



Escola de Engenharia
Universidade do Minho

Computação Gráfica

Primitivas Gráficas

Relatório de Desenvolvimento

João Manuel Martins Cerqueira (A65432)
Sónia Catarina Guerra Costa (A71506)
Tiago Costa Loureiro (A71191)

Ano letivo 2016/2017

Conteúdo

1	Introdução	2
1.1	Estrutura do documento	2
2	Análise e Especificação	3
2.1	Especificação do Problema	3
3	Conceção/Desenho da Resolução	5
3.1	Desenvolvimento do Gerador	5
3.1.1	Plano	6
3.1.2	Caixa	7
3.1.3	Esfera	8
3.1.4	Cone	9
3.2	Desenvolvimento do Motor	10
4	Codificação e Testes	11
4.1	Problemas de Implementação	11
4.2	Compilação	12
4.2.1	Como executar	12
4.2.2	Gerador	13
4.2.3	Motor	14
5	Conclusão	15
A	Código do Programa	16
A.1	Código Gerador	16
A.2	Código Motor	21

Capítulo 1

Introdução

Este trabalho prático está dividido em duas partes. Uma primeira parte onde é pedido que se desenvolva um gerador de vértices de triângulos necessários para criar as formas geométricas primitivas e guardá-los num ficheiro .3d. E uma segunda etapa onde o principal objetivo é o desenvolvimento de um motor capaz de ler um ficheiro XML com o nome dos ficheiros .3d onde os vértices do modelo a desenvolver estão guardados e assim criar as formas geométricas primitivas pretendidas.

1.1 Estrutura do documento

A estrutura que este relatório segue, excluindo o presente capítulo onde se faz uma pequena introdução do assunto, :

- No capítulo 2 faz-se uma análise detalhada do problema proposto de modo a poder-se especificar as entradas, resultados e formas de transformação.
- No capítulo 3 faz-se referência à conceção/desenho da Resolução dos problemas propostos, mostrando assim as estruturas de dados e todos os algoritmos usados durante a realização deste trabalho.
- No capítulo 4 faz-se referência a decisões e problemas de implementação que surgiram.
- No capítulo 5 faz-se uma conclusão / síntese de todo o trabalho realizado e uma análise crítica dos resultados.
- Em apêndice faz-se uma referência ao código necessário para a implementação do programa.

Este trabalho é finalizado com a Bibliografia que contém todas as referências bibliográficas usadas na realização do mesmo.

Capítulo 2

Análise e Especificação

2.1 Especificação do Problema

Primeiramente ser desenvolvido um Gerador em C++ que recebe como argumentos os dados necessários para a criação de uma figura. Após processar os dados, o programa imprime num ficheiro .3d, cujo nome é dado como argumento, todos os vértices dos triângulos precisos para gerar uma figura. Este gerador capaz de desenvolver os vértices dos triângulos para as seguintes figuras:

- Plano, sendo este um quadrado no plano XZ, centrado na origem, feito com dois triângulos;
- Caixa, definido pelas dimensões de X, Y e Z e opcionalmente pelo número de divisões;
- Esfera, definida por raio, fatias e camadas;
- Cone, definido pelo raio da base, pela altura, fatias e camadas.

A segunda parte do trabalho consiste no desenvolvimento de um Motor em C++. Este motor recebe como argumento um ficheiro XML. Neste ficheiro XML estão guardados nomes de ficheiros .3d onde anteriormente foram guardados vértices de uma figura. Ao abrir e processar esses mesmos ficheiros será criada uma cena em 3 dimensões a partir desses mesmos vértices. Neste módulo encontram-se as informações sobre a câmara utilizada e também estão definidas as funções que permitem a interação do utilizador com o cenário final.

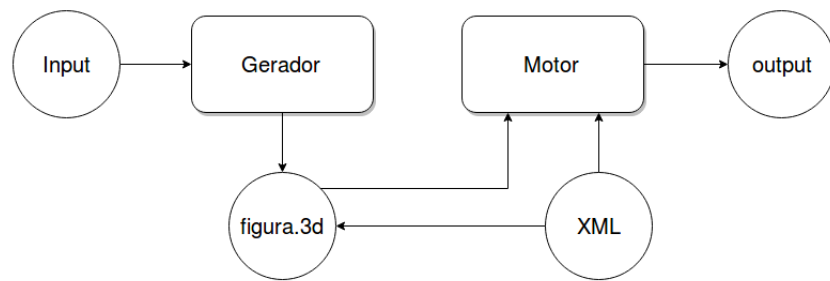


Figura 2.1: Diagrama do trabalho

Capítulo 3


Conceção/Desenho da Resolução

3.1 Desenvolvimento do Gerador

Este módulo tem como input os parâmetros necessários para a criação dos modelos pretendidos (como se pode ver no exemplo abaixo) e também recebe como último parâmetro o nome do ficheiro criado como output. Se não for dado nenhum parâmetro ou até mesmo dados parâmetros errados, o programa sai com mensagem de erro.

```
$ gerador plane 4 plano.3d
```

Inicialmente imprime-se no ficheiro .3d o número de triângulos cujos vértices serão impressos. Para desenvolver o Gerador é preciso perceber como são criadas as figuras a partir de triângulos.



```
plane.3d
1 2
2 -2 0.0 -2
3 -2 0.0 2
4 2 0.0 2
5 2 0.0 2
6 2 0.0 -2
7 -2 0.0 -2
```

Figura 3.1: Exemplo de output de plano.3d

3.1.1 Plano

Para construir o plano, a função recebe um único parâmetro, a sua dimensão. Como é referido no enunciado, sabe-se que o plano é quadrado e que se encontra no plano XZ, restringindo as coordenadas de y a zero sendo apenas necessário calcular as coordenadas x e z. Sendo o plano quadrado e centrado na origem, rapidamente se chega que a coordenada x tem os valores $x/2$ ou $-x/2$ e a coordenada z é análoga. Ou seja, os 4 pontos do plano são $(-x/2, 0, -z/2)$, $(-x/2, 0, z/2)$, $(x/2, 0, -z/2)$ e $(x/2, 0, z/2)$.

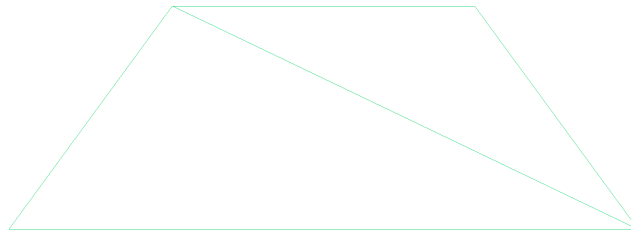


Figura 3.2: Exemplo plano

3.1.2 Caixa

No desenho da caixa começa-se por receber as suas dimensões x , y e z e ainda o número de dimensões. Pretende-se desenhar a caixa centrada no ponto que o motor escolheu, por isso começa por calcular o seu ponto de origem. Sabendo as dimensões da caixa, facilmente percebemos que o seu ponto de origem é o ponto $(-x/2, -y/2, -z/2)$. Sabemos também que na diagonal oposta temos o ponto $(x/2, y/2, z/2)$. De seguida, calcula-se qual as dimensões x , y e z que cada triângulo terá e estas são guardadas nas variáveis dim_x , dim_y e dim_z respectivamente. Seguidamente, cria-se a caixa a partir do ponto de origem, desenhando as três faces conectadas a esse ponto. Começando em $(-x/2, -y/2, -z/2)$ desenhamos os triângulos conectados a esse ponto e em seguida avança-se para o próximo ponto em z . Depois de percorrermos todos os pontos de $(-x/2, -y/2, -z/2)$ até $(-x/2, -y/2, z/2)$ subimos para a próxima coordenada em y e recomeça-se a coordenada z em $z/2$, até ao ponto $(-x/2, y/2, z/2)$. Após este passo, todos os pontos para $x = -x/2$ foram percorridos e, de forma análoga aos passos anteriores passamos para a próxima coordenada de x até que eventualmente tenham sido desenhadas todas as 3 faces conectadas ao ponto de origem. Após serem desenhadas estas 3 faces, faz-se de forma análoga para as outras 3 faces partindo agora do ponto $(x/2, y/2, z/2)$ e em vez de incrementar para andar nos eixos x , y e z decrementa-se de acordo com as dimensões de cada triângulo.

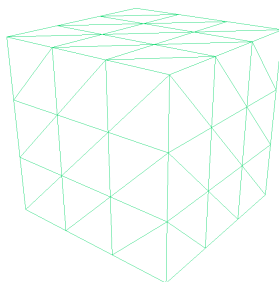


Figura 3.3: Exemplo Caixa

3.1.3 Esfera

Para desenhar a esfera recebe-se como parâmetros o raio da esfera e o número de fatias e camadas. Inicialmente calcula-se o ângulo de cada fatia e camada. De seguida, começa-se a desenhar a esfera a partir do centro e em direção ao topo e á base. Primeiro calcula-se os raios de cada fatia e camada, juntamente com os ângulos de cada uma em relação ás primeiras. Com os raios e com os ângulos obtem-se os 3 pontos de cada triângulo.

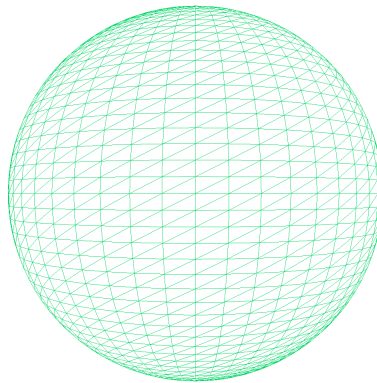


Figura 3.4: Exemplo Esfera

3.1.4 Cone

Para a construção do cone são dados como parâmetros o raio da base, a altura do cone e número de camadas e fatias. Primeiro é calculado ângulo de cada fatia e de cada camada e ainda a altura de cada camada. De seguida calcula-se o raio da próxima camada e, usando esse raio, o raio da primeira base e os ngulos de cada camada e fatia, obtem-se a coordenada x e z de cada ponto e com a altura da camada obtem-se o ponto y.

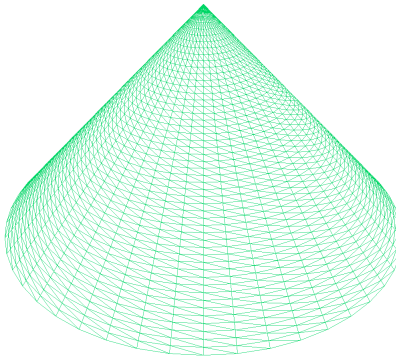


Figura 3.5: Exemplo Cone

3.2 Desenvolvimento do Motor

O motor apresenta um dos principais problemas desta primeira fase devido à necessidade de aprender a manipular os ficheiros XML. A estratégia optada foi primeiro receber o ficheiro XML como argumento e de seguida, com a funo `le_xml(char *nome)` que recebe o nome do ficheiro obtendo todos os nomes dos ficheiros `.3d` que constituem a cena e são guardados num vector chamado `lista_ficheiros`. Após isso, na funo `renderScene()` é chamada a função `desenha()`. Esta última vai ler os nomes dos ficheiros `.3d` que estão no vector `lista_ficheiros` e a partir desses nomes abre os ficheiros correspondentes e desenha os triângulos representados pelos vértices que estão nos ficheiros, atribuindo cores aleatórias a cada figura.

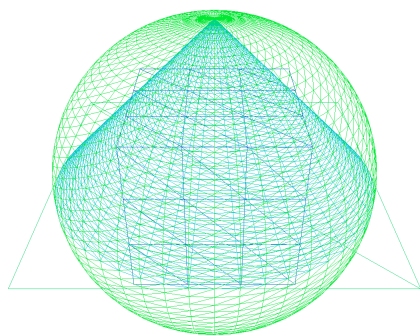


Figura 3.6: Output do Motor

Capítulo 4

Codificação e Testes

4.1 Problemas de Implementação

No desenho das figuras surgiram alguns problemas. Na construção da caixa apenas foi possível desenhar a figura recorrendo a 6 ciclos (for) e 6 condições (if), os quais percorrem todos os pontos possíveis da caixa, incluindo os interiores. Uma vez que não se quer desenhar dentro da caixa (apenas desenhar a superfície) não seria necessário percorrer todos os pontos. Contudo foi a única solução encontrada dados os requisitos. O cone também suscitou algumas dúvidas, pelo facto de ser necessário calcular o raio a cada camada. Numa primeira tentativa, calculou-se o raio de acordo com o raio da base e o cosseno do ângulo da stack, contudo a figura ficava com superfícies curvas.

4.2 Compilação

Para que o programa seja de fácil acesso a qualquer utilizador, foi criada uma makefile que permite executar todo o programa desde o input até à construção final das figuras em 3 dimensões. Foi criado também um ficheiro readme.md que explica da seguinte forma como executar.

4.2.1 Como executar

No terminal escrever make e a makefile vai gerar os ficheiros .3d com as seguintes propriedades:

- plane
 - comprimento = 4
 - nome do ficheiro = plane.3d
- box
 - tamanho X = 2
 - tamanho Y = 2
 - tamanho Z = 2
 - número de divisões = 3
 - nome do ficheiro = box.3d
- sphere
 - raio = 2
 - fatias = 50
 - camadas = 50
 - nome do ficheiro = sphere.3d
- cone
 - raio da base = 2
 - altura = 2
 - fatias = 50
 - camadas = 50
 - nome do ficheiro = cone.3d

De seguida é executado o motor com o argumento "configuracao.xml", que contém esta informação:

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

4.2.2 Gerador

Os ficheiros .3d so gerados na pasta principal do motor.

- Argumentos:
 - Plane
 - * Comprimento
 - * Nome do ficheiro para guardar
 - Box
 - * Tamanho X
 - * Tamanho Y
 - * Tamanho Z
 - * Número de divisões(opcional—default = 1)
 - * Nome do ficheiro para guardar
 - Sphere
 - * Raio
 - * Fatias
 - * Camadas
 - * Nome do ficheiro para guardar
 - Cone
 - * Raio da base
 - * Altura
 - * Fatias
 - * Camadas
 - * Nome do ficheiro para guardar

4.2.3 Motor

- Argumentos:
 - Nome do ficheiro .xml (procura dentro da pasta xml que est na pasta motor)
- Opções do menu:
 - Fill
 - * `glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);`
 - Line
 - * `glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);`
 - Point
 - * `glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);`
- Teclado
 - Seta cima
 - * roda para cima
 - Seta baixo
 - * roda para baixo
 - Seta direita
 - * roda para a direita
 - Seta esquerda
 - * roda para a esquerda
 - Page up
 - * aproxima os objetos
 - Page down
 - * afasta os objetos

Capítulo 5

Conclusão

O presente relatório teve como objetivo explicar todo o processo de construção de um cenário em 3 dimensões desde o input do utilizador até ao output. Neste trabalho, que envolve a construção de modelos gráficos primitivos através de triângulos, não foram usados os métodos mais eficientes, o que não aconteceria se fossem usados VBO's, algo que será implementado numa próxima fase.

Apêndice A

Código do Programa

A.1 Código Gerador

```
#include <iostream>
#include <fstream>
#include <math.h>

using namespace std;

fstream file;

void plano(float comprimento){
    float m_comp = comprimento/2;

    // N DE TRIANGULOS
    file << "2" << endl;

    file << -m_comp << " 0.0 " << -m_comp << endl;
    file << -m_comp << " 0.0 " << m_comp << endl;
    file << m_comp << " 0.0 " << m_comp << endl;

    file << m_comp << " 0.0 " << m_comp << endl;
    file << m_comp << " 0.0 " << -m_comp << endl;
    file << -m_comp << " 0.0 " << -m_comp << endl;
}

void caixa(float x, float y, float z, int dimensions){
    double dim_x = x / dimensions, dim_y = y / dimensions, dim_z = z / dimensions;
    double ori_x = -x / 2, ori_y = -y / 2, ori_z = -z / 2; // origem x, y e z
    double xx = ori_x, yy = ori_y, zz = ori_z; // ponto "origem"

    file << 4 << endl;

    for (xx = ori_x; xx < (-ori_x); xx += dim_x) {
        //x2 = x1 + dim_x;

        for (yy = ori_y; yy < (-ori_y); yy += dim_y) {
            //y2 = y1 + dim_y;

            for (zz = ori_z; zz < (-ori_z); zz += dim_z) {
                //z2 = z1 + dim_z;

                if (xx == ori_x) {
```

```

        //glColor3f(0.09 << " " << 0.5 << " " << 0.99 << endl;
        file << xx << " " << yy + dim_y << " " << zz << endl;
        file << xx << " " << yy << " " << zz << endl;
        file << xx << " " << yy + dim_y << " " << zz + dim_z << endl;

        //glColor3f(0.18 << " " << 0.5 << " " << 0.90 << endl;
        file << xx << " " << yy + dim_y << " " << zz + dim_z << endl;
        file << xx << " " << yy << " " << zz << endl;
        file << xx << " " << yy << " " << zz + dim_z << endl;

    }

    if (yy == ori_y) {
        //glColor3f(0.27 << " " << 0.5 << " " << 0.81 << endl;
        file << xx << " " << yy << " " << zz << endl;
        file << xx + dim_x << " " << yy << " " << zz << endl;
        file << xx + dim_x << " " << yy << " " << zz + dim_z << endl;

        //glColor3f(0.36 << " " << 0.5 << " " << 0.73 << endl;
        file << xx + dim_x << " " << yy << " " << zz + dim_z << endl;
        file << xx << " " << yy << " " << zz + dim_z << endl;
        file << xx << " " << yy << " " << zz << endl;
    }

    if (zz == ori_z) {
        //glColor3f(0.45 << " " << 0.5 << " " << 0.64 << endl;
        file << xx << " " << yy << " " << zz << endl;
        file << xx << " " << yy + dim_y << " " << zz << endl;
        file << xx + dim_x << " " << yy << " " << zz << endl;

        //glColor3f(0.54 << " " << 0.5 << " " << 0.55 << endl;
        file << xx + dim_x << " " << yy << " " << zz << endl;
        file << xx << " " << yy + dim_y << " " << zz << endl;
        file << xx + dim_x << " " << yy + dim_y << " " << zz << endl;
    }
}
}
}

for (xx = -ori_x; xx > ori_x; xx += dim_x) {
    for (yy = -ori_y; yy > ori_y; yy += dim_y) {
        for (zz = -ori_z; zz > ori_z; zz += dim_z) {
            if (xx == -ori_x) {
                //glColor3f(0.63 << " " << 0.63 << " " << 0.46 << endl;
                file << xx << " " << yy << " " << zz << endl;
                file << xx << " " << yy - dim_y << " " << zz << endl;
                file << xx << " " << yy - dim_y << " " << zz - dim_z << endl;

                //glColor3f(0.72 << " " << 0.72 << " " << 0.37 << endl;
                file << xx << " " << yy - dim_y << " " << zz - dim_z << endl;
                file << xx << " " << yy << " " << zz - dim_z << endl;
                file << xx << " " << yy << " " << zz << endl;
            }

            if (yy == -ori_y) {
                //glColor3f(0.81 << " " << 0.81 << " " << 0.28 << endl;
                file << xx - dim_x << " " << yy << " " << zz << endl;
                file << xx << " " << yy << " " << zz << endl;
                file << xx - dim_x << " " << yy << " " << zz - dim_z << endl;

                //glColor3f(0.9 << " " << 0.9 << " " << 0.19 << endl;
                file << xx - dim_x << " " << yy << " " << zz - dim_z << endl;
                file << xx << " " << yy << " " << zz << endl;
                file << xx << " " << yy << " " << zz - dim_z << endl;
            }

            if (zz == -ori_z) {

```

```

        //glColor3f(0.95 << " " << 0.95 << " " << 0.10 << endl;
        file << xx << " " << yy - dim_y << " " << zz << endl;
        file << xx << " " << yy << " " << zz << endl;
        file << xx - dim_x << " " << yy << " " << zz << endl;

        //glColor3f(0.99 << " " << 0.99 << " " << 0.01 << endl;
        file << xx - dim_x << " " << yy << " " << zz << endl;
        file << xx - dim_x << " " << yy - dim_y << " " << zz << endl;
        file << xx << " " << yy - dim_y << " " << zz << endl;
    }
}
}
}

void esfera(float radius, int slices, int stacks){
    int i, j; // iteradores
    double alpha1 = 0, alpha2 = 0; // angulo de cada fatia
    double beta1 = 0, beta2 = 0; // angulo de cada corte
    double alpha = (2 * M_PI) / slices; //
    double beta = -(M_PI) / stacks;

    // N DE TRIANGULOS
    file << 2*stacks*slices << endl;

    for (j = -stacks/2; j < stacks/2; j++) {
        beta1 = beta * j;
        beta2 = beta * (j + 1);
        double raio1 = radius * cos(beta1);
        double raio2 = radius * cos(beta2);

        for (i = 0; i < slices; i++) {
            alpha1 = alpha * i;
            alpha2 = alpha * (i + 1);

            file << raio1 * sin(alpha1) << " " << radius * sin(beta1) << " " << raio1 *
                cos(alpha1) << endl;
            file << raio2 * sin(alpha1) << " " << radius * sin(beta2) << " " << raio2 *
                cos(alpha1) << endl;
            file << raio1 * sin(alpha2) << " " << radius * sin(beta1) << " " << raio1 *
                cos(alpha2) << endl;

            file << raio2 * sin(alpha2) << " " << radius * sin(beta2) << " " << raio2 *
                cos(alpha2) << endl;
            file << raio1 * sin(alpha2) << " " << radius * sin(beta1) << " " << raio1 *
                cos(alpha2) << endl;
            file << raio2 * sin(alpha1) << " " << radius * sin(beta2) << " " << raio2 *
                cos(alpha1) << endl;
        }
    }
}

void cone(float radius, float height, int slices, int stacks) {
    int i, j; // iteradores
    double altura2 = 0, altura1 = 0; // altura de cada base
    double alpha1 = 0, alpha2 = 0; // angulo de cada fatia
    double alpha = (2 * M_PI) / slices; //
    double stack_height = height / stacks;
    double raio2, raio1 = radius;

    // N DE TRIANGULOS
    file << stacks*slices*2+slices << endl;

    for (j = 0; j < stacks; j++) {
        altura2 += stack_height;
        raio1 = radius - radius * ((float)j / stacks);
        raio2 = radius - radius * ((float)(j + 1) / stacks);
    }
}

```

```

for (i = 0; i < slices; i++) {
    alpha1 = alpha * i;
    alpha2 = alpha * (i + 1);

    if (j == 0) {
        file << 0.0 << " " << 0.0 << " " << 0.0 << endl;
        file << raio1 * sin(alpha2) << " " << 0.0 << " " << raio1 * cos(alpha2) <<
            endl;
        file << raio1 * sin(alpha1) << " " << 0.0 << " " << raio1 * cos(alpha1) <<
            endl;
    }

    file << raio1 * sin(alpha1) << " " << altura1 << " " << raio1 * cos(alpha1) <<
        endl;
    file << raio1 * sin(alpha2) << " " << altura1 << " " << raio1 * cos(alpha2) <<
        endl;
    file << raio2 * sin(alpha1) << " " << altura2 << " " << raio2 * cos(alpha1) <<
        endl;

    file << raio1 * sin(alpha2) << " " << altura1 << " " << raio1 * cos(alpha2) <<
        endl;
    file << raio2 * sin(alpha2) << " " << altura2 << " " << raio2 * cos(alpha2) <<
        endl;
    file << raio2 * sin(alpha1) << " " << altura2 << " " << raio2 * cos(alpha1) <<
        endl;
    }
    altura1 = altura2;
}

int main(int argc, char **argv) {

    if(argc > 1){
        /* S para quando est em debug
        string caminho = "../motor/";
        */
        string caminho = "../motor/";

        if(argv[1] == string("plane")){
            if(argc == 4){
                file.open(caminho + argv[3], std::fstream::out);
                plano(atoi(argv[2]));
            }else{
                cout << "Faltam argumentos!" << endl;
                cout << "Os argumentos necessrios so:" << endl;
                cout << "\t- comprimento" << endl;
                cout << "\t- nome do ficheiro para guardar os vrtices" << endl;
            }
        }else if(argv[1] == string("box")) {

            if (argc == 6){
                file.open(caminho + argv[5], std::fstream::out);
                caixa(stof(argv[2]), stof(argv[3]), stof(argv[4]), 1);
            }else if(argc == 7){
                file.open(caminho + argv[6], std::fstream::out);
                caixa(stof(argv[2]), stof(argv[3]), stof(argv[4]), atoi(argv[5]) );
            }else{
                cout << "Faltam argumentos!" << endl;
                cout << "Os argumentos necessrios so:" << endl;
                cout << "\t- Tamanho X" << endl;
                cout << "\t- Tamanho Y" << endl;
                cout << "\t- Tamanho Z" << endl;
                cout << "\t- (OPCIONAL) nmero de divises" << endl;
                cout << "\t- nome do ficheiro para guardar os vrtices" << endl;
            }
        }else if(argv[1] == string("sphere")){

```

```

        if(argc == 6){
            file.open(caminho + argv[5], std::fstream::out);
            esfera(stof(argv[2]), stof(argv[3]), stof(argv[4]) );
        }else{
            cout << "Faltam argumentos!" << endl;
            cout << "Os argumentos necessrios so:" << endl;
            cout << "\t- raio" << endl;
            cout << "\t- fatias" << endl;
            cout << "\t- camadas" << endl;
            cout << "\t- nome do ficheiro para guardar os vrtices" << endl;
        }

    }else if(argv[1] == string("cone")){

        if(argc == 7){
            file.open(caminho + argv[6], std::fstream::out);
            cone(stof(argv[2]), stof(argv[3]), stof(argv[4]), stof(argv[5]) );
        }else{
            cout << "Faltam argumentos!" << endl;
            cout << "Os argumentos necessrios so:" << endl;
            cout << "\t- raio da base" << endl;
            cout << "\t- altura" << endl;
            cout << "\t- fatias" << endl;
            cout << "\t- camadas" << endl;
            cout << "\t- nome do ficheiro para guardar os vrtices" << endl;
        }

    }else{
        cout << "Figura invlida" << endl;
    }
}

    cout << "No foi dado nenhum argumento" << endl;
}

file.close();

return 0;
}

```

A.2 Código Motor

```
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#include <math.h>
#include "tinyxml/tinyxml.h"
#include <iostream>
#include <vector>
#include <fstream>
#include <cstring>
#include <sstream>

// para no estar sempre a escrever std::
using namespace std;

/* Ainda no usado
#define EXP 0
#define FPS 1
*/

// Vector que guarda a lista de ficheiros
std::vector<string> lista_ficheiros;
// Vector que guarda a lista das cores (1 para cada figura)
std::vector< pair<float, float> > lista_cores;

// flag para mudar o drwing mode
int flag_drawing_mode = 1;

// ngulos para "rodar a camera"
float alfa = 0.0f, beta = 0.0f, radius = 7.0f;
float camX, camY, camZ;

/* Ainda no usado
float dx = 0.0f;
float dy = 0.0f;
float dz = 0.0f;
int modo_camera = 0;
*/

/* Esta funcao vai buscar os nomes dos ficheiros .3d que esto no vector lista_ficheiros
 * Desenha todos os pontos de cada ficheiro e por ficheiro atribui uma cor do vector
 * lista_cores
 */
void desenha(void){

    for(int i=0; i<lista_ficheiros.size(); i++){

        const char *f = lista_ficheiros[i].c_str();
        ifstream fi(f);
        string str;

        /*
        float vermelho = lista_cores[i][0];
        float verde = lista_cores[i][1];
        float azul = lista_cores[i][2];
        */

        float verde = lista_cores[i].first;
        float azul = lista_cores[i].second;

        glColor3f(0, verde, azul);
        glBegin(GL_TRIANGLES);
```

```

        getline(fi, str);
        while (getline(fi, str)) {
            float v1, v2, v3;
            istringstream ss(str);

            ss >> v1;
            ss >> v2;
            ss >> v3;
            glVertex3f(v1, v2, v3);
        }
        glEnd();
    }
}

void cria_cores(int x){
    float vermelho=255, verde=255, azul=255;
    bool flag;

    //cout << x << endl;

    for(int i=0; i<x; i++){
        flag = true;
        while(flag) {
            vermelho = rand() % 255;
            vermelho = vermelho / 255;

            verde = rand() % 255;
            verde = verde / 255;

            azul = rand() % 255;
            azul = azul / 255;

            if (verde > 0 && verde < 1 && azul > 0 && azul < 1) {
                float arr[3] = {vermelho, verde, azul};
                lista_cores.push_back(make_pair(verde, azul));
                flag = false;
                //cout << "verde: " << verde << " | " << "azul: " << azul << endl;
            }
        }
    }
}

void spherical2Cartesian() {
    camX = radius * cos(beta) * sin(alfa);
    camY = radius * sin(beta);
    camZ = radius * cos(beta) * cos(alfa);
}

void changeSize(int w, int h) {
    // Prevent a divide by zero, when window is too short
    // (you cant make a window with zero width).
    if(h == 0)
        h = 1;

    // compute window's aspect ratio
    float ratio = w * 1.0 / h;

    // Set the projection matrix as current
    glMatrixMode(GL_PROJECTION);
    // Load Identity Matrix
    glLoadIdentity();

    // Set the viewport to be the entire window
    glViewport(0, 0, w, h);
}

```

```

    // Set perspective
    gluPerspective(45.0f ,ratio, 1.0f ,1000.0f);

    // return to the model view matrix mode
    glMatrixMode(GL_MODELVIEW);
}

void renderScene(void) {

    // clear buffers
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // set the camera
    glLoadIdentity();
    gluLookAt(camX, camY, camZ,
              0.0, 0.0, 0.0,
              0.0f, 1.0f, 0.0f);

    // put the geometric transformations here
    if(flag_drawing_mode == 0){
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    }else if(flag_drawing_mode == 1){
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    }else if(flag_drawing_mode == 2){
        glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
    }

    //glutWireTeapot(1);
    desenha();

    // End of frame
    glutSwapBuffers();
}

void processKeys(unsigned char c, int xx, int yy) {

}

void processSpecialKeys(int key, int xx, int yy) {

    switch (key) {
        case GLUT_KEY_RIGHT:
            alfa -= 0.1; break;

        case GLUT_KEY_LEFT:
            alfa += 0.1; break;

        case GLUT_KEY_UP:
            beta += 0.1f;
            if (beta > 1.5f)
                beta = 1.5f;
            break;

        case GLUT_KEY_DOWN:
            beta -= 0.1f;
            if (beta < -1.5f)
                beta = -1.5f;
            break;

        case GLUT_KEY_PAGE_UP:
            radius -= 0.1f;
            if (radius < 0.1f)
                radius = 0.1f;
            break;
    }
}

```



```

        case GLUT_KEY_PAGE_DOWN:
            radius += 0.1f;
            break;
    }
    spherical2Cartesian();
    glutPostRedisplay();
}

int le_xml(char *nome){
    string caminho = "xml/" + (string)nome;

    TiXmlDocument doc;

    if(!doc.LoadFile(caminho.c_str())){
        cout << "Nome do ficheiro invlido" << endl;
        return 1;
    }

    TiXmlNode* pRoot = doc.FirstChild();

    TiXmlElement* pListElement = pRoot->FirstChildElement("model");
    if (pListElement == NULL) return 0;

    while (pListElement != NULL){
        const char* nome_aux = NULL;

        nome_aux = pListElement->Attribute("file");
        if (nome_aux == NULL) return 0;

        /* S quando est em debug
        std::string nome_ficheiro = "../";
        */
        std::string nome_ficheiro = "";
        nome_ficheiro += nome_aux;

        lista_ficheiros.push_back(nome_ficheiro);

        //cout << nome_ficheiro << endl;

        pListElement = pListElement->NextSiblingElement("model");
    }

    cria_cores(lista_ficheiros.size());
    return 0;
}

void processMenuEvents(int option) {
    switch (option) {
        case 0 :
            flag_drawing_mode = 0;
            break;
        case 1 :
            flag_drawing_mode = 1;
            break;
        case 2 :
            flag_drawing_mode = 2;
            break;
        default:
            break;
    }

    glutPostRedisplay();
}

```

```

}

void createGLUTMenus() {

    int menu;

    menu = glutCreateMenu(processMenuEvents);

    glutAddMenuEntry("Fill",0);
    glutAddMenuEntry("Line",1);
    glutAddMenuEntry("Point",2);

    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

int main(int argc, char **argv) {

    // init GLUT and the window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH|GLUT_DOUBLE|GLUT_RGBA);
    glutInitWindowPosition(0,0);
    glutInitWindowSize(glutGet(GLUT_SCREEN_WIDTH),glutGet(GLUT_SCREEN_HEIGHT));
    glutCreateWindow("MOTOR");

    // Required callback registry
    glutDisplayFunc(renderScene);
    glutReshapeFunc(changeSize);

    // Callback registration for keyboard processing
    glutKeyboardFunc(processKeys);
    glutSpecialFunc(processSpecialKeys);

    // MENUS
    glutDetachMenu(GLUT_RIGHT_BUTTON);
    createGLUTMenus();

    // OpenGL settings
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    glClearColor(1,1,1,1);

    if(argc == 2){
        if(!xml(argv[1]) == 1){
            cout << "O ficheiro xml no foi encontrado" << endl;
        }
    }else{
        cout << "Nmero de argumentos invalido" << endl;
    }
    glutPostRedisplay();
    spherical2Cartesian();

    // enter GLUT's main cycle
    glutMainLoop();

    return 1;
}

```

Bibliografia

- <https://www.khronos.org/registry/OpenGL-Refpages/es3.0/>
- <http://www.cplusplus.com/>
- <https://elearning.uminho.pt/>