Universidade do Minho
Escola de Engenharia

COMPUTAÇÃO GRÁFICA

FASE 1

Alexandre Martins A77523

André Vieira A78322

Eduardo Rocha A77048

Ricardo Neves A78764

ÍNDICE

Introdução 2
Generator 3
Plano 3
Caixa 4
Esfera 5
Cone 6
Figura Adicional – Cilindro 7
XML 9
Extração das Coordenadas 10
OpenGL 10
Resultados obtidos 11
Conclusão e Trabalho Futuro 12

INTRODUÇÃO

Neste relatório iremos apresentar e discutir o trabalho realizado pelo grupo, no âmbito da Unidade Curricular de Computação Gráfica, do 3º ano do Mestrado Integrado em Engenharia Informática.

Nesta primeira fase do trabalho prático, tivemos como objetivo a geração de modelos em 3 dimensões, tendo em conta um ficheiro de configuração, escrito em XML.

Portanto, neste relatório, iremos apresentar detalhadamente todo o processo realizado, de modo a cumprir os objetivos previamente estabelecidos pelo docente da Unidade Curricular.

Para isto, apresentamos também neste documento, algumas linhas de pensamento que o grupo seguiu, ferramentas usadas, e excertos de código fonte utilizado (e respetiva explicação), de modo a suportar a perceção do trabalho realizado.

Esta primeira fase encontra-se dividida em duas partes principais. A primeira prende-se à criação de um gerador de ficheiros enquanto que a segunda, por sua vez, pressupõe a elaboração de um motor que permita ler um ficheiro de configuração escrito em XML.

Deste modo, o produto final terá de ser o desenho dos modelos gerados anteriormente pelo grupo.

GENERATOR

Para ir de encontro ao proposto pelo enunciado, elaboramos as primitivas gráficas propostas recorrendo às posições relativas dos vértices dos diferentes modelos. Aqui é onde são gerados os pontos que, unidos em triângulos, formam as figuras geométricas pedidas.

Plano

O plano foi a figura geométrica mais fácil de implementar, uma vez que é apenas a junção de dois triângulos simples.

Para a sua geração, apenas é necessário referenciar a largura do plano quadrado, sendo que, com isto, são desenhados os dois triângulos constituintes.

```
file << "" << (-length/2) << " 0 " << (-length/2) << "\n";
file << "" << (-length/2) << " 0 " << (length/2) << "\n";
file << "" << (length/2) << " 0 " << (length/2) << "\n";

file << "" << (-length/2) << " 0 " << (-length/2) << "\n";
file << "" << (length/2) << " 0 " << (length/2) << "\n";
file << "" << (length/2) << " 0 " << (-length/2) << "\n";
```

Figura 1 - Código do plano

Este é o resultado do plano obtido. Aqui, confirma-se que são usados apenas dois triângulos, unidos pelas suas hipotenusas.

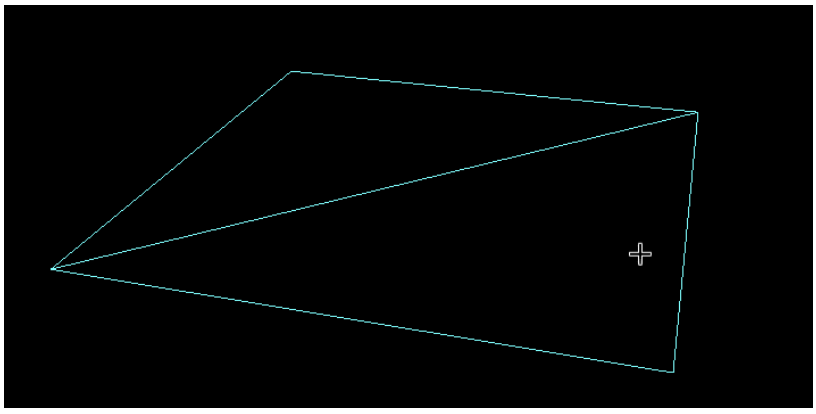

Figura 2 - Desenho do plano

Caixa

A segunda figura geométrica desenhada foi a caixa, sendo que é necessário especificar o seu comprimento, altura e largura e, opcionalmente, o número de divisões.

Para o seu desenho, a melhor forma encontrada foi gerar uma face de cada vez.

```
x = length/2;
z = width/2;
for (int i = 0; i < div; i++) {
    y = height;
    for (int j = 0; j < div; j++) {
        file << "" << x << " " << y << " " << z << "\n";
        file << "" << x << " " << (y - divY) << " " << z << "\n";
        file << "" << x << " " << (y - divY) << " " << (z - divZ) << "\n";

        file << "" << x << " " << y << " " << z << "\n";
        file << "" << x << " " << (y - divY) << " " << (z - divZ) << "\n";
        file << "" << x << " " << y << " " << (z - divZ) << "\n";

        y -= divY;
    }
    z -= divZ;
}
```

Figura 3 - Código da caixa

Assim, o gerador percorre os ciclos das divisões especificadas, desenhado todos os triângulos que compões essa face.

Este é o resultado final para a caixa, sendo que o grupo especificou um número de divisões igual a 5, como se pode constatar.

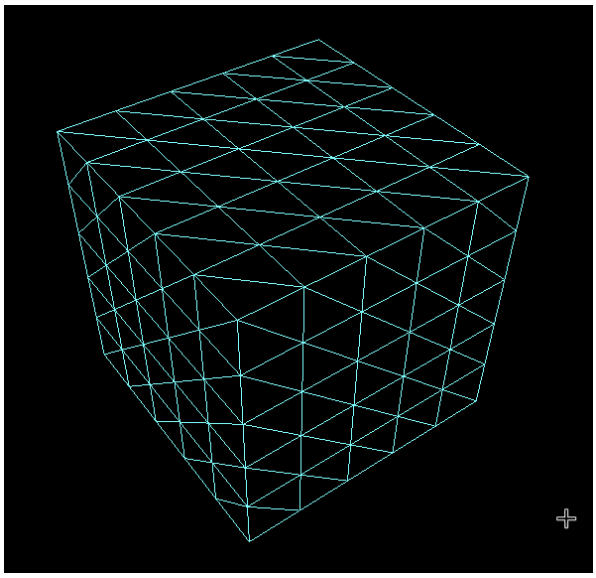

Figura 4 - Desenho da caixa

Esfera

Para a construção da esfera, temos de especificar primeiramente o raio da mesma, o número de *slices* e o número de *stacks*. Quanto maior o número destes dois últimos parâmetros, mais "redonda" ficará a esfera resultante.

Assim, percorrendo o número de *stacks* e de *slices*, e recorrendo às coordenadas polares, desenha-se as coordenadas correspondentes aos vértices dos triângulos.

Desta maneira, os pontos de todos os triângulos vão sendo obtidos à medida que o ângulo vai alterando.

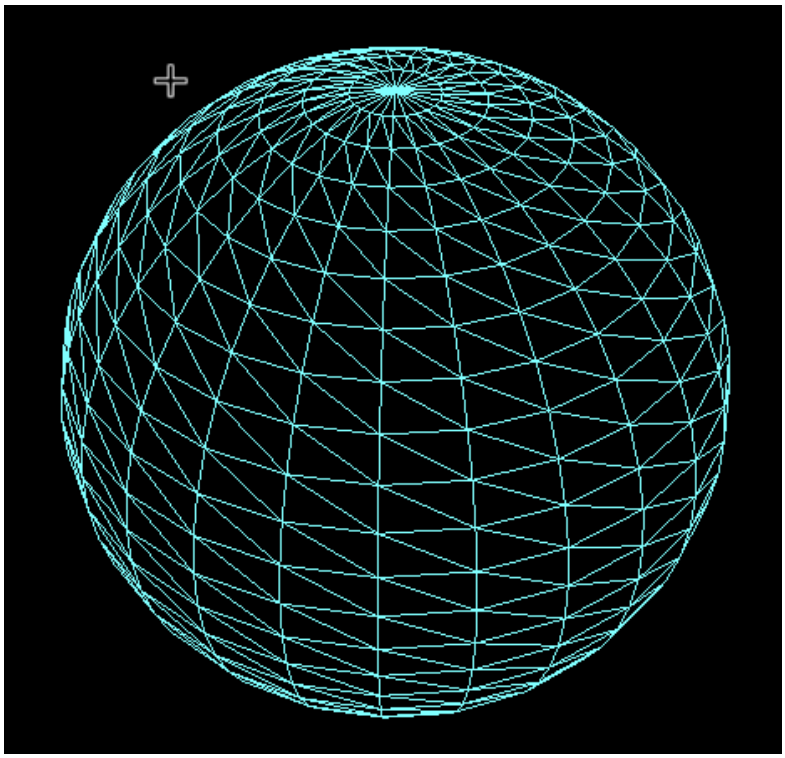

Figura 5 - Desenho da esfera

Cone

Para o desenho do cone, constatamos que podíamos utilizar novamente as coordenadas polares, uma vez que a sua base é uma circunferência. De salientar que os parâmetros necessários para a geração do cone são o raio da sua base, a sua altura e o número de *slices*.

```
file << "" << "0" << " 0 " << "0" << "\n";
file << "" << (radius * sin(alpha + delta)) << " 0 " << (radius * cos(alpha + delta)) << "\n";
file << "" << (radius * sin(alpha)) << " 0 " << (radius * cos(alpha)) << "\n";

file << "" << "0 " << height << " 0" << "\n";
file << "" << (radius * sin(alpha)) << " 0 " << (radius * cos(alpha)) << "\n";
file << "" << (radius * sin(alpha + delta)) << " 0 " << (radius * cos(alpha + delta)) << "\n";

alpha += delta;
```

Figura 6 - Código do cone

Assim, por cada iteração do ciclo *for*, é desenhada uma parte da circunferência base e uma das laterais.

O resultado final do cone é o seguinte:

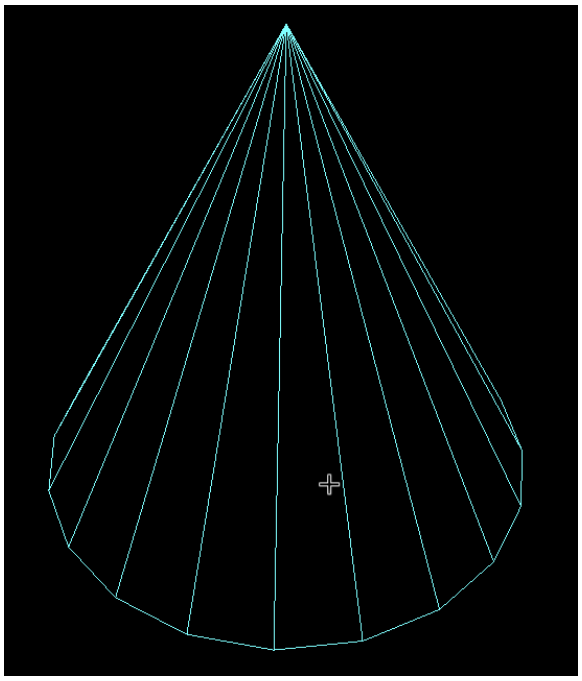

Figura 7 - Desenho do cone

Figura Adicional – Cilindro

Para além das figuras geométricas pedidas no enunciado, tomamos a iniciativa de criar uma outra figura a 3 dimensões. Assim, o grupo decidiu criar um cilindro.

O método de criação do cilindro foi muito idêntico ao da figura anterior, o cone. Isto deve-se ao facto de ambas as suas bases serem círculos, o que permite o uso de coordenadas polares.

```
for (int i = 0; i < slices; i++) {
    file << "0 0 0" << "\n";
    file << (r * sin(alpha + delta)) << " 0 " << (r * cos(alpha + delta)) << "\n";
    file << (r * sin(alpha)) << " 0 " << (r * cos(alpha)) << "\n";

    file << (r * sin(alpha + delta)) << " " << height << " " << (r * cos(alpha + delta)) << "\n";
    file << (r * sin(alpha)) << " " << height << " " << (r * cos(alpha)) << "\n";
    file << (r * sin(alpha)) << " 0 " << (r * cos(alpha)) << "\n";

    file << (r * sin(alpha + delta)) << " " << height << " " << (r * cos(alpha + delta)) << "\n";
    file << (r * sin(alpha)) << " 0 " << (r * cos(alpha)) << "\n";
    file << (r * sin(alpha + delta)) << " 0 " << (r * cos(alpha + delta)) << "\n";

    file << "0 " << height << " 0" << "\n";
    file << (r * sin(alpha)) << " " << height << " " << (r * cos(alpha)) << "\n";
    file << (r * sin(alpha + delta)) << " " << height << " " << (r * cos(alpha + delta)) << "\n";
    alpha += delta;
}
```

Figura 8 - Código do cilindro

Para cada iteração do ciclo, desenha-se uma parte de cada uma das bases do cilindro (base e topo) e os dois triângulos que formam uma parte da sua lateral.

Aqui vemos o resultado do cilindro final:

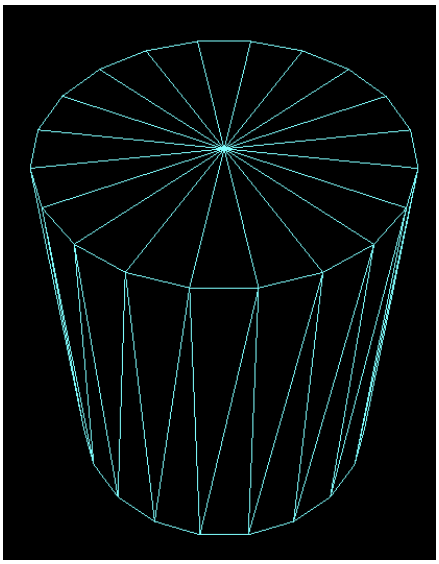

Figura 9 - Desenho do cilindro

Na Main do Generator especificado até aqui, é onde o utilizador poderá escolher a figura geométrica a desenhar através da sua linha de comandos. Aqui, são recolhidos os parâmetros necessários, usados nas invocações das diversas funções.

```
// Sphere(radius, slices, stacks, filename)
if (strcmp(argv[1], "sphere") == 0)
    sphere(argv[2], argv[3], argv[4], argv[5]);
```

Figura 10 - Exemplo da Main

No exemplo do código acima, podemos observar que cada função de geração está associada a um sólido. Se o primeiro argumento dado pelo utilizador for "sphere", então é chamada a função "generateSphere" com os parâmetros necessários para a criação do mesmo.

```
>> ./generator sphere 7 20 20 sphere.3d
```

Figura 11 - Invocação no terminal

Pelo exemplo acima, podemos ver que o utilizador tem a intenção de criar uma esfera de raio igual a 7, sendo que é construída com 20 *slices* e 20 *stacks*. As coordenadas dos pontos gerados são guardadas no ficheiro sphere.3d, que irá ser analisado mais à frente.

XML

O *parser* utilizado para ler o ficheiro XML disponibilizado foi o tinyXML2, como aconselhado no enunciado do trabalho prático.

O *tinyXML2* é uma pequena e simples biblioteca *open source* de análise sintática de XML para a linguagem C++, que funciona em múltiplos sistemas operativos.

```
<scene>
    <model file="plane.3d" />
</scene>
```

Figura 12 - Exemplo do XML

Depois de efetuadas as alterações necessárias ao *parser* para a leitura do ficheiro, os dados foram guardados numa estrutura de dados, para acesso futuro. Aqui, foram guardados os nomes dos ficheiros a que é necessário aceder (por exemplo, "plane.3d"), onde se encontram as várias coordenadas dos pontos, que, interligados em forma de triângulos, formam uma das figuras disponíveis.

De salientar que um ficheiro XML pode conter uma ou mais figuras a desenhar.

O nosso *parser* encontra a ocorrência do atributo "file", guardando o nome do ficheiro no array "files", retornando o mesmo no final.

EXTRAÇÃO DAS COORDENADAS

A extração das coordenadas é efetuada na função "extractor". Esta permite extrair as coordenadas de um ficheiro *.3d* e armazenar em vetores (tuplos de três coordenadas).

OPENGL

O *OpenGL* , lançado em janeiro de 1992 é um *software* gratuito utilizado na computação gráfica, com o intuito de desenvolver aplicativos gráficos, objetos em 3 dimensões, jogos, entre outras funcionalidades. É uma ferramenta semelhante ao Direct3D ou Glide.

OpenGL tem em vista a utilização de linguagens de programação como o C ou C++. No entanto, pode ser utilizada com diversas outras linguagens, apresentando um alto nível de eficiência.

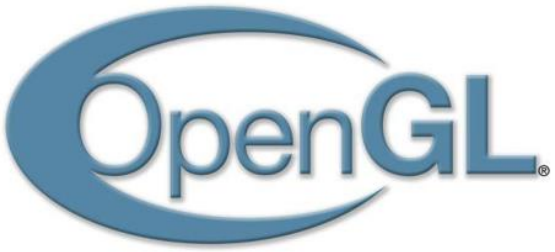

Figura 13 - Logotipo do OpenGL

Como já foi referido, temos um vetor preenchido com todas as coordenadas dos pontos a desenhar. Assim, é necessário agrupar os pontos em grupos de 3, de modo a criar triângulos. Todos estes triângulos ligam-se entre si, criando uma figura geométrica em 3 dimensões.

Como se pode observar no código fonte abaixo, são percorridos 3 pontos do vetor "triangles", sendo que este procedimento ocorre até ao seu fim.

```
void drawTriangles(void)
{
    vector<coords>::iterator i;
    i = triangles.begin();

    while (i != triangles.end()) {

        coords point1 = *i;
        i++;
        coords point2 = *i;
        i++;
        coords point3 = *i;
        i++;

        glBegin(GL_TRIANGLES);
            glColor3f(red, green, blue);
            glVertex3d(get<0>(point1), get<1>(point1), get<2>(point1));
            glVertex3d(get<0>(point2), get<1>(point2), get<2>(point2));
            glVertex3d(get<0>(point3), get<1>(point3), get<2>(point3));
        glEnd();
    }
}
```

Figura 14 - Código do desenho dos triângulos

Deste modo, obtemos uma (ou várias) figura 3D.

RESULTADOS OBTIDOS

Ao longo do relatório, fomos mostrando figuras isoladas, onde o ficheiro XML apenas contém uma entrada.

Assim, decidimos testar um par de ficheiros XML que contivesse mais do que uma figura geométrica, chegando a estes resultados.

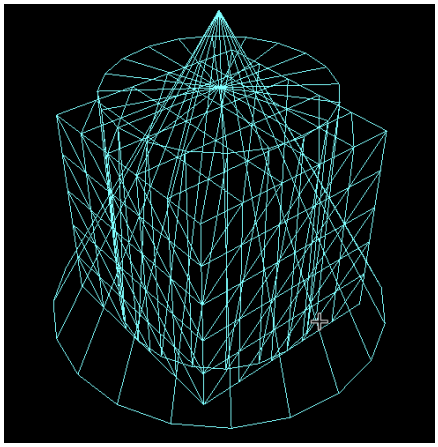

Figura 15 - Resultado de um cone, caixa e cilindro

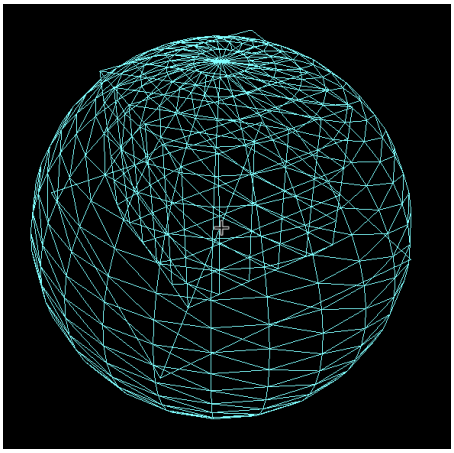

Figura 16 - Resultado de um plano, uma caixa e uma esfera

CONCLUSÃO E TRABALHO FUTURO

Esta primeira fase do TP de Computação Gráfica permitiu um aprofundamento dos conhecimentos relativos à linguagem C++ e ao uso da ferramenta *OpenGL*, sendo que o objetivo principal foi cumprido, uma vez que todos os sólidos propostos foram desenhados, acrescentando uma nova figura extra. Permitiu também perceber bem melhor como são implementados alguns dos sólidos que a ferramenta *OpenGL* já contém predefinidos.

Para a próxima fase do trabalho, de forma geral, iremos criar cenas envolvendo transformações geométricas, como a translação, a rotação e a escala.

Assim, com este trabalho realizado até aqui, sentimo-nos capazes de continuar este projeto, sendo que as bases já foram construídas e consolidadas.