enquanto que Ponto não passa de uma struct cujos campos representam os valores tridimensionais de um ponto. Esta classe também é a responsável por desenhar utilizando as ferramentas GLUT e acedendo diretamente às suas variáveis de instância. Assim a função que irá desenhar é a seguinte:

```
void Grupo :: drawGroup(){
    glPushMatrix();
    for (auto &transformation : transformations) transformation->transform();
    for (auto const &model : models) draw(model);
    for (auto &child : childgroups) child.drawGroup();
    glPopMatrix();
}
```

Primeiro começamos por aplicar as transformações,deste modo permite que os grupos filhos herdam as transformações,de seguida percorremos os models para os desenhar já com as transformações aplicadas anteriormente e,por fim percorremos os grupos filhos que por recursividade irão voltar a repetir o processo. Neste processo onde irão ser aplicadas transformações geométricas,irão ocorrer alterações na matriz de transformação,por isso devemos guardar o estado inicial,tal como após outras transformações o estado deve voltar ao ínicio,para isto são usadas a funções do GLUT : **glPushMatrix()** e **glPopMatrix()** .

1.2.2 Transformacao

A classe Transformacao serve de super classe para as classes Cor, Escala, Rotação e Translação (que podem ser encontrados nas sub-secções 1.2.3, 1.2.4, 1.2.5 e 1.2.6, respetivamente) e contém apenas a assinatura do seguinte método abstrato:

```
virtual void transform() = 0;
```

Todas as subclasses têm este método definido de modo a aplicar a sua função GLUT correspondente.

1.2.3 Cor

Nesta subclasse são definidas as cores a serem usadas na construção do modelo do sistema solar, usando a função da superclasse Transformacao e também a função já existente glColor3f(r,g,b). No ficheiro XML usamos a cor no formato RGB sendo necessário a conversão para formato percentual para a chamada da função do GLUT.

1.2.4 Escala

Aqui é definida a escala do modelo em questão, usando a função da superclasse Transformacao e também a função já existente glScale3f(x,y,z).

1.2.5 Rotação

Aqui é definida a rotação a ser aplicada na construção dos planetas modelo de modo a que estes fiquem em posições angulares normais.Foram usadas a função da superclasse Transformacao e a função já existente glRotate3f(a,x,y,z)

1.2.6 Translação

A definição da translação a ser aplicada na construção do modelo de modo a que estes fiquem em posições relativas naturais. Foram usadas a função da superclasse Transformacao e a função já existente glTranslate3f(x,y,z).

1.3 Cena de demonstração

Para a segunda fase do trabalho prático foi-nos proposta a elaboração de uma demo scene estática do modelo do sistema solar, que contém o sol, os planetas e respectivas luas.

De modo a reaproveitarmos o trabalho realizado na primeira fase, o ficheiro com o modelo da esfera foi utilizado para a representação de todos os planetas do sistema solar. Assim, a esfera utilizado tem raio 1 para tornar as escalas e distâncias mais fáceis de calcular.

O sol têm um subgrupo onde os planetas estao incluidos, assim como cada planeta tem um subgrupo onde se encontram as suas respetivas luas. Foi decidido não desenhar todas as luas pois alguns dos planetas, como Júpiter e Urano, possuem uma grande quantidade de luas, então decidimos, pelo menos por agora, não as representar todas.

Para as distâncias também não foram usadas as verdadeiras pois, mesmo reduzindo as escalas, os planetas continuavam a ficar muito distantes uns dos outros, pelo que optamos por valores que melhor se enquadravam no modelo final.

Para o cálculo das escalas dos planetas usamos o site https://www.1728.org/solarsys.htm.

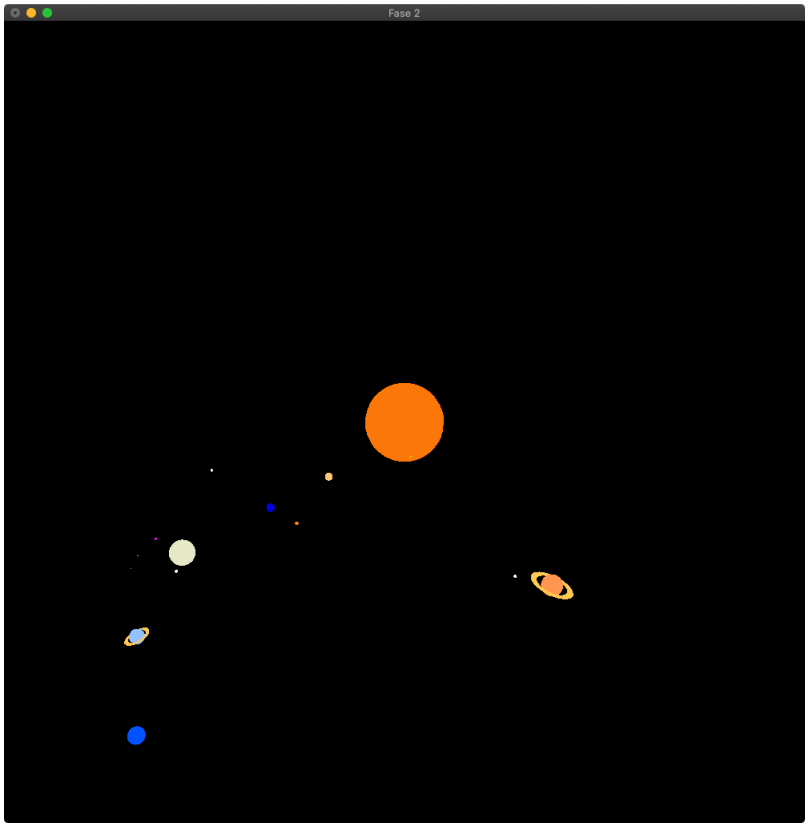

Figura 1.2: Demonstração do Sistema Solar.

1.4 Alterações importantes feitas desde a última fase

Descobrimos que o engine já não necessita do full path para ler o ficheiro XML pois este é corrido dentro da pasta build/Debug/. O mesmo acontecia para o ficheiro XML que precisava do full path para dizer ao programa onde estavam os ficheiros .3d. Sendo assim, em vez do generator criar ficheiros no local onde este corre, passámos a criá-los diretamente na pasta build/Debug/ tal como a localização do ficheiro XML.