TP1-CG-MIEI

João Bernardo Freitas a74814

3 Março 2020

Contents

1 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 fase tivemos como objectivo gerar diversos modelos em 3 dimensões através de um gerador, que cria os ficheiros .3d que contêm as coordenadas dos modelos e um ficheiro de configuração, escrito em XML, que indica os ficheiros a serem lidos, bem como um motor que desenha esses modelos através dos ficheiros previamente lidos.
Esta primeira fase encontra-se dividida em duas partes principais. O desenvolvimento do gerador de ficheiros e o desenvolvimento do motor.

2 Generator

Nesta parte é necessário criar os ficheiros .3d de acordo com o input do utilizador. Para além disso é também necessário a criação de um ficheiro **config.xml** que indica quais os ficheiros a ler.
Os modelos em si são definidos por triângulos, sendo que cada ponto desse triângulo tem 3 coordenadas **px,py,pz**.

2.1 Plano

O plano foi a figura mais fácil de implementar, visto que é uma junção simples de dois triângulos. O tamanho do plano é dado pelo utilizador e o plano está centrado na origem.
Como tal sabemos que **py=0** e **px/pz** irão variar entre **0,comprimento/2 e -comprimento/2**.

```
void plano(char* c, char* f){
    double l = atof(c);
    ofstream ficheiro;
    ficheiro.open(f, ios::app);

    if (ficheiro.is_open()) {
        ficheiro << "0" << " 0 " << -(l/2) << "\n";
        ficheiro << "0" << " 0 " << l/2 << "\n";
        ficheiro << l/2 << " 0 " << "0" << "\n";

        ficheiro << "0" << " 0 " << l/2 << "\n";
        ficheiro << "0" << " 0 " << -(l/2) << "\n";
        ficheiro << -(l/2) << " 0 " << "0" << "\n";
    }
    ficheiro.close();

    config(f);
}
```

Figure 1: Algoritmo de desenho do plano

Para a criação do plano utilizamos o comando **./generator plane *comprimento ficheiro*.3d**, que irá criar um ficheiro plano.3d que contém as coordenadas do plano e o ficheiro config.xml que irá conter o nome do ficheiro que o motor deve carregar, se o ficheiro config.xml já existir então acrescenta o novo ficheiro.

```
joaob@PC-JB:/mnt/c/Users/joaob/Desktop/TP/project$ ./generator plane 6 plano.3d
joaob@PC-JB:/mnt/c/Users/joaob/Desktop/TP/project$ cat config.xml
<scene>
<model file="plano.3d"/>
</scene>
joaob@PC-JB:/mnt/c/Users/joaob/Desktop/TP/project$ cat plano.3d
0 0 -3
0 0 3
3 0 0
0 0 3
0 0 -3
-3 0 0
```

Figure 2: Criação do plano e conteúdo dos ficheiros

Obtendo a seguinte figura.

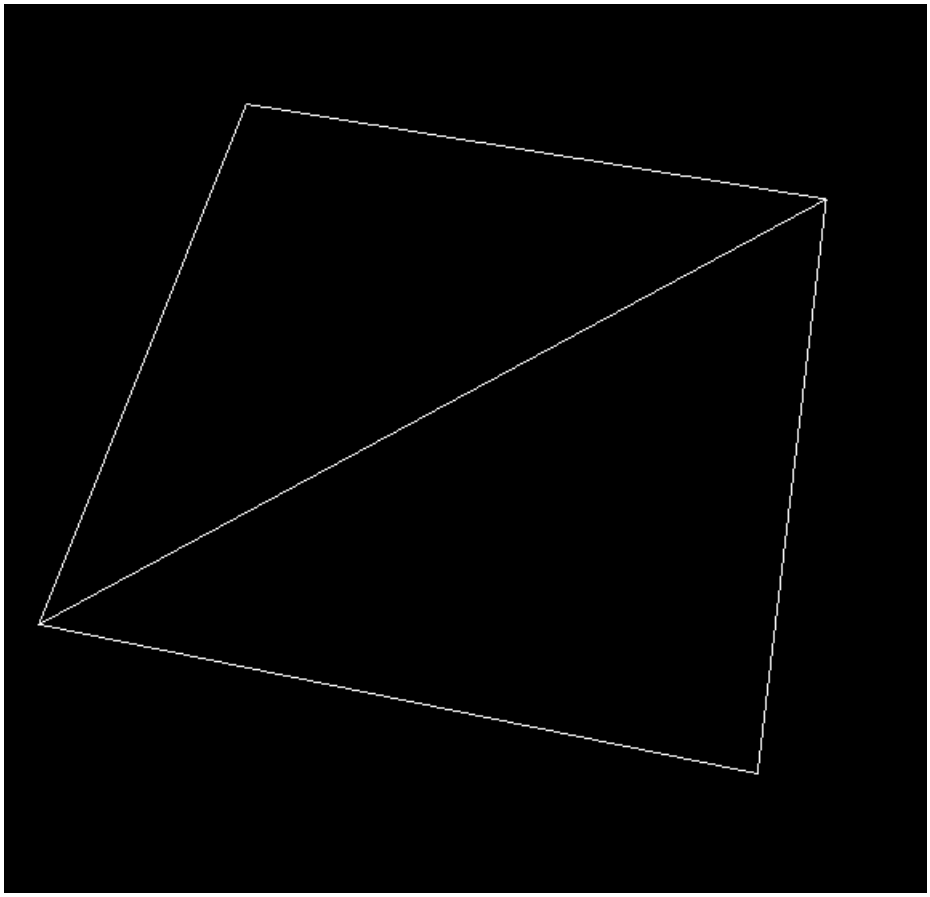

Figure 3: Resultado do plano

2.2 Caixa

A segunda figura a desenhar foi a caixa, sendo que o utilizador tem de especificar o comprimento, altura, largura e o número de divisões, podendo este último ser igual a zero.
Uma caixa é definida através de, no mínimo, 6 planos, sendo que em cada plano só irá haver uma variação em duas das suas coordenadas.

- Face Superior - px e pz variam, py fica sempre igual á altura.
- Face Inferior - px e pz variam, py fica sempre igual a zero.
- Face Frontal - py e pz variam, px fica sempre igual ao comprimento.
- Face Posterior- py e pz variam, px fica sempre igual a zero.
- Face Esquerda - px e py variam, pz fica sempre igual á largura.
- Face Direita - px e py variam, pz fica sempre igual a zero.

Se o utilizador especificar 1 ou mais divisões será necessário dividir a altura/comprimento/largura de forma a que cada face da caixa seja composta por diversos planos de igual tamanho. Como tal decidimos utilizar ciclos para desenhar os diversos planos que definem uma caixa.

```
float c = comp / div;
float a = altura / div;
float l = larg / div;
float cini = c;
float lini = l;
float aini = a;
for (c; c <= comp; c += cini) {
    for (l; l <= larg; l += lini) {
        for (a; a <= altura; a += aini) {
            //face superior
            ficheiro << c << " " << altura << " " << l << "\n";
            ficheiro << c-cini << " " << altura << " " << l-lini << "\n";
            ficheiro << c-cini << " " << altura << " " << l << "\n";
            ficheiro << c << " " << altura << " " << l-lini << "\n";
            ficheiro << c-cini << " " << altura << " " << l-lini << "\n";
            ficheiro << c << " " << altura << " " << l << "\n";
            //face inferior
            ficheiro << c << " 0 " << l << "\n";
            ficheiro << c-cini << " 0 " << l << "\n";
            ficheiro << c-cini << " 0 " << l-lini << "\n";
            ficheiro << c << " 0 " << l << "\n";
            ficheiro << c-cini << " 0 " << l-lini << "\n";
            ficheiro << c << " 0 " << l-lini << "\n";
            //frente
            ficheiro << comp << " " << a-aini << " " << l << "\n";
            ficheiro << comp << " " << a - aini << " " << l-lini << "\n";
            ficheiro << comp << " " << a << " " << l << "\n";
            ficheiro << comp << " " << a - aini << " " << l-lini << "\n";
            ficheiro << comp << " " << a << " " << l-lini << "\n";
            ficheiro << comp << " " << a << " " << l << "\n";
            //verso
            ficheiro << "0 " << a << " " << l << "\n";
            ficheiro << "0 " << a-aini << " " << l-lini << "\n";
            ficheiro << "0 " << a-aini << " " << l << "\n";
            ficheiro << "0 " << a << " " << l-lini << "\n";
            ficheiro << "0 " << a-aini << " " << l-lini << "\n";
            ficheiro << "0 " << a << " " << l << "\n";
            //esquerda
            ficheiro << c-cini << " " << a-aini << " " << larg << "\n";
            ficheiro << c << " " << a - aini << " " << larg << "\n";
            ficheiro << c << " " << a << " " << larg << "\n";
            ficheiro << c - cini << " " << a << " " << larg << "\n";
            ficheiro << c - cini << " " << a - aini << " " << larg << "\n";
            ficheiro << c << " " << a << " " << larg << "\n";
            //direita
            ficheiro << c - cini << " " << a - aini << " 0\n";
            ficheiro << c - cini << " " << a << " 0\n";
            ficheiro << c << " " << a << " 0\n";
            ficheiro << c - cini << " " << a - aini << " 0\n";
            ficheiro << c << " " << a << " 0\n";
            ficheiro << c << " " << a - aini << " 0\n";
        }
        a = aini;
    }
    l = lini;
```

Figure 4: Algoritmo de desenho da caixa

Para a criação da caixa utilizamos o comando **./generator box** ***comprimento altura largura divisões ficheiro*.3d**.

```
./generator box 6 6 6 2 caixa.3d
```

Figure 5: Criação da caixa

Obtendo a seguinte figura.

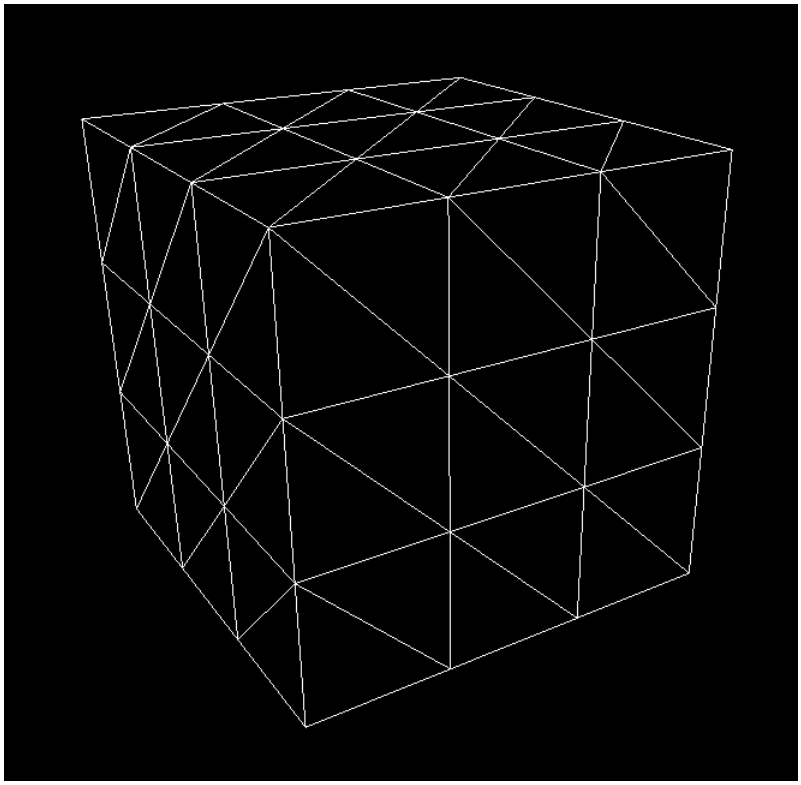

Figure 6: Resultado da caixa

2.3 Cone

Para o desenho do cone o utilizador necessita de especificar o raio, altura e partes sendo que quanto maior for o número de partes mais redondo fica o cone. O algoritmo utilizado para desenhar o cone é semelhante ao do cilindro, feito nas aulas práticas, sendo que as únicas diferenças estão no desaparecimento da superfície superior e de metade dos triângulos que definiam as faces do cilindro, para além disso o resto dos triângulos que definem as faces vão ter um ponto em comum **0,*altura*,0**.

```
//circulo inferior
for (int a = 0; a < 360; a += 360 / p) {
    double h1 = a * pi / 180;
    int n = a + 360 / p;
    double h2 = n * pi / 180;

    ficheiro << sin(h1) * r << " 0 " << cos(h1) * r << "\n";
    ficheiro << "0" << " 0 " << "0" << "\n";
    ficheiro << sin(h2) * r << " 0 " << cos(h2) * r << "\n";
}
//faces
for (int a = 0; a < 360; a += 360 / p) {
    double h1 = a * pi / 180;
    int n = a + 360 / p;
    double h2 = n * pi / 180;
    ficheiro << "0"<< " " << l << " " << "0" << "\n";
    ficheiro << sin(h1) * r << " 0 " << cos(h1) * r << "\n";
    ficheiro << sin(h2) * r << " 0 " << cos(h2) * r << "\n";
}
```

Figure 7: Algoritmo de desenho do cone

Para a criação de um cone utilizamos o seguinte comando **./generator cone *raio altura partes ficheiro*.3d**.

```
./generator cone 1 4 10 cone.3d
```

Figure 8: Criação do cone

Obtendo a seguinte figura.

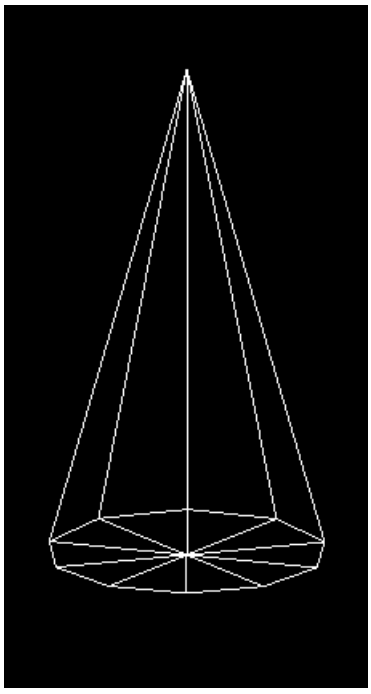

Figure 9: Resultado do cone

2.4 Esfera

A última figura geométrica que tinhamos de desenhar é a esfera, que é desenhada de acordo com o raio, **stacks**, cortes horizontais, e **slices**, cortes verticais, definido pelo utilizador.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.

```
for (int i = 0; i < stacks; i++) {
    alfa = 0;
    for (int j = 0; j < slices; j++) {

        ponto1x = raio * sin(beta) * sin(alfa);
        ponto1y = raio * cos(beta);
        ponto1z = raio * sin(beta) * cos(alfa);

        ponto2x = raio * sin(beta + deltaBeta) * sin(alfa);
        ponto2y = raio * cos(beta + deltaBeta);
        ponto2z = raio * sin(beta + deltaBeta) * cos(alfa);

        ponto3x = raio * sin(beta) * sin(alfa + deltaAlfa);
        ponto3y = raio * cos(beta);
        ponto3z = raio * sin(beta) * cos(alfa + deltaAlfa);

        ponto4x = raio * sin(beta + deltaBeta) * sin(alfa + deltaAlfa);
        ponto4y = raio * cos(beta + deltaBeta);
        ponto4z = raio * sin(beta + deltaBeta) * cos(alfa + deltaAlfa);

        ficheiro << ponto1x << " " << ponto1y << " " << ponto1z << "\n";
        ficheiro << ponto4x << " " << ponto4y << " " << ponto4z << "\n";
        ficheiro << ponto3x << " " << ponto3y << " " << ponto3z << "\n";

        ficheiro << ponto1x << " " << ponto1y << " " << ponto1z << "\n";
        ficheiro << ponto2x << " " << ponto2y << " " << ponto2z << "\n";
        ficheiro << ponto4x << " " << ponto4y << " " << ponto4z << "\n";

        alfa += deltaAlfa;
    }
    beta += deltaBeta;
}
```

Figure 10: Algoritmo de desenho da esfera

Para a criação da esfera utilizamos o seguinte comando **./generator sphere *raio slices stacks ficheiro*.3d**.

```
./generator sphere 1 20 20 esfera.3d
```

Figure 11: Criação da esfera

Obtendo a seguinte figura

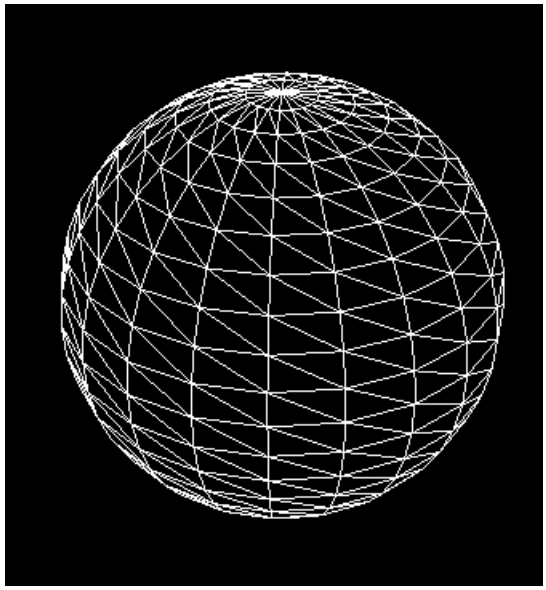

Figure 12: Resultado da esfera

2.5 Composição

Por fim é possível desenhar várias figuras ao mesmo tempo.

```
joaob@PC-JB:/mnt/c/Users/joaob/Desktop/TP/project$ ./generator cone 2 2 10 cone.3d
joaob@PC-JB:/mnt/c/Users/joaob/Desktop/TP/project$ ./generator sphere 1 40 40 esfera.3d
joaob@PC-JB:/mnt/c/Users/joaob/Desktop/TP/project$ ./generator plane 10 plano.3d
joaob@PC-JB:/mnt/c/Users/joaob/Desktop/TP/project$ cat config.xml
<scene>
<model file="cone.3d"/>
<model file="esfera.3d"/>
<model file="plano.3d"/>
</scene>
```

Figure 13: Criação de várias figuras

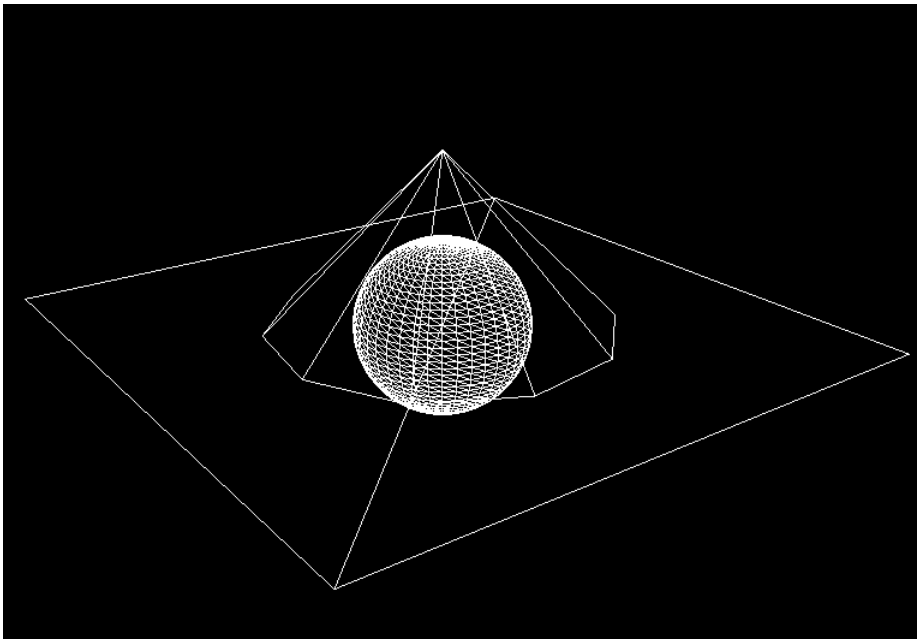

Figure 14: Resultado com várias figuras

3 Engine

Nesta última parte era necessário atingir alguns objetivos, nomeadamente:

- Obter os ficheiros.3d do config.xml
- Obter os vértices dos ficheiros.3d
- Desenhar os triângulos através desses vértices

Para ler o ficheiro config.xml utilizamos a biblioteca **tinyXML2**, que nos permitirá fazer uma análise sintática de **XML** para **C++**.

```
void parser(char* xmlDoc){
    XMLDocument doc;
    XMLError error = doc.LoadFile(xmlDoc);

    int i = 0;

    char** files = (char**)malloc(sizeof(char*) * 15);

    char pre[4] = "../";
    char file[15];

    if (!error) {
        XMLElement* elem = doc.FirstChildElement();
        for (const XMLElement* child = elem->FirstChildElement(); child; i++, child = child->NextSiblingElement()) {

            char* tmp = (char*)child->Attribute("file");
            strcpy(file, tmp);

            char ready[30];
            ready[0] = '\0';

            strcat(ready, pre);
            strcat(ready, file);

            files[i] = (char*)malloc(sizeof(char) * 15);
            strcpy(files[i], ready);

            memset(file, 0, 15);
            memset(ready, 0, 30);
        }
    }
    for (int j = 0; j < i; j++) {
        readFile(files[j]);
    }
}
```

Figure 15: xmlParser

Este *parser* encontra ocorrências do atributo "file" e guarda no array **files** o nome do ficheiro, sendo que esse array é depois percorrido pela função *readFile* que extrai as coordenadas de cada um dos ficheiros para um vector.

```
void readFile(char* fich) {
    std::string ficheiro(fich);
    string linha;
    string novo;
    string delimiter = " ";
    int pos;
    float xx, yy, zz;
    Point p;
    ifstream file;
    file.open(ficheiro);
    if (file.is_open()) {

        while (getline(file, linha)) {

            pos = linha.find(delimiter);
            novo = linha.substr(0, pos);
            xx = atof(novo.c_str());
            linha.erase(0, pos + delimiter.length());
            p.x = xx;

            pos = linha.find(delimiter);
            novo = linha.substr(0, pos);
            yy = atof(novo.c_str());
            linha.erase(0, pos + delimiter.length());
            p.y = yy;

            pos = linha.find(delimiter);
            novo = linha.substr(0, pos);
            zz = atof(novo.c_str());
            linha.erase(0, pos + delimiter.length());
            p.z = zz;

            vertices.push_back(p);
        }
        file.close();
    }
    else {
        cout << "ERRO AO LER FICHEIRO" << endl;
    }
}
```

Figure 16: Leitura dos ficheiros

Por fim temos uma função que percorre todos os vectores e desenha os triângulos.

```
void drawVertices(void){
    glBegin(GL_TRIANGLES);
    glColor3f(1, 1, 1);

    for (size_t i = 0; i < vertices.size(); i++) {
        glVertex3f(vertices[i].x,
            vertices[i].y,
            vertices[i].z);
    }

    glEnd();
}
```

Figure 17: Função de desenho

4 Conclusão

A primeira fase do trabalho prático de Computação Gráfica permitiu um aprofundamento dos conhecimentos da ferramenta **OpenGL** bem como à linguagem **C++**. Visto que todos os sólidos propostos foram criados damos o objetivo principal como cumprido.