Computação Gráfica

LEI - 3º ano - 2º semestre
Universidade do Minho

Trabalho Prático

Fase 1

Grupo 10

Trabalho realizado por:	*Número*
Duarte Moreira	A93321
Lucas Carvalho	A93176
Manuel Novais	A89512

Braga, 13 de março de 2022

Índice

1. Introdução

O presente relatório foi realizado no âmbito da Unidade Curricular de Computação Gráfica e tem como objetivo expor detalhadamente as decisões e abordagens tomadas pelo grupo durante a fase de desenvolvimento do projeto.

Este trabalho, dividido em 4 fases, tem como objetivo final a criação de um mini cenário baseado em gráficos 3D. Nesta primeira fase é-nos proposto o desenvolvimento de duas aplicações: uma para gerar ficheiros com a informação dos modelos - **generator** - e outra que lerá ficheiros de configuração, escritos em XML, e irá exibir os modelos criados - **engine**.

2. Gerador

A aplicação **generator** foi o ponto de partida do nosso trabalho. O objetivo desta é, para já, armazenar em ficheiro as coordenadas dos vértices necessários para a criação de um modelo dependendo dos parâmetros que recebe.

Nesta fase, o **generator** deverá reconhecer os seguintes comandos:

1. generator plane <lenght> <divisions> <filename.3d>
2. generator box <dimension> <grid division> <filename.3d>
3. generator sphere <radius> <slices> <stacks> <filename.3d>
4. generator cone <radius> <height> <slices> <stacks> <filename.3d>

A sua implementação resume-se à leitura dos parâmetros e dependendo destes, invocar a respetiva função de cálculo dos vértices. Este cálculo segue algoritmos que serão explorados nos capítulos seguintes. Quanto ao ficheiro em que são armazenados os pontos calculados, estes apresentam um estrutura muito simples em que cada linha apresenta informação sobre um vértice, com três números do tipo *float*, separados por um espaço em branco, sendo estes, a posição **x**,**y** e **z** do ponto em questão. Segue-se um exemplo:

```
-1 2 -1
-1 2 -0.333333
-0.333333 2 -0.333333
-0.333333 2 -0.333333
-0.333333 2 -1
-1 2 -1
```

Figura 2.1: Ficheiro 3D.

3. Motor

Já a segunda parte do trabalho consistiu na implementação do **engine**, uma aplicação que irá ler um ficheiro de configuração, escrito em XML, e posteriormente exibir os modelos lá contidos. Este ficheiro contém informações tais como a posição da câmara, o *lookAt point*, o *up vector* e a *projection*, assim como o nome dos modelos a serem lidos, criados com a aplicação anterior.

Para nos auxiliar na leitura do ficheiro XML, decidimos utilizar o *parser* **tinyXML2** devido à sua simplicidade. Como dito anteriormente, esta leitura fornece-nos o nome dos ficheiros .3d que necessitam de ser também interpretados. Para tal, foi criada uma estrutura de dados **Ponto** que armazena o valor das coordenadas de cada vértice, conforme a imagem seguinte:

```
struct Ponto {
    float x;
    float y;
    float z;
};
```

Figura 3.1: Estrutura de dados **Ponto**.

Dito isto, é necessário referir que, para a leitura dos modelos foi usada a biblioteca "vector.h" que nos permitiu armazenar os pontos lidos num vetor, uma estrutura de dados que tem o comportamento semelhante ao de um *array* dinâmico.

4. Primitivas Gráficas

4.1 Plano

Para a criação de um plano são passados como argumentos três parâmetros: o seu comprimento, o número de divisões ao longo de cada eixo e o nome do ficheiro 3d onde serão guardados os vértices calculados.

A estratégia utilizada para o seu desenho foi começar com uma das extremidades do plano - **ponto inicial** - e ir calculando as coordenadas dos quadrados mais pequenos (devido ao parâmetro que define o número de subdivisões) ao longo do eixo X. Assim que se atinge um determinado valor de x - **limite** - é incrementado o valor de z em **inc** unidades e retomado o valor de x para o inicial, construindo assim uma fila de quadrados (cada um composto por dois triângulos) iterativamente.

$$limite = \frac{size}{2}$$

$$inc = \frac{size}{divisions}$$

$$pontoinicial = (-limite, 0, -limite)$$

4.2 Caixa

Para a criação de um caixa são passados como argumentos três parâmetros: a sua dimensão, o número de divisões ao longo de cada eixo e o nome do ficheiro 3d onde serão guardados os vértices calculados.

A estratégia utilizada para o seu desenho foi bastante semelhante á do plano mas tendo agora em atenção alguns pormenores como por exemplo a base da caixa e o plano em cima descritos são observáveis de posições diferentes, o que implicou uma mudança na ordem de criação dos vértices. Tendo isto em mente, utilizou-se a estratégia em cima mencionada seis vezes, uma para cada face/plano da caixa.

4.3 Esfera

Para a criação de uma esfera são passados como argumentos quatro parâmetros: o seu raio, o número de slices/fatias, o número de stacks/camadas e o nome do ficheiro 3d onde serão guardados os vértices calculados.

A estratégia utilizada para o seu desenho foi começar por calcular dois ângulos: um que representa o ângulo entre cada fatia da esfera - alpha - e outro o ângulo entre o centro da esfera - (0,0,0) - e a primeira camada - beta.

$$alpha = \frac{2\pi}{slices}$$

$$beta = \frac{\pi}{stacks}$$

Com isto, faz se uso de dois ciclos *for*, um dentro do outro. O mais exterior percorre as camadas da esfera fazendo uso de duas variáveis auxiliares, **beta1** que representa o ângulo da camada atual e **beta2** o da seguinte. Já o ciclo mais aninhado vai incrementando o ângulo das fatias da esfera com o auxílio das variáveis **alpha1** que representa o ângulo da fatia atual e **alpha2** e da seguinte. É também dentro deste que se calculam as coordenadas de todos os pontos necessários para a construção da figura. Para o cálculo dos mesmos foram usadas as fórmulas que se seguem:

$$beta1 = beta * j$$

$$beta2 = beta * (j + 1)$$

$$alpha1 = alpha * i$$

$$alpha2 = alpha * (i + 1)$$

$$x = r * sin(b) * sin(a)$$

$$y = r * cos(b)$$

$$z = r * sin(b) * cos(a)$$

Onde **j** e **i** representam as variáveis que são incrementadas a cada iteração dos respetivos ciclos e **a|b** como sendo, ou **alpha1|beta1**, ou **alpha2|beta2**, dependendo do triângulo ou da ordem dos pontos que estamos a calcular.

4.4 Cone

Para a criação de um cone são passados como argumentos cinco parâmetros: o seu raio, a altura, o número de slices/fatias, o número de stacks/camadas e o nome do ficheiro 3d onde serão guardados os vértices calculados.

A estratégia utilizada para o seu desenho foi primeiramente desenhar a base, ou seja, um círculo. Para isso, calculamos o ângulo de cada fatia - **alpha** - e iniciamos um ciclo onde a cada iteração calculamos três vértices, que correspondem a uma fatia, de acordo com as seguintes fórmulas:

$$alpha = \frac{2\pi}{slices}$$

$$alpha1 = alpha * n$$
$$alpha2 = alpha1 + alpha$$

$$x = r * sin(a)$$
$$y = 0$$
$$z = r * cos(a)$$

Onde **n** representa a variável que é incrementada a cada iteração do ciclo e **a** como sendo, ou **alpha1**, ou **alpha2**, dependendo do ponto que estamos a calcular. De sublinhar que o primeiro ponto a ser calculado a cada iteração é o ponto (0,0,0), o centro da base.

Falta agora a construção do corpo do cone, para tal, são usados mais uma vez dois ciclos *for* um dentro do outro. O primeiro controla a altura da camadas com **camada1**, altura da camada atual e com **camada2**, altura da camada seguinte e também o raio das fatias a cada uma das camadas, que vai diminuindo progressivamente, onde **fatia1** é o raio da fatia atual e **fatia2** é o raio da fatia seguinte. Já no ciclo mais interior é regulado o ângulo das fatias com **alpha1** - ângulo da fatia atual - e **alpha2** - ângulo da fatia seguinte - e é onde se calculam as coordenados dos vértices com base nas fórmulas que se seguem:

$$camada = \frac{altura}{stacks}$$
$$camada1 = camada * i$$
$$camada2 = camada * (i + 1)$$

$$fatia1 = raio - (\frac{raio}{stacks} * i)$$
$$fatia2 = raio - (\frac{raio}{stacks} * (i + 1))$$

$$alpha1 = alpha * j$$
$$alpha2 = alpha * (j + 1)$$

$$x = fatia * sin(a)$$
$$y = cam$$
$$z = fatia * cos(a)$$

Onde **i** e **j** representam as variáveis que são incrementadas a cada iteração dos respetivos ciclos e **fatia|a|cam** como sendo, ou **fatia1|alpha1|camada1**, ou **fatia2|alpha2|camada2**, dependendo do triângulo ou da ordem dos pontos que estamos a calcular.

5. Funcionalidades extras

De forma a aumentar o desafio do trabalho, a equipa decidiu adicionar algumas funcionalidades extras às do enunciado, com o objetivo de alcançar um resultado mais interessante, único e um pouco mais interativo.

Estão então descritas abaixo as funcionalidades acrescentadas.

5.1 Cilindro

Para a criação de um cilindro são passados como argumentos quatro parâmetros: o seu raio, a altura, o número de slices/fatias e o nome do ficheiro 3d onde serão guardados os vértices calculados.

A estratégia utilizada para o seu desenho foi construir uma slice por inteiro de cada vez, ou seja, trabalhar com um ciclo *for* fazendo tantas iterações quanto o número de slices fornecido. A cada uma destas iterações serão calculados os pontos para desenhar 4 triângulos, um para a base, um para o topo e dois para o corpo. Usando os planos *y=0* e *y=altura*, **alpha** como sendo o ângulo de cada fatia e as seguintes fórmulas, conseguimos obter todos os pontos necessários.

$$alpha = \frac{2\pi}{slices}$$

$$x = raio * sin(i * alpha)$$

$$y = 0|altura$$

$$z = raio * cos(i * alpha)$$

Onde **i** representa a variável que é incrementada a cada iteração do ciclo. É também necessário referir que, a cada iteração, é armazenado o valor do novo vértice calculado para que possa ser reutilizado na iteração posterior.

5.2 Câmara

Foram adicionadas funções de leitura de inputs de forma a possibilitar o uso de comandos para a câmara.

O utilizador pode mudar a posição da câmara de maneira a que esta se mova na superfície de uma esfera olhando sempre para o centro. Para além disso este pode ainda aproximar e afastar a mesma.

5.3 Outros

Como comandos extra de forma a tornar o cenário mais apelativo para o utilizador e também para auxiliar a sua visualização é possível alternar os modos de preenchimento das figuras e ativar/desativar a visualização dos eixos.

6. Resultados Obtidos

6.1 Testes

Apresentam-se a seguir os resultados obtidos com as nossas aplicações de acordo com os ficheiros de testes fornecidos pela equipa docente.

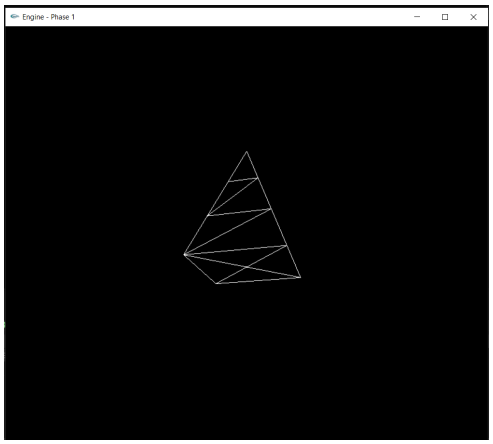

Figura 6.1: Resultado de *test_1_1.xml*

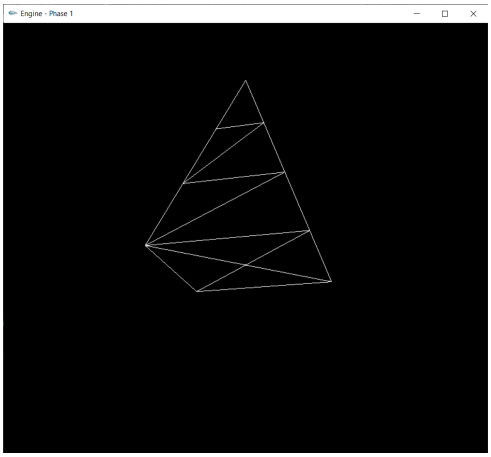

Figura 6.2: Resultado de *test_1_2.xml*

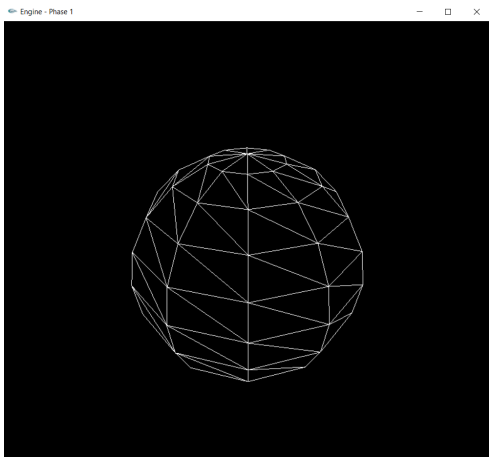

Figura 6.3: Resultado de *test_1_3.xml*

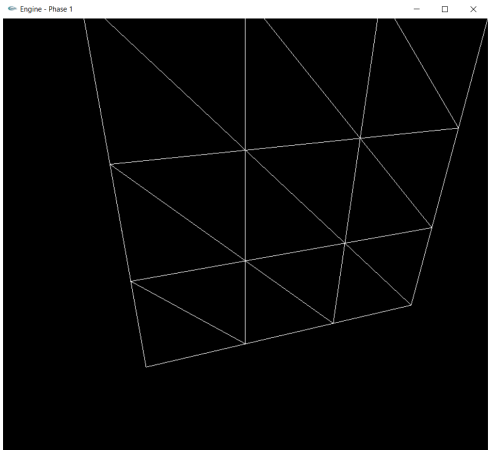

Figura 6.4: Resultado de *test_1_4.xml*

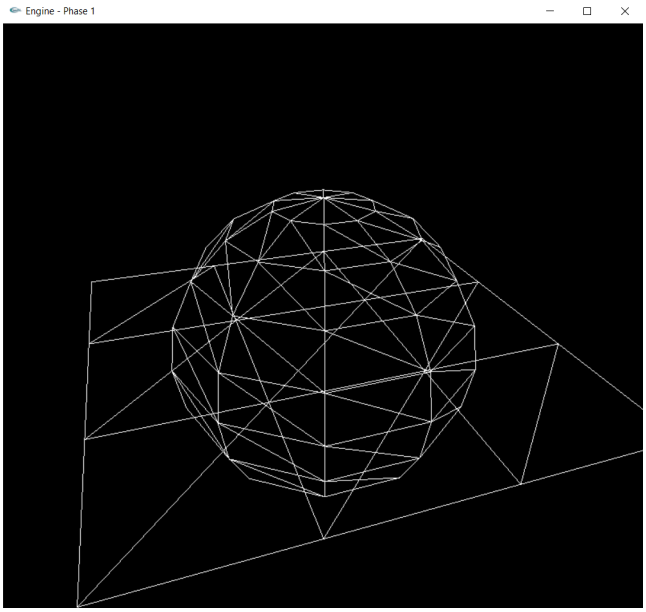

Figura 6.5: Resultado de *test_1_5.xml*

6.2 Cenários de demonstração

De seguida expõem-se algumas experiências de modelos efetuados pelo grupo de forma a tirar uma maior proveito e ter uma maior perceção do trabalho desenvolvido ao longo desta primeira fase.

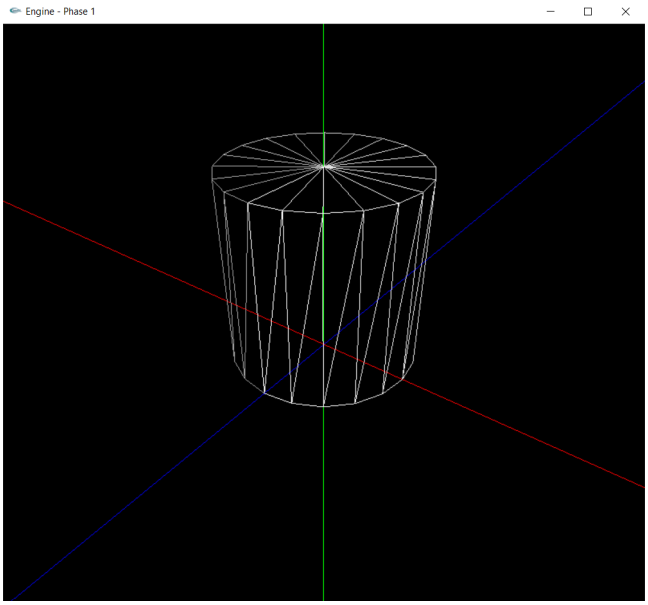

Figura 6.6: Cenário de demonstração 1.

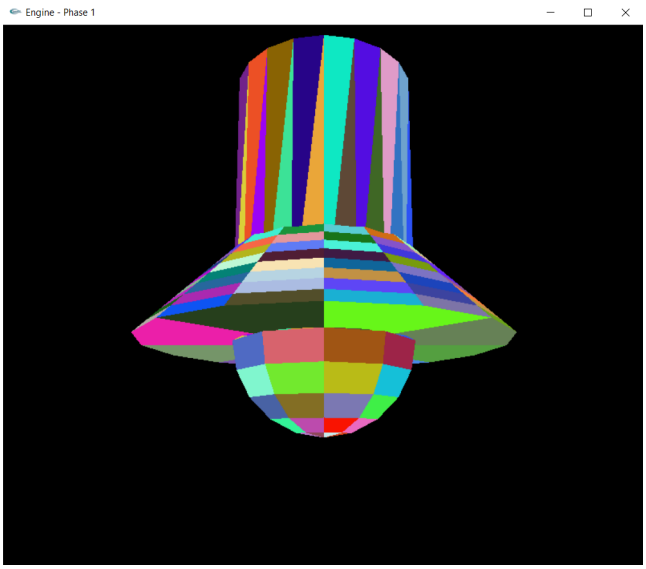

Figura 6.7: Cenário de demonstração 2.

7. Conclusão

Em jeito de conclusão sublinhamos que a realização desta primeira fase do trabalho foi bastante interessante e apelativa, uma vez que nos foi possível observar os resultados do nosso trabalho a cada etapa do mesmo.

Como é possível observar ao longo deste relatório, os requisitos fornecidos pelo enunciado são satisfeitos tanto ao nível do **generator** como ao nível do **engine** e também, como é proposto, o acrescento de algumas funcionalidades extras que esperamos poder explorar ainda mais nas fases seguintes.

Por fim, o grupo considera que esta fase foi importante para desenvolver e aperfeiçoar os conhecimentos abordados nas aulas práticas no que toca à utilização da API **OpenGL** e da linguagem de programação **C++**.