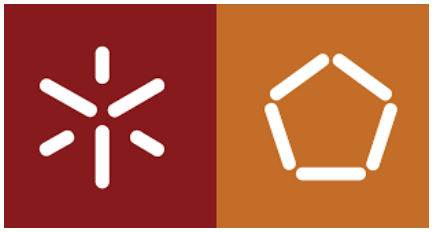

Universidade do Minho

Licenciatura em Engenharia Informática

Computação Gráfica

Trabalho Prático - Fase 1

José Freitas (A96140) Paula Marques (A90088)
Inês de Castro (A95458) Hugo Pereira (A93752)

8 de março de 2024

Índice

1 Introdução

Este trabalho, desenvolvido como parte da disciplina de Computação Gráfica, compreende duas partes principais Generator e a Engine.

A primeira parte, o Generator, tem como objetivo criar os pontos necessários para desenhar as várias primitivas gráficas (plano, caixa, esfera e cone).

Já a Engine, por sua vez, é responsável por representar visualmente as primitivas geradas anteriormente. Este relatório detalhará a metodologia e o raciocínio utilizados durante esta primeira fase do projeto.

2 Trabalho Desenvolvido

2.1 Generator

2.1.1 Plano

Para desenhar um plano fornecemos à função 2 parâmetros: número de divisões (*divisions*) e comprimento dos lados (*dimension*).

O número de divisões vai influenciar o tamanho do plano. Considerando o número de divisões o plano terá n x n quadrados. Cada quadrado constituído por dois triângulos.

```
float unit = dimension / divisions;
float half = dimension / 2.0f;

for (int x = 0; x < divisions; x++){
        for (int z = 0; z < divisions; z++){
                // Triangulo da Esquerda
                vertices.push_back(x * unit - half);
                vertices.push_back(0.0f);
                vertices.push_back(z* unit - half);

                vertices.push_back(x * unit - half);
                vertices.push_back(0.0f);
                vertices.push_back((z+1) * unit - half);

                vertices.push_back((x + 1) * unit - half);
                vertices.push_back(0.0f);
                vertices.push_back(z * unit - half);

                // Triangulo da Direita
                // ...
        }
}
```

2.1.2 Caixa

Por sua vez, a criação da caixa, do mesmo modo que o plano, recebe 2 parâmetros: número de divisões (*divisions*) e comprimento dos lados (*dimension*).

Assim sendo, assumimos que a caixa é a união de seis planos concorrentes, que representam as seis faces da caixa.

A caixa estará centrada na origem.

Como tal, o grupo usou o código do plano, alterando as devidas informações, de forma a criar a caixa.

2.1.3 Esfera

No que diz respeito à criação da esfera pede 3 parâmetros: o raio (*radius*), o número de camadas (*stacks*) e o número de fatias(*slices*). A esfera terá o centro na origem.

A esfera é dividida em n x m retângulos, cada um com 2 triângulos, qm que n é as *stacks* e m são as *slices*.

A esfera é constituída por um raio e dois ângulos, *phi* e *theta*.

```
float deltaPhi = (float)(M_PI) / stacks;
    float deltaTheta = (float)(2 * M_PI) / slices;

    for (int i = 0; i < stacks; i++) {
        float phi = i * deltaPhi;
        float nextPhi = (i + 1) * deltaPhi;

        for (int j = 0; j < slices; j++) {
            float theta = j * deltaTheta;
            float nextTheta = (j + 1) * deltaTheta;

            // Primeiro triangulo do quadrilatero
            vertices.push_back(radius * sin(phi) * cos(theta));
            vertices.push_back(radius * cos(phi));
            vertices.push_back(radius * sin(phi) * sin(theta));

            vertices.push_back(radius * sin(phi) * cos(nextTheta));
            vertices.push_back(radius * cos(phi));
            vertices.push_back(radius * sin(phi) * sin(nextTheta));

            vertices.push_back(radius * sin(nextPhi) * cos(theta));
            vertices.push_back(radius * cos(nextPhi));
            vertices.push_back(radius * sin(nextPhi) * sin(theta));

            // Segundo triangulo do quadrilatero
            //...
        }
    }
```

2.1.4 Cone

Para a criação do cone foram necessários 4 parâmetros: raio (*radius*), altura (*height)*, fatias (*slices*) e blocos (*stacks*).

O cone vai estar entrado na origem e a base no plano XZ. A sua realização foi dividida em duas partes: a base (circunferência) e a superfície lateral.

Primeiramente serão gerados os pontos para a base. A base é feita com triângulos no plano XZ, e todos os triângulos partilham o centro da circunferência (0,0,0). Quanto maior número de fatias, mais redonda ficará a circunferência na base. É calculado o ângulo entre cada slice (delta).

```
float delta = (2*M_PI) / slices;

//Bottom
    for(int i = 0; i < slices; i++)
    {
        float alfa = delta * i;

        vertices.push_back(0.0f);
        vertices.push_back(0.0f);          // centro
        vertices.push_back(0.0f);

        vertices.push_back(radius * sin(alfa + delta));
        vertices.push_back(0.0f);
        vertices.push_back(radius * cos(alfa + delta));

        vertices.push_back(radius * sin(alfa));
        vertices.push_back(0.0f);
        vertices.push_back(radius * cos(alfa));
    }
```

De seguida, é preciso construir os triângulos que formam a superfície lateral do cone. Estes triângulos são construídos de baixo para cima em blocos, diminuindo o raio e aumentando o y.

```
//Body
    for(int i = 0; i<stacks; i++)
    {
        float y = i * height / stacks;
        float next_y = (i + 1) * height / stacks;
        float r = radius * (1 - static_cast<float>(i) / stacks);
        float next_r = radius * (1 - static_cast<float>(i + 1) / stacks);
```

```
    for(int j = 0; j<slices; j++)
    {
        float alfa = delta * j;

        vertices.push_back(r * sin(alfa));
        vertices.push_back(y);
        vertices.push_back(r * cos(alfa));

        (...) // resto dos pontos

        vertices.push_back(next_r * sin(alfa + delta));
        vertices.push_back(next_y);
        vertices.push_back(next_r * cos(alfa + delta));
    }
}
```

2.2 Engine

Nesta segunda parte, foi desenvolvida uma aplicação capaz de ler e desenhar através da informação contida num ficheiro XML, onde está o código para obter os modelos a serem desenhados.

Para o parse desses ficheiros XML, que contém informações como a câmara, o tamanho da janela e os modelos, foi utilizada a biblioteca TinyXML, Após ter a certeza que o ficheiro é carregado com sucesso e que o elemento world encontra-se no mesmo, o código itera sobre os elementos do world usando um loop while. A cada iteração, o código verifica o nome do elemento e realiza ações com base nesse nome - camera, window e group.

Os modelos da aplicação gerados pelo *Generator* (formato 3d), foi implementado um algoritmo que lê o ficheiro e regista todos os pontos.

2.2.1 Testes

De modo a garantir a eficácia do programa criado, o grupo testou com os ficheiros de teste que foram disponibilizados no BlackBoard.

Os resultados, como poderão ser vistos nas figuras seguintes foram satisfatórios pois os resultados gerados foram idênticos aos esperados.

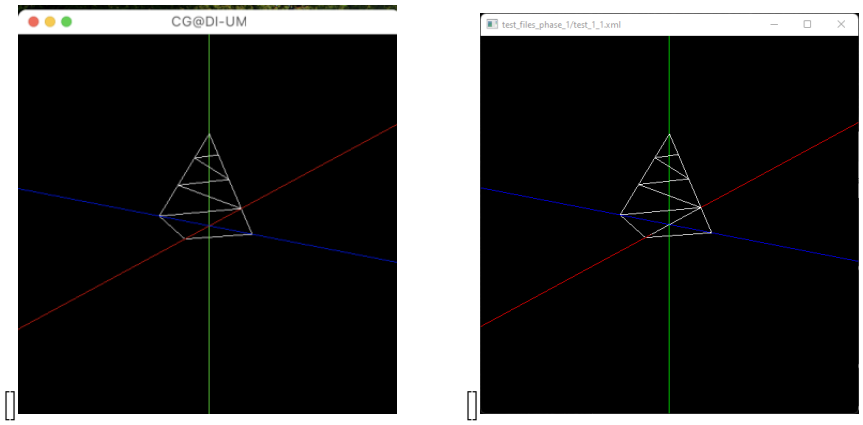

Figura 1: Teste 1.1 gerado e esperado, respetivamente.

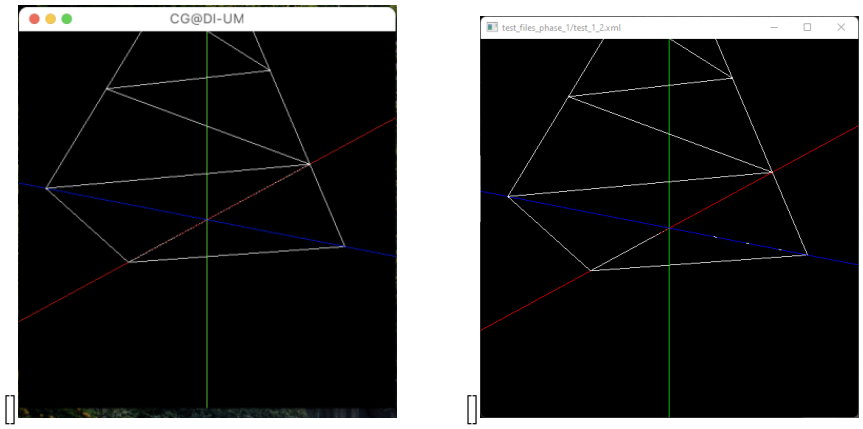

Figura 2: Teste 1.2 gerado e esperado, respetivamente.

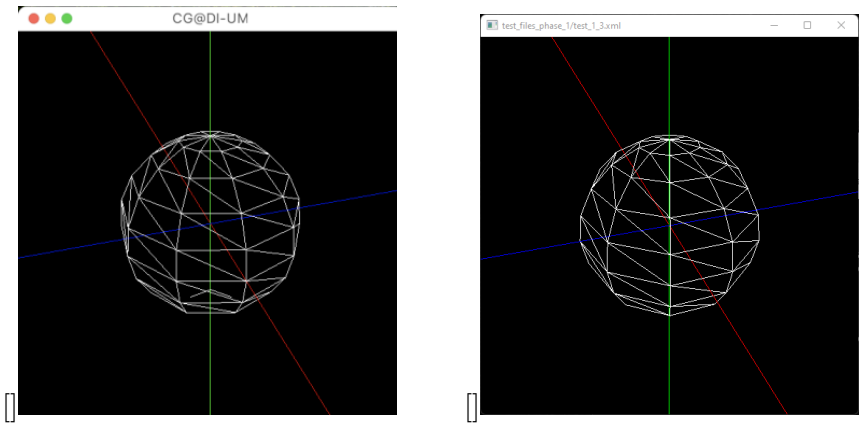

Figura 3: Teste 1.3 gerado e esperado, respetivamente.

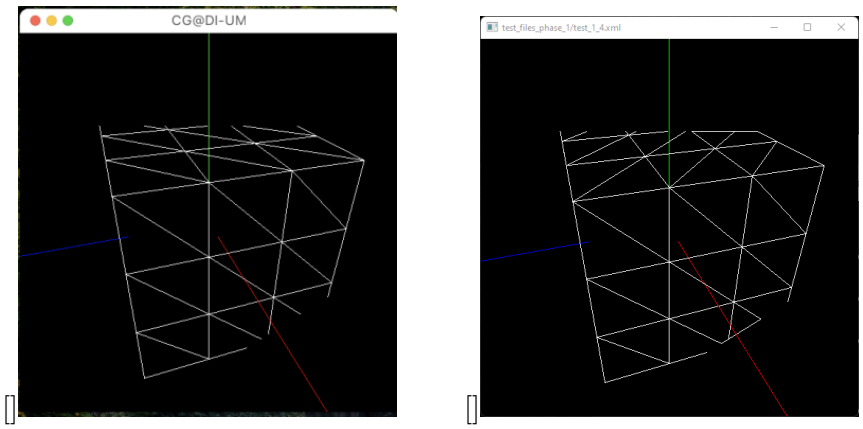

Figura 4: Teste 1.4 gerado e esperado, respetivamente.

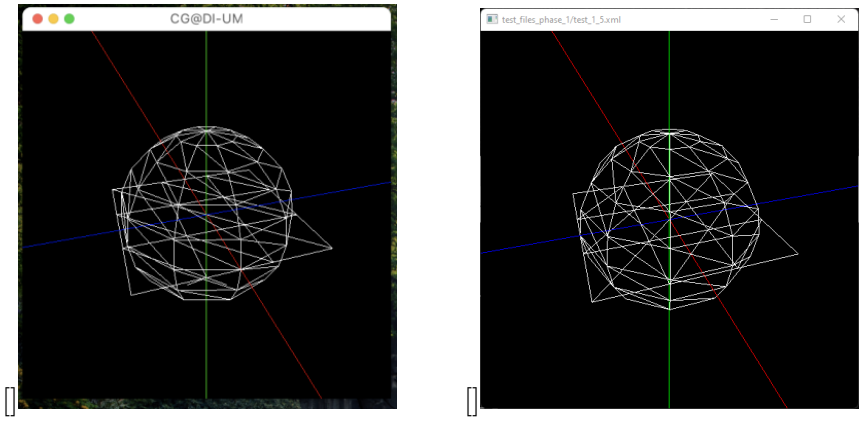

Figura 5: Teste 1.5 gerado e esperado, respetivamente.

3 Conclusão

Este trabalho prático permitiu ao grupo aplicar os conhecimentos adquiridos durante as aulas da disciplina de computação gráfica, possibilitando o aprofundamento do nosso conhecimento sobre o OpenGL e o C++. Concluímos que a primeira fase deste projeto foi um sucesso, uma vez que fomos capazes de cumprir os requisitos, resultando em duas fases: Engine e Generator.

Estamos confiantes que o sucesso desta primeira fase, ajudar-nos-á na execução das próximas fases.