Universidade do Minho
Escola de Ciências

# Computação Gráfica

Ano letivo 21/22

- Fase 1- Primitivas Gráficas


Grupo 10:

- Hugo Costeira - A87976
- João Silva - A87939
- João Goulart - A82643
- Pedro Martins - A87964

# Índice

# 1. Introdução

Este trabalho foi destinado à unidade curricular de Computação Gráfica e propôs o desenvolvimento de um motor gráfico 3D que conseguisse demonstrar o seu funcionamento através de alguns exemplos.
O projeto foi dividido em quatro fases, correspondendo este relatório à primeira destas quatro. Nesta fase temos como objetivo a criação de duas aplicações: o *generator* que fornece a informação dos modelos pretendidos e onde estarão contidos os vértices de cada um e a *engine*, cuja função será ler ficheiros XML responsáveis por informar quais modelos serão carregados e, posteriormente, mostrá-los.

# 2. Generator

O Generator é responsável por criar ficheiros 3D que contém os vértices necessários para a criação de cada uma das primitivas gráficas indicadas.

## 2.1. Plane

Para a criação desta figura recorremos à função *plane* que recebe como parâmetros o comprimento, as divisões do plano e ainda o ficheiro onde os vértices do mesmo vão ser escritos.

Inicialmente, criamos 2 variáveis *divSize* e *numVertices* que representarão o tamanho das divisões do plano e o número de vértices necessários para construir o plano. De seguida criamos duas variáveis *var_x* e *var_z* que representam as coordenadas dos vértices nos eixos do x e do z. Como é um plano, este encontra-se paralelo ao plano *xOz* logo a variável y terá sempre valor 0, não sendo assim necessário a criação da sua variável.

Os valores destas variáveis são atribuídos tendo em conta o comprimento do plano dado como argumento à função, sendo atribuído a ambas a metade do comprimento recebido na função *plane*. Após termos estas variáveis representadas, Iniciamos um ciclo for a iterar sobre o número de divisões onde temos uma sequência de impressões no ficheiro destino dos vértices do plano, seguindo a ordem certa tendo em conta a regra da mão direita, de modo que o plano seja visto por cima.

```
void plane(float length, int divisions, char* filename) {
    FILE* f;
    f = fopen(filename, "w");

    float divSize = length / divisions;
    int numVertices = 3 * 2 * pow(divisions, 2.0f);


    if (f) {

        fprintf(f, "%d \n", numVertices * 3);
        int i;
        int j;
        float var_x = -length/2;
        float var_z = -length/2;
        for (i = 0; i < divisions; i++) { //percorrer as "stacks"
            for (j = 0; j < divisions; j++) { //percorrer as "slices"
                fprintf(f, "%f %f %f \n", var_x, 0.0f, var_z);
                fprintf(f, "%f %f %f \n", var_x, 0.0f, var_z + divSize);
                fprintf(f, "%f %f %f \n", var_x + divSize, 0.0f, var_z + divSize);

                fprintf(f, "%f %f %f \n", var_x, 0.0f, var_z);
                fprintf(f, "%f %f %f \n", var_x + divSize, 0.0f, var_z + divSize);
                fprintf(f, "%f %f %f \n", var_x + divSize, 0.0f, var_z);


                var_x += divSize;
            }
            var_z += divSize;
            var_x = -length/2;
        }
       printf("Ficheiro .3d criado com sucesso.\n");
    }

    else {
        printf("Erro ao criar o ficheiro.\nTente novamente.\n");
    }
    fclose(f);
```

Figura 1: Função que gera o Plano

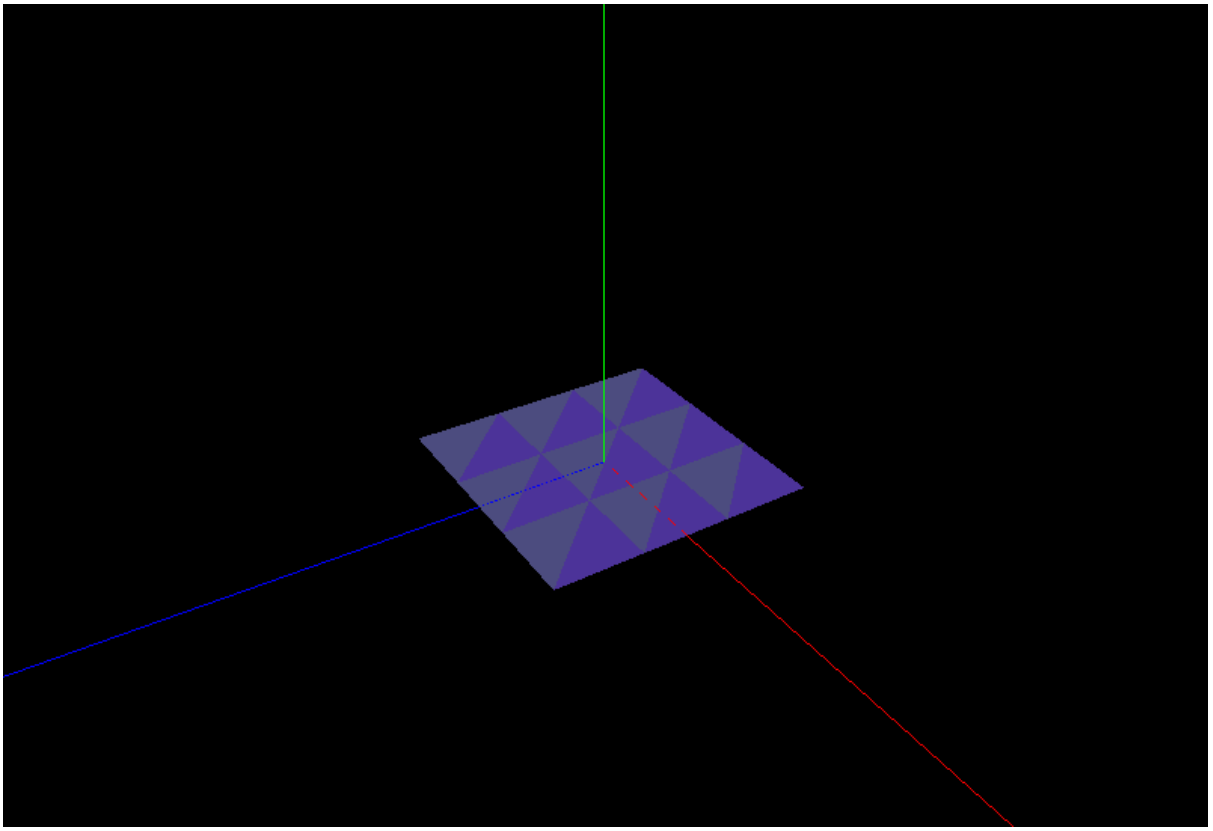

Figura 2: Plano com comprimento de lado 1 e 3x3 divisões

## 2.2. Box

A função box( float length, int divisons, char* filename) que cria esta primitiva gráfica, neste caso uma caixa e que recebe como parâmetros o comprimento do lado de cada quadrado da caixa (length) e o número de divisões que cada lado vai ter (divisions). Esta função irá apenas determinar os vértices para formação da caixa, guardando-o num ficheiro do tipo “nomedoficheiro.3d”.

Deste modo, para gerar os vértices necessários para desenhar a caixa dividimos a mesma por faces. Cada face é composta por divisions*divisons (divisions^2) triângulos em que o comprimento dos seus catetos é length/divisions, sendo que cada face contem 2*(divisions^2) vértices. Assim multiplicando esse número de vértices por 3 obtemos o número de coordenadas de cada vértice, sendo este valor útil para quando precisarmos de ler o conteúdo do “nomedoficheiro.3d”.

Passando para a criação de vértices de cada face, definimos 6 (2 para cada 2 faces opostas) variáveis para saber as coordenadas das faces opostas, como por exemplo, do eixo dos X e Z para saber as coordenadas das faces as faces de cima e de baixo, do eixo dos X e Y para saber as coordenadas das faces da frente e de trás e do eixo dos Y e Z para saber as coordenadas das faces da esquerda e da direita, respetivamente:

Faces Cima/baixo
float var_xBT = -length / 2;
float var_zBT = -length / 2;
Faces Frente/atrás
float var_yFB = -length / 2; //FB -> Front Back
float var_xFB = -length / 2;
Faces Esquerda/direita
float var_yLR = -length / 2; //LR -> Left Right
float var_zLR = -length / 2;

Dividimos tudo por 2 para obter a caixa centrada no ponto (0,0).

Para criarmos os vários triângulos de cada face (divisions^2 triângulos) fizemos dois ciclos for em que no 1º ciclo atualizamos as variáveis, fazendo com que se escrevem as coordenadas dos vértices da linha seguinte(var_zBT += divSize; var_yFB += divSize; var_yLR += divSize;), acima da que foi escrita anteriormente, fazendo também o "reset" das variáveis para começarem a escrever as coordenadas dos vértice no início da face (var_xBT = -length / 2; var_xFB = -length/ 2; var_zLR = -length/ 2;), fazendo o mesmo para . No 2º ciclo escrevemos as coordenadas de uma fila de cada face da caixa em que atualizamos as coordenadas necessárias para escrever as coordenadas do triângulo imediatamente a seguir na mesma linha do anterior (var_xBT += divSize; var_xFB += divSize; var_zLR += divSize;). As coordenadas de cada vértice foram criadas com base na regra da mão direita.

```
void box(float length, int divisions, char* filename) {

    FILE* f;
    f = fopen(filename, "w");

    float divSize = length / divisions;
    int numVertices = 3 * 2 * pow(divisions, 2.0f) * 6;

    if (f) {
        fprintf(f, "%d \n", numVertices * 3);

        float incr1 = -length / 2;
        float incr2 = -length / 2;

        //float var_xBT = -length / 2; //BT -> Bottom Top
        //float var_zBT = -length / 2;
        //float var_yFB = -length / 2; //FB -> Front Back
        //float var_xFB = -length / 2;
        //float var_yLR = -length / 2; //LR -> Left Right
        //float var_zLR = -length / 2;

        for (int i = 1; i <= divisions; i++) {
            for (int j = 1; j <= divisions; j++) {

                //bottom
                fprintf(f, "%f %f %f \n", incr1, -length / 2, incr2);
                fprintf(f, "%f %f %f \n", incr1 + divSize, -length / 2, incr2 + divSize);
                fprintf(f, "%f %f %f \n", incr1, -length / 2, incr2 + divSize);

                fprintf(f, "%f %f %f \n", incr1, -length / 2, incr2);
                fprintf(f, "%f %f %f \n", incr1 + divSize, -length / 2, incr2);
                fprintf(f, "%f %f %f \n", incr1 + divSize, -length / 2, incr2 + divSize);

                //top
                fprintf(f, "%f %f %f \n", incr1, length / 2, incr2);
                fprintf(f, "%f %f %f \n", incr1, length / 2, incr2 + divSize);
                fprintf(f, "%f %f %f \n", incr1 + divSize, length / 2, incr2 + divSize);

                fprintf(f, "%f %f %f \n", incr1, length / 2, incr2);
                fprintf(f, "%f %f %f \n", incr1 + divSize, length / 2, incr2 + divSize);
                fprintf(f, "%f %f %f \n", incr1 + divSize, length / 2, incr2);

                //front
                fprintf(f, "%f %f %f \n", incr1, incr2 + divSize, length / 2);
                fprintf(f, "%f %f %f \n", incr1, incr2, length / 2);
                fprintf(f, "%f %f %f \n", incr1 + divSize, incr2, length / 2);

                fprintf(f, "%f %f %f \n", incr1, incr2 + divSize, length / 2);
                fprintf(f, "%f %f %f \n", incr1 + divSize, incr2, length / 2);
                fprintf(f, "%f %f %f \n", incr1 + divSize, incr2 + divSize, length / 2);
```

```
            //back
            fprintf(f, "%f %f %f \n", incr1, incr2 + divSize, -length / 2);
            fprintf(f, "%f %f %f \n", incr1 + divSize, incr2, -length / 2);
            fprintf(f, "%f %f %f \n", incr1, incr2, -length / 2);

            fprintf(f, "%f %f %f \n", incr1, incr2 + divSize, -length / 2);
            fprintf(f, "%f %f %f \n", incr1 + divSize, incr2 + divSize, -length / 2);
            fprintf(f, "%f %f %f \n", incr1 + divSize, incr2, -length / 2);

            //left
            fprintf(f, "%f %f %f \n", -length / 2, incr2 + divSize, incr1 + divSize);
            fprintf(f, "%f %f %f \n", -length / 2, incr2 + divSize, incr1);
            fprintf(f, "%f %f %f \n", -length / 2, incr2, incr1);

            fprintf(f, "%f %f %f \n", -length / 2, incr2 + divSize, incr1 + divSize);
            fprintf(f, "%f %f %f \n", -length / 2, incr2, incr1);
            fprintf(f, "%f %f %f \n", -length / 2, incr2, incr1 + divSize);

            //right
            fprintf(f, "%f %f %f \n", length / 2, incr2 + divSize, incr1 + divSize);
            fprintf(f, "%f %f %f \n", length / 2, incr2, incr1);
            fprintf(f, "%f %f %f \n", length / 2, incr2 + divSize, incr1);

            fprintf(f, "%f %f %f \n", length / 2, incr2 + divSize, incr1 + divSize);
            fprintf(f, "%f %f %f \n", length / 2, incr2, incr1 + divSize);
            fprintf(f, "%f %f %f \n", length / 2, incr2, incr1);

            incr1 += divSize;

            //var_xBT += divSize;
            //var_xFB += divSize;
            //var_zLR += divSize;

        }
        incr2 += divSize;

        //var_zBT += divSize;
        //var_yFB += divSize;
        //var_yLR += divSize;

        incr1 = -length / 2;

        //var_xBT = -length / 2;
        //var_xFB = -length / 2;
        //var_zLR = -length / 2;
        }
    printf("Ficheiro .3d criado com sucesso.\n");
}
fclose(f);
```

Figuras 3 e 4: Função que gera a Box

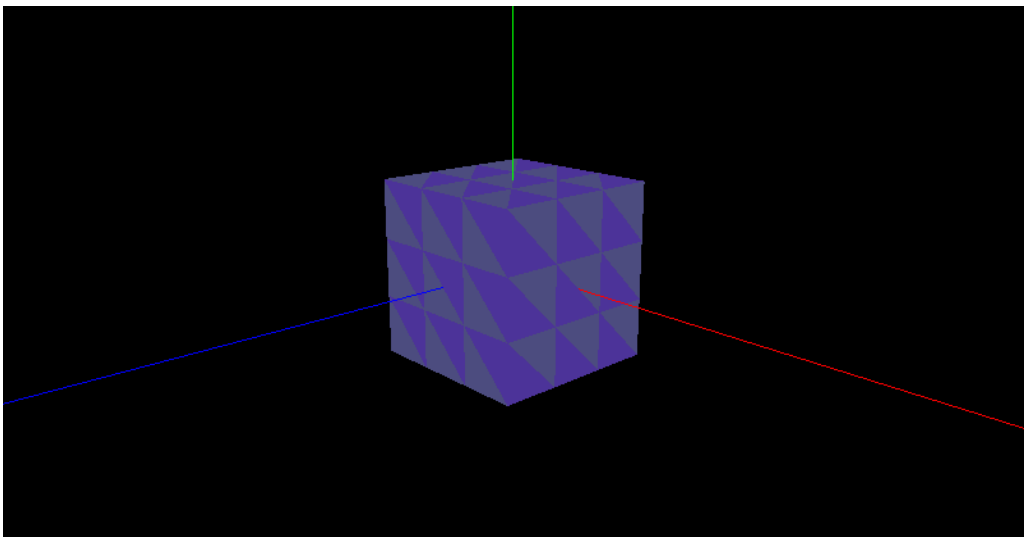

Figura 5: Box de comprimento do lado da base 1 e 3x3 divisões

## 2.3. Sphere

Para este modelo criamos a função sphere(float radius, int slices, int stacks, char* filename) que receberá os parâmetros raio da esfera, número de *slices* (isto é, número de divisões da esfera produzida ao longo do eixo do x), número de *stacks* (isto é, número de divisões da esfera produzida ao longo do eixo do y) e o ficheiro onde serão guardados os vértices.

Para começar, calculamos o ângulo que existe entre cada *slice* (armazenado na variável angleCirc) e o ângulo que existe entre cada *stack* (armazenado na variável angleSides). Estes ângulos serão representados em radianos e serão responsáveis pela variação do ângulo ao longo das *slices* e das *stacks.*

Devido ao facto de cada esfera ser constituída por um determinado número de *stacks* e um determinado número de *slices*, com o objetivo de construir esta primitiva gráfica utilizamos dois ciclos *for*, responsáveis por percorrer cada *stack* e as *slices* de cada *stack* do ciclo exterior. No primeiro ciclo o ângulo angleSides é iniciado com o valor $-\pi/2$ e no final do ciclo esta variável terá o valor $\pi/2$. Já no segundo ciclo o ângulo angleCirc inicia com valor 0 e no fim de cada ciclo terá o valor $2\pi$. Estes valores iniciais dos ângulos mencionados permitirão centrar a esfera na origem.

A cada iteração de cada ciclo são calculados os diferentes valores que as coordenadas dos vértices de cada triângulo da esfera podem tomar, sendo que no primeiro ciclo são calculados os possíveis valores que a coordenada y dos vértices pode tomar e no segundo ciclo os possíveis valores para as coordenadas x e z dos vértices.

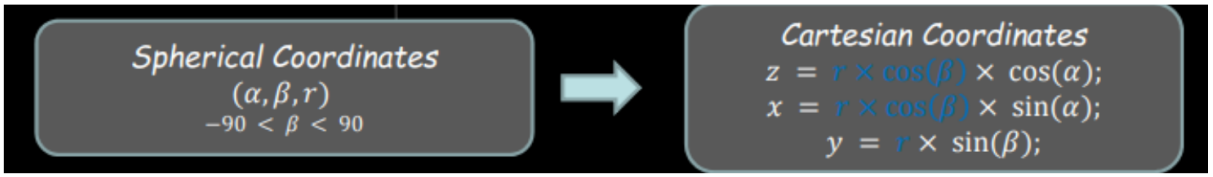


Figura 6: Fórmula de conversão de coordenadas esféricas para cartesianas)

O processo de representação da esfera consistiu numa sequência de impressões de vértices utilizando dois ciclos, onde os ângulos angleSides e angleCirc variavam de acordo com o número de *slices* e de *stacks* requisitados pelo utilizador, sendo desenhados dois triângulos para cada *slice*.

```
void sphere(float radius, int slices, int stacks, char* filename) {
    FILE* f;
    f = fopen(filename, "w");
    float angleCirc = 2 * M_PI / slices;
    float angleSides = M_PI / stacks;
    int numVertices = 6 * stacks * slices;

    if (f) {
        fprintf(f, "%d \n", 3 * numVertices);

        for (int i = 0; i < stacks; i++) {
            float angleSidesTemp = -(M_PI / 2) + i * angleSides;
            float y1 = radius * sin(angleSidesTemp + angleSides);
            float y2 = radius * sin(angleSidesTemp);
            for (int j = 0; j < slices; j++) {
                float angleCircTemp = j * angleCirc;
                float x1 = radius * cos(angleSidesTemp + angleSides) * sin(angleCircTemp);
                float x2 = radius * cos(angleSidesTemp) * sin(angleCircTemp);
                float x3 = radius * cos(angleSidesTemp) * sin(angleCircTemp + angleCirc);
                float x4 = radius * cos(angleSidesTemp + angleSides) * sin(angleCircTemp + angleCirc);
                float z1 = radius * cos(angleSidesTemp + angleSides) * cos(angleCircTemp);
                float z2 = radius * cos(angleSidesTemp) * cos(angleCircTemp);
                float z3 = radius * cos(angleSidesTemp) * cos(angleCircTemp + angleCirc);
                float z4 = radius * cos(angleSidesTemp + angleSides) * cos(angleCircTemp + angleCirc);

                fprintf(f, "%f %f %f \n", x1, y1, z1);
                fprintf(f, "%f %f %f \n", x2, y2, z2);
                fprintf(f, "%f %f %f \n", x3, y2, z3);

                fprintf(f, "%f %f %f \n", x1, y1, z1);
                fprintf(f, "%f %f %f \n", x3, y2, z3);
                fprintf(f, "%f %f %f \n", x4, y1, z4);
            }
        }
        printf("Ficheiro .3d criado com sucesso.\n");
    }
    else {
        printf("Erro ao criar o ficheiro.\nTente novamente.\n");
    }
    fclose(f);
}
```

Figura 7: Função que gera a esfera

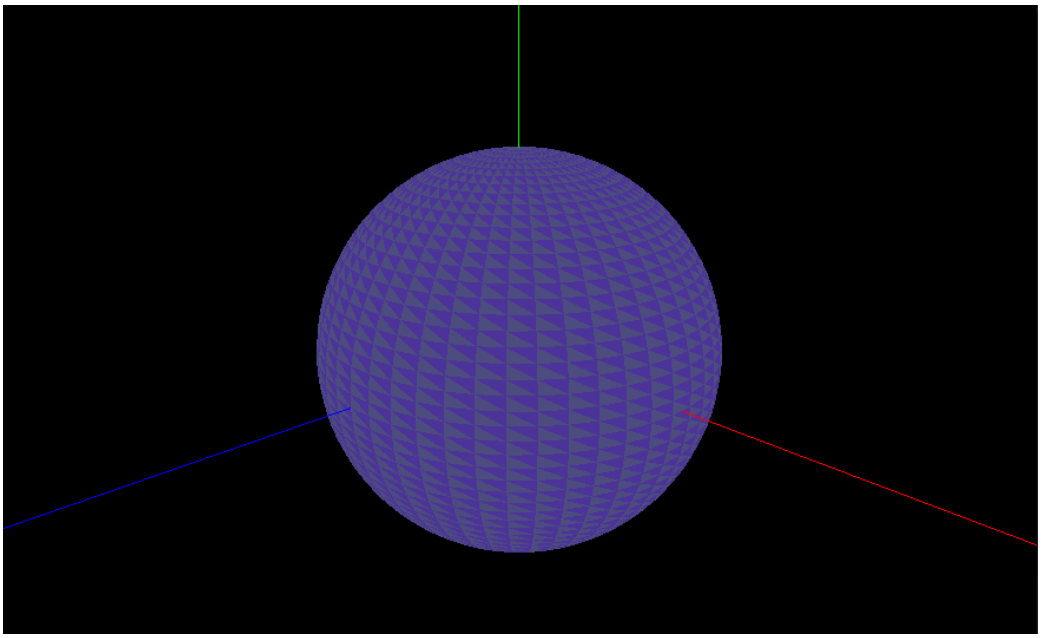

Figura 8: Esfera de raio 3, 50 slices e stacks

## 2.4. Cone

Para a criação do cone criamos a função cone(float radius, float height, int slices, int stacks, char* filename), sendo cada parâmetro o raio,altura,divisões ao longo do x,divisões ao longo do y e nome com o qual queremos criar o ficheiro.

Em primeiro lugar definimos duas variáveis *float angleBase = 2 * M_PI / slices* e *float angleSides = height / stacks* que representam cada divisão ao longo do X(*angleBase)* e ao longo do eixo do Y (*angleSides).*

De seguida, passamos a criação dos vértices para a base.Dentro de um ciclo for , que vai percorrer as *slices* , definimos mais uma variável *float angleBaseTemp = i * angleBase* que vai ser usada para calcularmos as coordenadas de cada ponto.A fórmula usada para calcular a coordenada X foi *radius * sin(angleBaseTemp)* e para o Y *radius * cos(angleBaseTemp).*

Para a criação dos lados do cone temos um primeiro ciclo for a iterar sobre as *stacks* que vai criar 4 variáveis:

1. float sides1 = i * angleSides;
2. float sides2 = (i + 1) * angleSides;
3. float rTemp1 = radius - ((radius / stacks) * i);
4. float rTemp2 = radius - ((radius / stacks) * (i + 1));

Depois definimos outro ciclo iterando agora sobre as *slices* onde criamos as variáveis:

1. float angleBaseTemp2 = j * angleBase;
2. float x1 = rTemp1 * sin(angleBaseTemp2);
3. float x2 = rTemp2 * sin(angleBaseTemp2 + angleBase);

4. float x3 = rTemp2 * sin(angleBaseTemp2);
5. float x4 = rTemp1 * sin(angleBaseTemp2 + angleBase);
6. float y1 = sides1;
7. float y2 = sides2;
8. float z1 = rTemp1 * cos(angleBaseTemp2);
9. float z2 = rTemp2 * cos(angleBaseTemp2 + angleBase);
10. float z3 = rTemp2 * cos(angleBaseTemp2);
11. float z4 = rTemp1 * cos(angleBaseTemp2 + angleBase);

Por fim, criamos dois triângulos de cada vez, de acordo com a regra da mão direita, com as coordenadas anteriormente definidas.

```
void cone(float radius, float height, int slices, int stacks, char* filename) {
    FILE* f;
    f = fopen(filename, "w");


    float angleBase = 2 * M_PI / slices;
    float angleSides = height / stacks;


    if (f){
        fprintf(f, "%d \n", 3 * 3 * slices + 3 * 6 * stacks * slices); //perguntar gouveia
        //base
        for (int i = 0; i < slices; i++) {
            float angleBaseTemp = i * angleBase;
            fprintf(f, "%f %f %f \n", radius * sin(angleBaseTemp), 0.0f, radius * cos(angleBaseTemp));
            fprintf(f, "%f %f %f \n", 0.0f, 0.0, 0.0f);
            fprintf(f, "%f %f %f \n", radius * sin(angleBaseTemp + angleBase), 0.0f, radius * cos(angleBaseTemp + angleBase));
        }

        //sides
        for (int i = 0; i < stacks; i++) {
            float sides1 = i * angleSides;
            float sides2 = (i + 1) * angleSides;
            float rTemp1 = radius - ((radius / stacks) * i);
            float rTemp2 = radius - ((radius / stacks) * (i + 1));
            for (int j = 0; j < slices; j++) {
                float angleBaseTemp2 = j * angleBase;

                float x1 = rTemp1 * sin(angleBaseTemp2);
                float x2 = rTemp2 * sin(angleBaseTemp2 + angleBase);
                float x3 = rTemp2 * sin(angleBaseTemp2);
                float x4 = rTemp1 * sin(angleBaseTemp2 + angleBase);
                float y1 = sides1;
                float y2 = sides2;
                float z1 = rTemp1 * cos(angleBaseTemp2);
                float z2 = rTemp2 * cos(angleBaseTemp2 + angleBase);
                float z3 = rTemp2 * cos(angleBaseTemp2);
                float z4 = rTemp1 * cos(angleBaseTemp2 + angleBase);

                fprintf(f, "%f %f %f \n", x1, y1, z1);
                fprintf(f, "%f %f %f \n", x2, y2, z2);
                fprintf(f, "%f %f %f \n", x3, y2, z3);

                fprintf(f, "%f %f %f \n", x1, y1, z1);
                fprintf(f, "%f %f %f \n", x4, y1, z4);
                fprintf(f, "%f %f %f \n", x2, y2, z2);
            }
        }
    }
    else {
    printf("Erro ao criar o ficheiro.\nTente novamente.\n");
    }

    fclose(f);


}
```

Figura 9 : Função que gera o cone

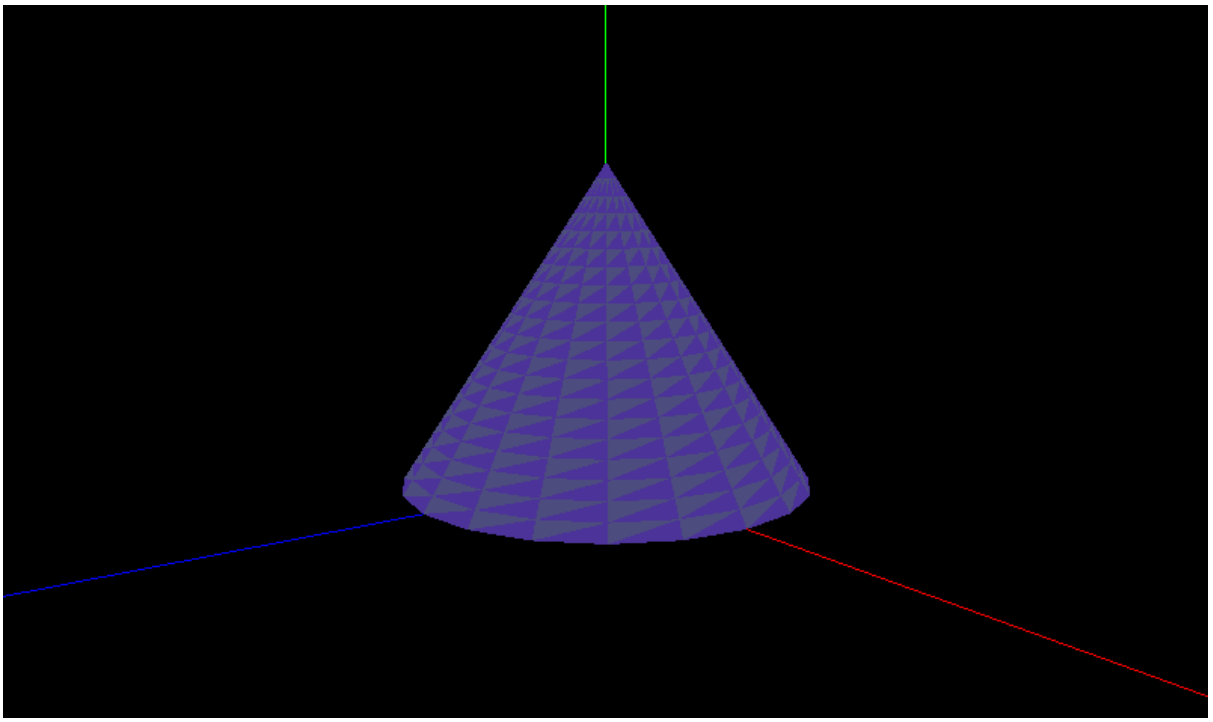

Figura 10: Cone raio 2, altura 3, 20 slices and 20 stacks

Fizemos também uma mistura de várias primitivas gráficas, tal como tinha também nos ficheiros de teste

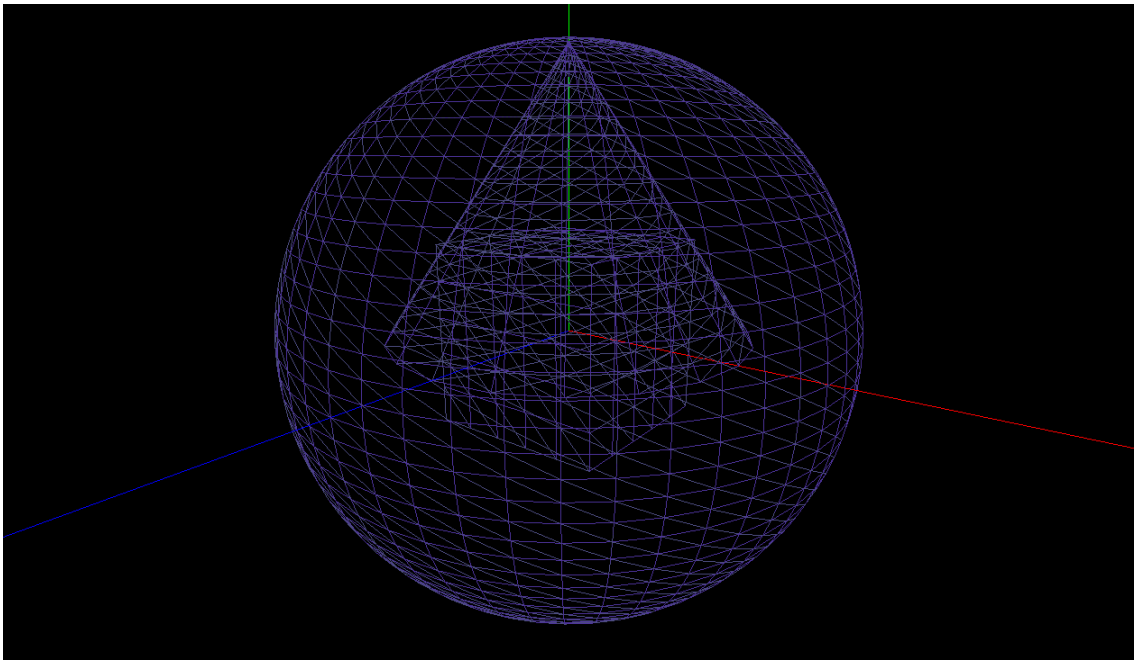

Figura 11:esfera de 3 50 50, cone raio 2, altura 3, 20 slices and 20 stacks e box de base 2 e 5x5 divisões

# 3. Engine

O Engine corresponde à parte do projeto que utiliza o ficheiro.3d gerado pelo Generator para produzir a forma geométrica pretendida.

Começamos por interpretar o ficheiro XML fornecido, utilizando a função readXML, com auxílio da biblioteca tinyxml2, obtendo as informações da position(que será utilizada no radius e nos ângulos alfa e beta (angleAlpha e angleBeta, respetivamente) de forma a podermos movimentar a câmara através do rato), lookat, up e proj para definir a posição da câmara, guardando cada coordenada no seu respetivo array (pos, lookat, up e proj), seguidamente vemos quantas figuras temos de desenhar percorrendo todos os models no ficheiro xml alocando dinamicamente o tamanho necessário para os seus vértices no array all_vertices e, por fim lemos o ficheiro.3d .

```
void readxml(char* filename) {
    XMLDocument doc;
    const char* file3d;
    int i = 0, j = 1, k = 1;// j é o numero de primitivas

    if (!(doc.LoadFile(filename))) {

        XMLElement* position = doc.FirstChildElement("world")->FirstChildElement("camera")->FirstChildElement("position");

        pos[0] = atof(position->Attribute("x"));
        pos[1] = atof(position->Attribute("y"));
        pos[2] = atof(position->Attribute("z"));


        XMLElement* LA = doc.FirstChildElement("world")->FirstChildElement("camera")->FirstChildElement("lookAt");

        lookat[0] = atof(LA->Attribute("x"));
        lookat[1] = atof(LA->Attribute("y"));
        lookat[2] = atof(LA->Attribute("z"));

        XMLElement* UP = doc.FirstChildElement("world")->FirstChildElement("camera")->FirstChildElement("up");

        up[0] = atof(UP->Attribute("x"));
        up[1] = atof(UP->Attribute("y"));
        up[2] = atof(UP->Attribute("z"));

        XMLElement* PJ = doc.FirstChildElement("world")->FirstChildElement("camera")->FirstChildElement("projection");

        proj[0] = atof(PJ->Attribute("fov"));
        proj[1] = atof(PJ->Attribute("near"));
        proj[2] = atof(PJ->Attribute("far"));
```

Figura 12 : Função que lê o ficheiro XML

```
    for (XMLElement* mod = doc.FirstChildElement("world")->FirstChildElement("group")->FirstChildElement("models")->FirstChildElement("model"); mod; mod = mod->NextSiblingElement()) {
        if (j = k) {
            k = k * 2;
            all_vertices = (float**)malloc(sizeof(float) * k);
        }
        j++;

    }

    for (XMLElement* mod = doc.FirstChildElement("world")->FirstChildElement("group")->FirstChildElement("models")->FirstChildElement("model"); mod; mod = mod->NextSiblingElement()) {

        file3d = mod->Attribute("file");

        readFile(file3d);
        all_vertices[i] = (float*)malloc(sizeof(float) * (N + 1));
        all_vertices[i][0] = (float)N;
        for (int w = 1; w <= N; w++) {
            all_vertices[i][w] = vertices[w - 1];
        }
        i++;
    }

    n_primitivas = i;
    //camera stuff
    radius = sqrt(pow(pos[0], 2) + pow(pos[1], 2) + pow(pos[2], 2)), dist = sqrt(pow(pos[0], 2) + pow(pos[2], 2)); //dist is the same as radius but y coord is 0
    angleAlpha = acos(pos[2] / dist), angleBeta = acos(dist / radius);
}

else {
    printf("O ficheiro XML nao foi encontrado.\n");
}
```

Figura 13: Função que lê o ficheiro XML(continuação)(em cima)
Figura 14: Função que lê o ficheiro .3d(em baixo)

```
void readFile(const char* filename) {
    FILE* f;

    f = fopen(filename, "r");
    char* n = (char*)malloc(sizeof(char) * 10);
    n[0] = 0;
    char* novo = (char*)malloc(sizeof(char) * 9);
    novo[0] = 0;
    char c;
    int i = 0;
    int flag = 1;

    if (f) {
        while ((c = fgetc(f)) != EOF) {
            if (flag) {
                if (c == '\n') {
                    num = atoi(n);
                    vertices = (float*)malloc(sizeof(float) * num);
                    flag = 0;
                }
                else strncat(n, &c, 1);
            }
            else {
                if (c == ' ') {
                    float elem = atof(novo);
                    vertices[i++] = elem;
                    novo[0] = 0;
                }
                else strncat(novo, &c, 1);
            }
        }
        N = i;
    }
    else {
        printf("Erro ao criar o ficheiro.\nTente novamente.\n");
    }
    fclose(f);
}
```

Fazemos depois o parsing do ficheiro .3d na função readFile, onde é lido o número de coordenadas presentes no ficheiro para uma variável num que, em seguida, é usada para definir o tamanho de memória alocada do array vertices onde são guardadas as coordenadas lidas no restante conteúdo do ficheiro 3d.

Tornamos também possível:

- A rotação da cena utilizando o rato

```
void mouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON) {
        if (state == GLUT_DOWN) {
            cord_x = x;
            cord_y = y;
        }
        else {
            cord_x = -1.0f;
            cord_y = -1.0f;
        }
    }
}

void motion_mouse(int x, int y) {
    if (cord_x >= 0) {
        angleAlpha = (x + cord_x) * 0.01f;
        if (angleBeta < M_PI / 2) {
            angleBeta = (y - cord_y) * 0.01f;
        }
        if (angleBeta > -M_PI / 2) {
            angleBeta = (y - cord_y) * 0.01f;
        }
    }

    glutPostRedisplay();
}
```

Figura 15: Funções responsáveis por controlar o movimento do rato

- A rotação da cena utilizando o teclado

```
// write function to process keyboard events
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
    case ('w'):
        radius -= 0.3f;
        break;
    case ('s'):
        radius += 0.3f;
        break;
    default:
        break;
    }

    glutPostRedisplay();
}
```

Figura 16: Funções responsáveis por controlar o teclado

- Tornar apenas visível os pontos, linhas e tudo preenchido através de um menu

```
void menu(int value) {
    switch (value) {
    case 1:
        glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
        break;
    case 2:
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        break;
    case 3:
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        break;
    default:
        break;
    }

    glutPostRedisplay();
}
```

Figura 17: Função responsável pelo menu de alteração para preencher a figura, apenas as linhas visíveis e apenas pontos visíveis.

- A movimentação da câmara

```
void renderScene(void) {

    // clear buffers
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);



    // set the camera
    glLoadIdentity();

    gluLookAt(cos(angleBeta) * sin(angleAlpha) * radius, sin(angleBeta) * radius, cos(angleBeta) * cos(angleAlpha) * radius,
        lookat[0], lookat[1], lookat[2],
        up[0], up[1], up[2]);
```

Figura 18: Código para movimentação da câmara

É calculado o raio radius e os ângulos angleAlpha e angleBeta na função readxml.

```
//camera stuff
radius = sqrt(pow(pos[0], 2) + pow(pos[1], 2) + pow(pos[2], 2)), dist = sqrt(pow(pos[0], 2) + pow(pos[2], 2));
angleAlpha = acos(pos[2] / dist), angleBeta = acos(dist / radius);
```

Figura 19: Código para determinar a posição inicial da câmara (dada no ficheiro xml)

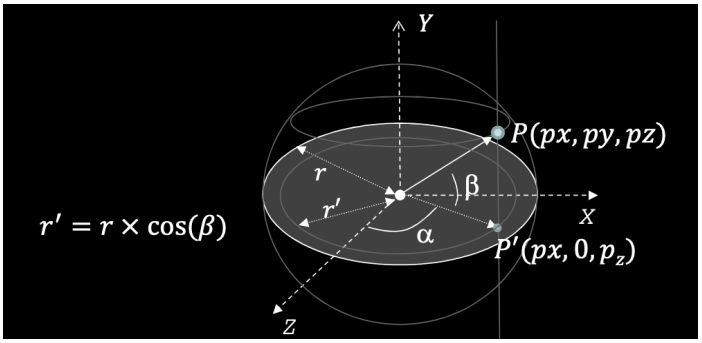


Figura 20: Representação das variáveis que estão atribuídas no readxml (Figura 19)

# 4. Conclusão

Com esta primeira fase do trabalho, melhoramos a nossa capacidade de trabalhar como glut, OpenGL e as linguagens de programação C e C++. Compreendemos melhor o funcionamento da cadeira o que será importante na realização das seguintes fases do trabalho.

Inicialmente tivemos dificuldade na construção da esfera em função das slices e stacks e a fazer com que a câmara se movimentasse e mais tarde, encontramos também problemas a desenhar mais do que uma figura.

Em suma, apesar das dificuldades, achamos que o nosso trabalho cumpre todos os requisitos propostos e foi realizado com sucesso.