



UNIVERSIDADE DO MINHO
Departamento de Informática

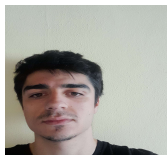
COMPUTAÇÃO GRÁFICA

Phase 1 - Graphical Primitives

Grupo 37

Feito por:

Dinis Gonçalves Estrada (A97503)
Emanuel Lopes Monteiro da Silva (A95114)



A97393



A95114

March 18, 2023
Ano Letivo 2022/23

Conteúdo

1	Introdução	2
2	Objetivo do trabalho	2
3	Arquitetura	3
4	Utils	3
4.1	.hpp	3
4.2	.cpp	3
5	Generator	4
5.1	Comandos	4
5.2	Cálculo dos Pontos	4
5.2.1	Plane	5
5.2.2	Box	6
5.2.3	Sphere	7
5.2.4	Cone	9
5.3	Criação e Escrita nos ficheiros	11
6	Engine	12
6.1	Comando	12
6.2	Leitura do Ficheiro	12
6.3	Estrutura de dados	12
6.4	Desenho das figuras	12
6.5	Output	12
6.5.1	Generator	12
6.5.2	Engine	13
7	Conclusão	13

1 Introdução

No âmbito da unidade curricular de Computação Gráfica foi nos proposto um trabalho prático que será desenvolvido em quatro fases. Nesta primeira fase foi necessário utilizar a ferramenta OpenGL utilizando C/C++ como a linguagem de programação que servirá de base a este projeto. Para que tal possa ser bem sucedido, o grupo teve de aplicar o conhecimento adquirido ao longo das várias aulas práticas e teóricas.

2 Objetivo do trabalho

Na primeira fase foi então desafiado ao grupo que representassem 4 primitivas gráficas, sendo elas um plano, uma caixa, um círculo e um cone que terão de ser calculados de acordo com vários parâmetros como a altura, a largura, o raio e as divisões. Para que seja possível representar estas formas será necessário criar vários triângulos a partir do cálculo dos seus vértices.

Com isto, foi necessário criar duas aplicações distintas sendo elas um generator e um engine. Sendo que o generator irá calcular todos os vértices dos triângulos e guardá-los num ficheiro .3d, enquanto que o engine irá interpretar um dado ficheiro XML que para além de conter as características que a câmara terá também terá o nome do ficheiro .3d, criado pelo generator, com os vértices/pontos criados de forma a gerar os triângulos que por sua vez irão gerar as formas geométricas.

3 Arquitetura

Tal como pedido, o programa encontra-se dividido em duas aplicações: “Generator” e “Engine”. Para tal temos dois packages, onde cada um deles contém ficheiros responsáveis pelas funcionalidades de cada uma das aplicações, além do ficheiro (Generator/generator.cpp e Engine/engine.cpp) que contém a função main que se ocupa, com base nos argumentos dados pelo utilizador, de realizar as respetivas ações. Existe também o package “Utils” que está encarregue de struct, funções que o Engine e o Generator têm em comum. Desta forma, evitamos a repetição de código e redundância, obtendo assim um código mais organizado e limpo. O package tinyXML foi descarregado do website <https://sourceforge.net/projects/tinyxml/>, de forma a length/2iliar a leitura de ficheiros XML. Estes packages encontram-se todos incluídos no package src.

Além disso, existe também o package build, que contém todos os executáveis criados pela MakeFile gerada pela CMakeList.txt e existe ainda um package out que guarda todos os ficheiros .3d gerados.

4 Utils

4.1 .hpp

No header utils.hpp, foi definida uma estrutura point, que contem as coordenadas float x, y e z que definem um dado ponto no espaço, e uma classe figure constituída por um vetor de points, sendo esta a estrutura a partir da qual será obtida a informação necessária para desenhar as várias figuras. O vetor nela contido é de acesso público, visto que o objetivo do uso da figure não é encapsular estes dados, mas sim facilitar a adição de novos pontos ao vetor através da função addPoint, instanciada também neste header e definida em utils.cpp.

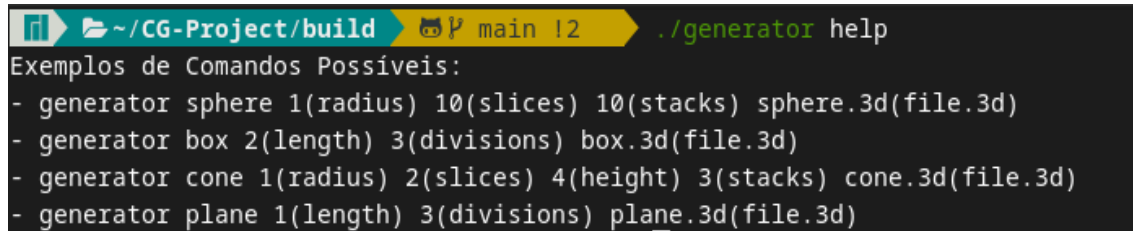
4.2 .cpp

A função addPoint é uma função auxiliar que cria um point partindo das suas coordenadas x, y e z, e acrescenta-o ao vetor da classe figure. Trata-se de uma função auxiliar usada frequentemente na criação de novos vetores de pontos (figures). Ao defini-la enquanto uma operação que atua dentro de uma classe, ao invés de receber e devolver um novo vetor, tornamos a sua implementação no código mais simples e levamos a que este seja muito menos extenso e repetitivo.

5 Generator

5.1 Comandos

Para que possa ser mais fácil o uso desta aplicação existe um comando help que disponibiliza os possíveis comandos.



```
~/CG-Project/build main 12 ./generator help
Exemplos de Comandos Possíveis:
- generator sphere 1(radius) 10(slices) 10(stacks) sphere.3d(file.3d)
- generator box 2(length) 3(divisions) box.3d(file.3d)
- generator cone 1(radius) 2(slices) 4(height) 3(stacks) cone.3d(file.3d)
- generator plane 1(length) 3(divisions) plane.3d(file.3d)
```

Figure 1: Menu Help

5.2 Cálculo dos Pontos

Na parte do Generator podemos encontrar um ficheiro .cpp que terá todas as funções relacionadas com o cálculo dos pontos dos triângulos de forma representar as figuras que o utilizador pretender. Estes pontos são inicialmente guardados numa struct denominada por figure que mais tarde será interpretada e guardaremos os seus pontos num ficheiro .3d para que o engine o possa ler.

5.2.1 Plane

A função presente no generator que gera os pontos dos triângulos para representar um plano denomina-se por `calcPlane` e recebe como parâmetros o tamanho das arestas do plano e o número de divisões em que este plano será dividido.

Para calcular o comprimento de cada triângulo começamos por dividir o número de divisões fornecido pelo tamanho fornecido pelo utilizador. Como este plano estará sempre no plano xz o y será sempre 0 e por isso apenas incrementamos ou subtraímos ao x e ao z de forma a que o centro do plano fique sempre no meio do referencial.

Após o cálculo de cada ponto dos triângulos, a interpretação destes pelo engine corresponde aos exemplo seguinte:

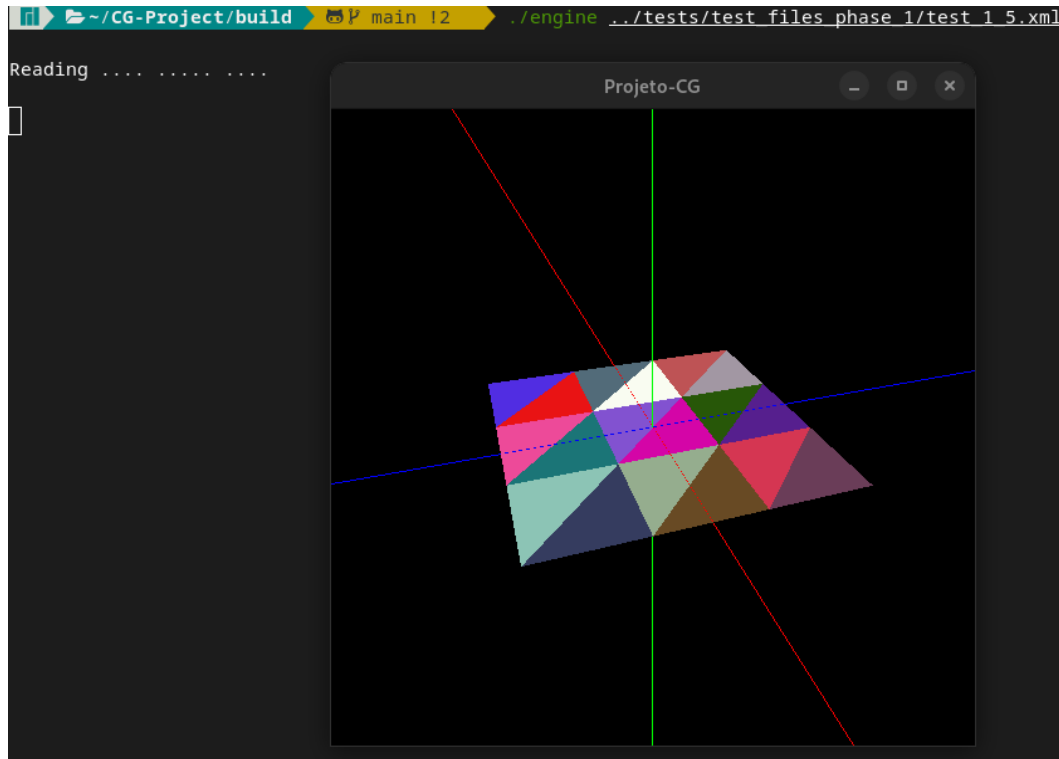


Figure 2: Plane com 2 lenght e 3 divisions

5.2.2 Box

A função presente no generator que gera os pontos dos triângulos para representar uma box denomina-se por `calcBox` e recebe como parâmetros o tamanho das arestas da box e o número de divisões que cada face terá. Para calcular todos estes triângulos dividimos o calculo em 6 partes correspondentes a cada face.

Após o cálculo de cada ponto dos triângulos, a interpretação destes pelo engine corresponde aos exemplo seguinte:

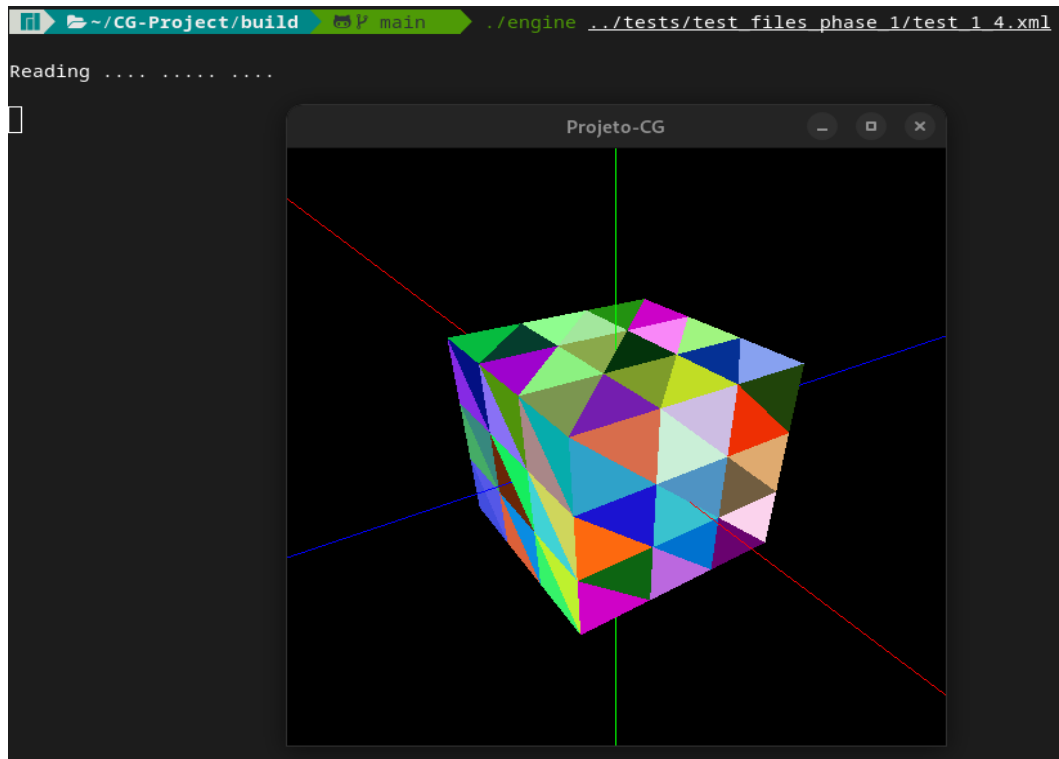


Figure 3: Box com 2 de length e 3 divisions

5.2.3 Sphere

A função presente no generator que gera os pontos dos triângulos para representar uma sphere denomina-se por calcSphere e recebe como parâmetros o raio da esfera, o número de stacks e o número de slices.

Para calcular todos estes triângulos usamos os seguintes intervalos e cálculos para o (x y z)

- $-\pi_2 \leq \phi \leq \pi_2$ (intervalo de tamanho π)
- $0 \leq \theta \leq 2 \times \pi$ (intervalo de tamanho $2 \times \pi$)
- $\delta\phi = \frac{\pi}{numStacks}$
- $\delta\theta = \frac{2 \times \pi}{numSlices}$
- $x = raio \times \cos(\phi) \times \sin(\theta)$
- $y = raio \times \sin(\phi)$
- $z = raio \times \cos(\phi) \times \cos(\theta)$

De acordo com a seguinte imagem e os intervalos e pontos acima conseguimos então estabelecer uma linha de raciocínio de forma a chegar ao resultado esperado:

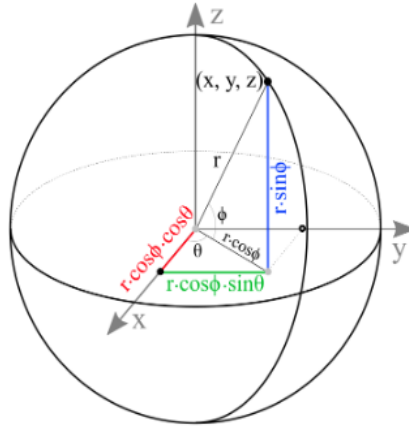


Figure 4: Instruções do cálculo dos pontos

Assim, para cada secção é só criar dois triângulos (6 vértices) que formam esse pequeno plano, tendo em conta que a cada iteração de i , o ângulo ϕ aumenta $\delta\phi$ (logo $\phi = -\frac{\pi}{2} + i \times \delta\phi$). Já a cada iteração de j , o ângulo θ aumenta $\delta\theta$ (logo $\theta = j \times \delta\theta$)

Após o cálculo de cada ponto dos triângulos, a interpretação destes pelo engine corresponde aos exemplo seguinte:

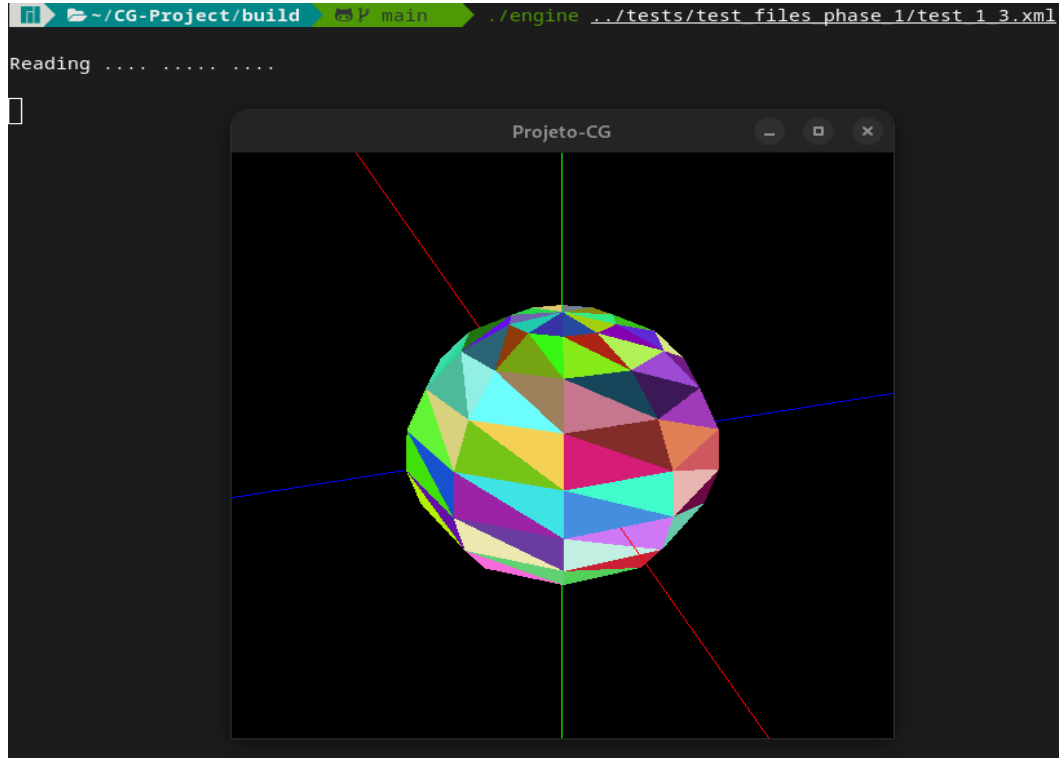


Figure 5: Sphere com raio 1, 10 stacks e 10 slices

5.2.4 Cone

A função cone recebe como parâmetros o raio (radius), a altura (height), o número de fatias (slices) e o número de camadas (stacks) e com estes valores calcula todos os vértices necessários para desenhar um cone com as medidas dadas. O argumento stacks define o número de “cortes” horizontais da base até ao bico, enquanto que as slices vão determinar o numero de divisões da base em partes iguais que, por sua vez, estabelecem o nível de curvatura do seu “círculo”. A figura que representa a base do cone será sempre um polígono regular com slices lados, ou seja, quanto maior for este valor mais parecido com um círculo se tornará ao observador.

Em relação à sua posição no referencial, foi definido que a sua base se situa no plano xOz e que o cone se encontra centrado na origem em relação aos eixos x e z.

Para desenhar o sólido, foram seguidos os seguintes passos:

a base do cone pode ser desenhada a partir de um triângulo isósceles, com um vértice na origem, que roda em torno do eixo y. Assim, a partir do cálculo do angulo δ (angulo entre os dois pontos da base do triângulo) e do raio podem ser calculados todos os vértices do polígono pelas equações abaixo:

$$x = \text{radius} \times \sin(\theta)$$

$$z = \text{radius} \times \cos(\theta)$$

$$\theta = i \times \delta$$

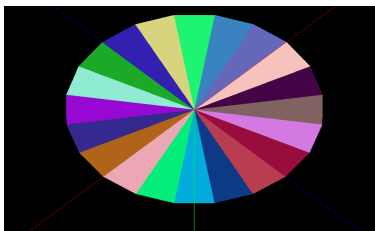


Figure 6: Base do cone de raio 1 com 20 slices

Para calcular todos os vértices das faces laterais do cone, para cada fatia foi-se diminuindo o raio e aumentando o valor de y de acordo com o número de stacks

Após o cálculo de cada ponto dos triângulos, a interpretação destes pelo engine corresponde aos exemplo seguinte:

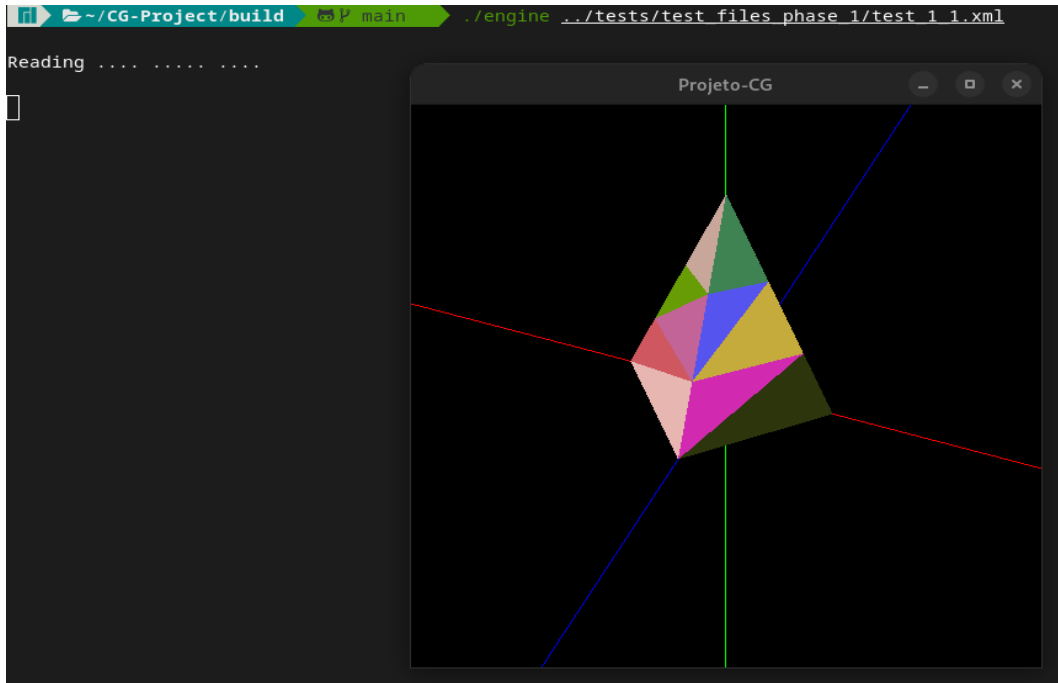


Figure 7: Cone com raio 1, 2 slices, 4 height e 3 stacks

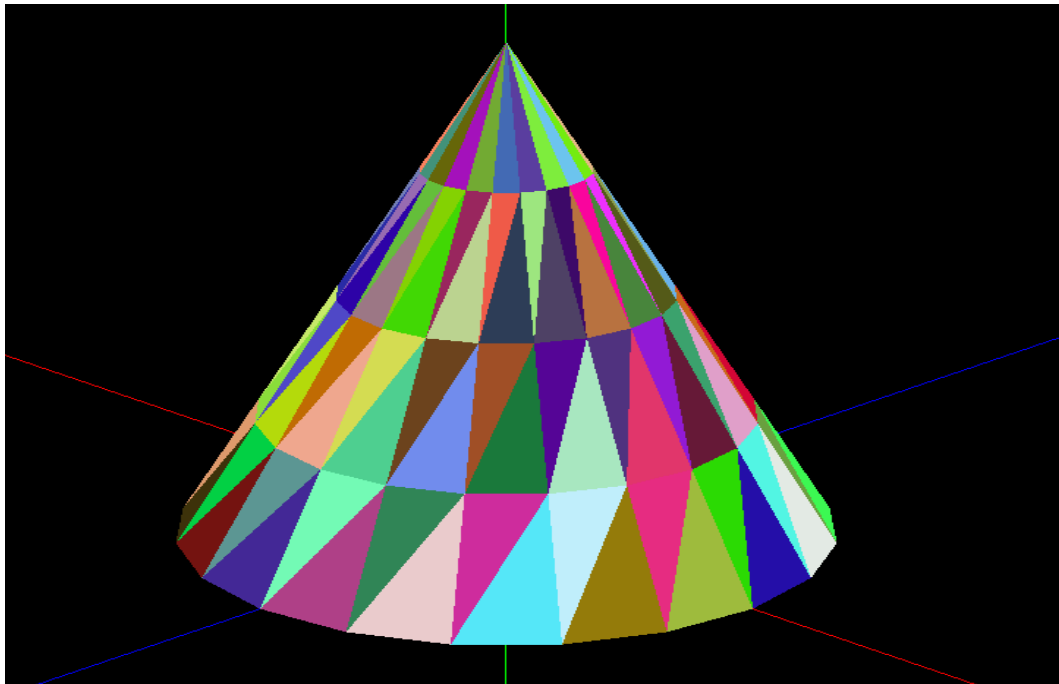


Figure 8: Cone de raio 1, 20 slices, 2 de altura e 4 stacks

5.3 Criação e Escrita nos ficheiros

Com o cálculo dos pontos terminado pelas funções anunciadas acima, estes são por fim armazenados num ficheiro .3d, onde cada linha representa um ponto com as coordenadas (x y z) tal como podemos ver no exemplo abaixo:

```
≡ sphere_1_10.3d x
home > emanuel > CG-Project > out > ≡ sphere_1_10.3d
1  p -0.951057 0.309017
2  -0 -1 -4.37114e-08
3  -2.56929e-08 -1 -3.53633e-08
4  0 -0.951057 0.309017
5  -2.56929e-08 -1 -3.53633e-08
6  0.181636 -0.951057 0.25
7  0.181636 -0.951057 0.25
8  -2.56929e-08 -1 -3.53633e-08
9  -4.1572e-08 -1 -1.35076e-08
10 0.181636 -0.951057 0.25
11 -4.1572e-08 -1 -1.35076e-08
12 0.293893 -0.951057 0.0954915
13 0.293893 -0.951057 0.0954915
14 -4.1572e-08 -1 -1.35076e-08
15 -4.1572e-08 -1 1.35076e-08
16 0.293893 -0.951057 0.0954915
17 -4.1572e-08 -1 1.35076e-08
18 0.293893 -0.951057 -0.0954915
19 0.293893 -0.951057 -0.0954915
20 -4.1572e-08 -1 1.35076e-08
21 -2.56929e-08 -1 3.53633e-08
22 0.293893 -0.951057 -0.0954915
23 -2.56929e-08 -1 3.53633e-08
24 0.181636 -0.951057 -0.25
25 0.181636 -0.951057 -0.25
26 -2.56929e-08 -1 3.53633e-08
27 3.82137e-15 -1 4.37114e-08
28 0.181636 -0.951057 -0.25
29 3.82137e-15 -1 4.37114e-08
30 -2.70151e-08 -0.951057 -0.309017
31 -2.70151e-08 -0.951057 -0.309017
32 3.82137e-15 -1 4.37114e-08
```

Figure 9: Ficheiro .3d

6 Engine

6.1 Comando

Após o utilizador executar o generator e este gerar o ficheiro 3d pretendido poderá agora executar o "engine" passando como argumento o caminho para o ficheiro que se pretende que seja exibido. Assume-se que os modelos indicados no ficheiro XML já existem e foram gerados.

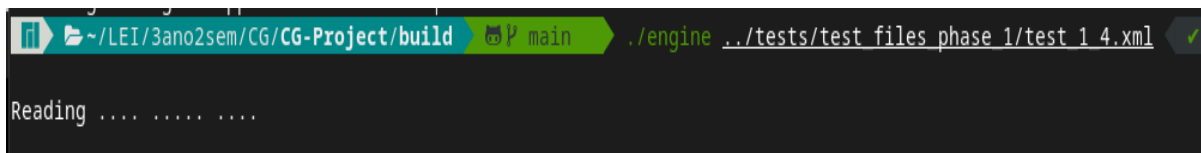
A terminal window with a dark background. The top bar shows the file explorer icon, the path ~/LEI/3ano2sem/CG/CG-Project/build, a green bar with a cat icon and the text 'main', and the command './engine ../tests/test files phase 1/test 1 4.xml' followed by a green checkmark icon. The terminal output shows 'Reading'.

Figure 10: Comando para correr o engine

6.2 Leitura do Ficheiro

Quando é feita a leitura do ficheiro XML que foi passado como argumento, é feita uma leitura primeiramente ao elemento "window" que contém as informações da janela, largura e altura. Seguidamente é feita uma leitura ao elemento "camera" que contém informação, neste caso coordenadas, sobre a posição da camera, no elemento "position", sobre para onde está apontada, no elemento "lookAt", e sobre o vetor up, no elemento "up", além disso também possui informações sobre a "fov", o "near" e o "far" no elemento "projection". Por último é feita uma leitura ao elemento "models", em que se abre cada um dos ficheiros .3d, e começa-se a fazer uma leitura de linha em linha. Sendo que cada linha representa um Ponto, guardamos todos esses pontos num vetor da struct Figure.

6.3 Estrutura de dados

Após a leitura do ficheiro, guardamos toda a informação lida numa estrutura de dados "World". Esta estrutura possui dois campos do tipo float para a "width" e "height" da janela, três float arrays para guardar as coordenadas da camera position, camera lookAt, camera up e um float array para guardar a projection; possui também um Map<Key,Value>, onde a key é um número inteiro de acordo com a ordem que a primitiva se encontrava no ficheiro XML, e o value é a Figure.

6.4 Desenho das figuras

O desenho das figuras é feito usando as funções definidas no namespace draw em world.cpp e world.hpp. Para tal, é utilizada a função drawFigure, que lê o vetor de pontos contido numa dada figure, selecionando cada três pontos dados por ordem e aplicando a função auxiliar drawTriangle, que irá desenhar um triângulo de uma cor aleatória partindo das coordenadas desses mesmos pontos. Deste modo, todos os triângulos de uma dada figura terão cores diferentes, o que torna a forma obtida mais distinguível.

6.5 Output

6.5.1 Generator

O output do gerador serve apenas para confirmar ao utilizador se os ficheiros pretendidos foram criados com sucesso.

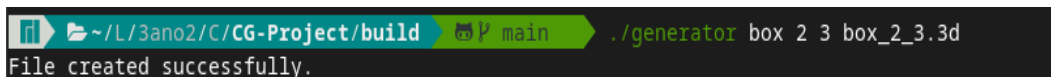
A terminal window with a dark background. The top bar shows the file explorer icon, the path ~/L/3ano2/C/CG-Project/build, a green bar with a cat icon and the text 'main', and the command './generator box 2 3 box_2_3.3d'. The terminal output shows 'File created successfully.'.

Figure 11: Comando para correr o engine

6.5.2 Engine

O output do engine indica ao utilizador quando o ficheiro .xml não foi lido com sucesso ou, caso a leitura seja bem sucedida, é aberta uma nova janela do OpenGL com o desenho das figuras pretendidas. Implementamos a possibilidade de poder rodar a câmara usando as setas do teclado; de poder aproximar e afastar a câmara da figura usando as teclas "page up" e "page down"; e ainda implementamos a possibilidade de mostrar a figura pela representação por linhas, pressionando a tecla "F1", e a representação por triângulos cheios, representação por default, pressionando a tecla "F2". Usamos a função "processSpecialKeys" para conseguirmos implementar todas as funcionalidades anteriormente descritas.

7 Conclusão

Através do desenvolvimento deste trabalho pudemos aplicar os conhecimentos adquiridos nas aulas da unidade curricular de Computação Gráfica, e aprofundar a nossa compreensão do modo de funcionamento do OpenGL e da linguagem C++. Para além disto, permitiu introduzir-nos a "markup languages", neste caso o XML, e trabalhar com a leitura e escrita de ficheiros deste tipo. Consideramos que desenvolvemos um bom trabalho, que implementa todas as funcionalidades exigidas nesta fase do trabalho. A abordagem utilizada poderá, no entanto, vir a sofrer alterações mediante os requisitos das fases seguintes.