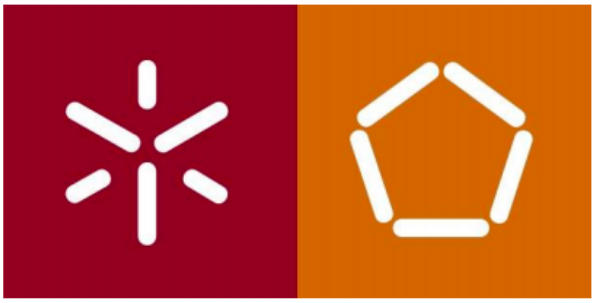

Universidade do Minho

Mestrado Integrado em Engenharia Informática

06/03/2020

Computação Gráfica
Fase 1

António Gonçalves (A85516)
João Fernandes (A84034)
Eduardo Conceição (A83870)
Rita Rosendo (A84475)

Contents

1 Introdução

Neste relatório iremos explicar a nossa abordagem aos problemas que nos foram propostos no âmbito da cadeira de Computação Gráfica. Começaremos por analisar o problema, descrevendo possíveis estratégias para o resolver e os principais objetivos a atingir.

De seguida, iremos explicar como o fizemos e qual o nosso pensamento para essas escolhas, com a apresentação de exemplos gerados pelo nosso programa.

Por fim, iremos discutir o produto final bem como as principais dificuldades com que nos deparamos enquanto desenvolvíamos o projeto.

2 Análise do Problema

Nesta fase do projeto foi-nos pedida a elaboração de dois programas: um gerador que guarda em ficheiros de texto as informações sobre os modelos criados, que, neste caso, se refere aos vértices que compõem os modelos, e um *engine* que irá ler os ficheiros de configuração (neste caso iremos utilizar ficheiros escritos em XML) desenhando a partir destes os modelos respetivos.

Analisando estes dois componentes mais especificamente temos:

-Gerador: Deverá conseguir gerar os ficheiros com os vértices necessários para construir os modelos de um plano, uma caixa, uma esfera e um cone, que serão explicados de seguida.

-Plano: Quadrado desenhado no plano XZ, com o comprimento das arestas fornecido e feito com 2 triangulos.

-Caixa: Irá necessitar das suas dimensões (X, Y e Z). Para além disto, será dividido em secções.

-Esfera: Precisa de receber o raio e o número de *slices* e *stacks*. Estes servirão para dividir a superfície esférica para obter os vértices para o desenho do modelo.

-Cone: Necessita do raio da base, da altura e do número de *slices* e *stacks*.

***-Engine*:** Como foi dito anteriormente, o *engine* deverá ler os ficheiros XML, que contêm o nome do ficheiro com os vértices do modelo pretendido, que, após serem lidos, serão usados para desenhar os triângulos necessários para finalizar a figura.

3 Resolução

Uma vez que todo o trabalho trata da leitura, escrita e representação de pontos 3D, decidimos criar uma estrutura de dados, Point, para armazenar essa informação, que neste caso deverá guardar os valores das coordenadas do x, do y e do z de cada ponto.

3.1 Gerador

Para o caso do plano e da caixa fazemos sempre dois triângulos a mais por quadrado pois o engine usa *culling*.

3.1.1 Plane

Para desenhar o plano será necessário calcular os seus vértices.

Usando um sistema cartesiano e o comprimento fornecido temos quatro vértices, A(aresta/2,0,-aresta/2), B(-aresta/2,0,-aresta/2), C(-aresta/2,0,aresta/2) e D(aresta/2,0,aresta/2). Finalmente, usamos 4 triangulos para fazer o plano, T1(D,A,B), T2(B,C,D), T3(B,A,D) e T4(D,C,B).

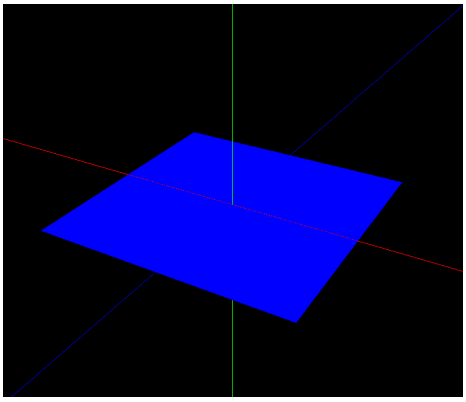

Figure 1: Representação do Plano.

3.1.2 Box

Para desenhar a caixa será necessário calcular os seus vértices.

XXXXXXXXXXXXXXX

Usando um sistema cartesiano e o comprimento fornecido temos oito vértices para a parte exterior da caixa, A(X,0,0), B(0,0,0), C(0,0,Z), D(X,0,Z), E(X,Y,0), F(0,Y,0), G(0,Y,Z) e H(X,Y,Z).

Para fazer as faces exteriores das caixas temos de fazer quatro triângulos para cada usando os vértices anteriores.

Base(DAB,BCD,BAD,DCB), Topo(HEF,FGH,FEH,HGF),

Frente(DHG,GCD,GHD,DCG), Costas(AEF,FBA,FEA,ABF),
Direita(DAE,EHD,EAD,DHE) e Esquerda(BFG,GCB,GFB,BCG).

Para construir as divisões internas da caixa fazemos fatias paralelas aos eixos Z e Y cuja distância entre si é $\frac{x}{n^{o}_{divisoes}}$, sendo constituídas por 4 triângulos cada.

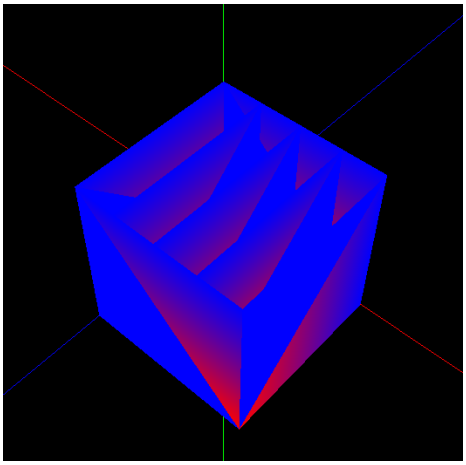

Figure 2: Representação da Caixa.

3.1.3 Sphere

Para desenhar a esfera será necessário calcular os seus vértices. Para isto iremos usar coordenadas esféricas.

Assim, consideramos 2 ângulos, α e β, para facilitar a representação dos pontos. O ângulo α irá variar entre $-\frac{\pi}{2}$ e $\frac{\pi}{2}$, sendo este usado para definir a *stack* em que o ponto de encontra, ou seja, a altura do vértice. O ângulo β irá variar de 0 até 2π, sendo este usado para definir onde se encontra o vértice nessa *stack*, ou seja, em qual *slice* se encontra.

Uma vez que estes ângulos irão servir para dividir a esfera em partes, tendo em conta o número de *stack* e de *slices* pretendidos na esfera, podemos afirmar que cada *slice* terá $\frac{2\pi}{slices}$ de diferença. De igual forma, as *stacks* terão $\frac{\pi}{stacks}$ de diferença entre elas.

Desta forma, uma vez que entre cada vértice só irá variar o valor dos ângulos β e α, podemos definir as coordenadas de cada ponto a partir do raio r e deles:

$$X = r * cos(\alpha) * sin(\beta)$$
$$Y = r * sin(\beta)$$
$$Z = r * cos(\alpha) * cos(\beta)$$

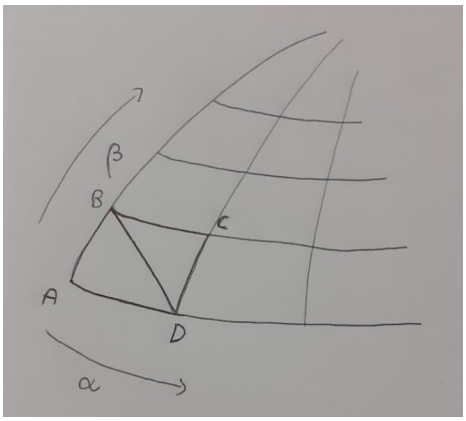

Figure 3: Representação de uma subsecção da esfera

Tendo isto em conta, em cada secção temos dois triângulos que, seguindo o esquema da figura 1, são constituídos pelos vértices ADB e BDC, por esta ordem.

Uma vez que em cada secção temos definido um ângulo *a* e *b*, e que temos, portanto, um ângulo *aNext* e *bNext*, os vértices são definidos da seguinte forma:

	x	y	z
A	$r * cos(\alpha) * sin(\beta)$	$r * sin(\alpha$	$r * cos(\alpha) * sin(\beta)$
B	$r * cos(\alpha) * sin(\beta n)$	$r * sin(\alpha)$	$r * cos(\alpha) * cos(\beta n)$
C	$r * cos(\alpha n) * sin(\beta n)$	$r * sin(\alpha n)$	$r * cos(\alpha n) * cos(\beta n)$
D	$r * cos(\alpha n) * sin(\beta)$	$r * sin(\alpha n)$	$r * cos(\alpha n) * cos(\beta)$

Assim, podemos entender que o ângulo *b* varia entre $-\frac{\pi}{2}$ e $\frac{\pi}{2}$, enquanto que o ângulo *a* varia entre 0 e 2π. Desta forma, uma vez que a trigonometria da esfera está definida, percorrendo cada *slice* e cada *stack*, obtemos o resultado final.

Aqui demonstramos uma esfera gerada desta forma (depois do ficheiro passar pelo *Engine*, a explicar à frente) com 1 de raio, 10 *stacks* e 10 *slices*:

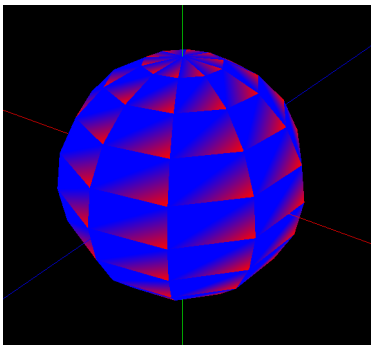

Figure 4: Representação de uma esfera.

3.1.4 Cone

Tal como na esfera, para desenhar o cone teremos de calcular os vértices da figura utilizando coordenadas esféricas. Para isso, consideramos o ângulo α, que varia entre 0 e 2π, que representa a *slice* do vértice. A diferença de cada *stack* será dada por $\frac{altura}{stacks}$.

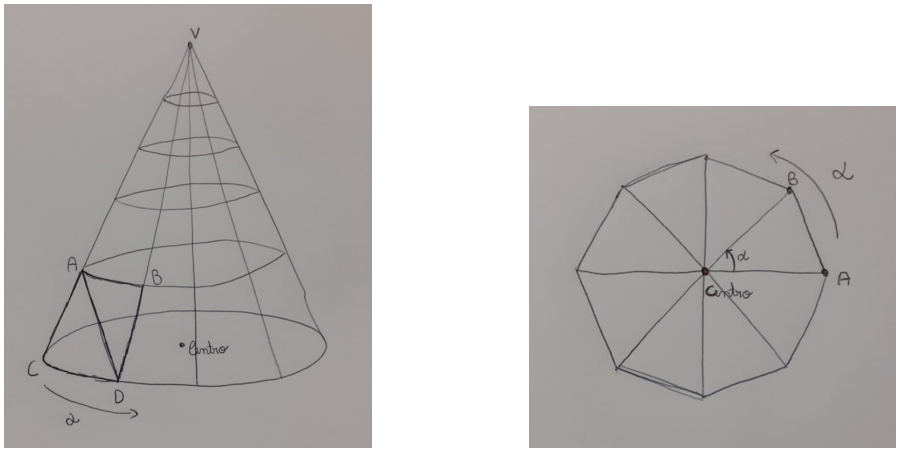

Figure 5: Representação de uma subsecção e da Base do Cone.

Para além disto, o raio de cada *stack* será igual a $r_{stack} = r - n_{stack} * Y$, sendo $Y = r/stacks$, sendo que este está representado na figura 5 por r2.

A geração do cone está dividida em duas secções: o lado do cone e a base.

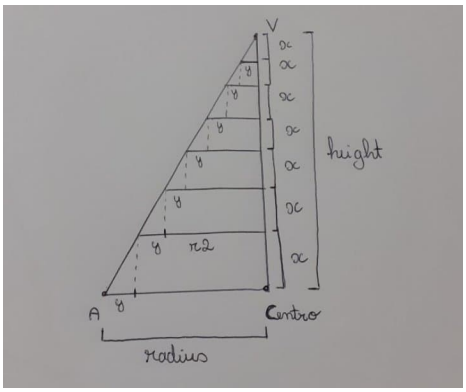

Figure 6: Representação de um corte do Cone.

3.1.5 Base do Cone

Como podemos ver na figura 4, na base do cone teremos de ter em conta o centro do cone, C, que corresponde à origem, e dois pontos, A e B, que formarão os triângulos que, juntos, formam a base. Assim, as coordenadas dos pontos serão baseadas num ângulo, *a*, que varia entre 0 e 2π. Assim, as coordenadas dos pontos serão as seguintes:

pA = $(r * sin(a), 0.0, r * cos(a))$
pB = $(r * sin(a_{Next}), 0.0, r * cos(a_{Next}))$
pC= $(0.0, 0.0, 0.0)$

Assim, juntando os triângulos que estes formam (sendo que será feito um número de triângulos igual ao número de *slices*), teremos a base do cone.

3.1.6 Lado do Cone

Os cálculos do lado do cone serão mais complexos que os da base. Teremos de ter em cada subsecção os quatro pontos que estão identificados na figura 4. Assim, as coordenadas destes pontos serão as seguintes:

pA = $((r_{atual} - r_{dif}) * sin(a), altura_{atual} + altura_{dif}, (r_{atual} - r_{dif}) * cos(a))$
pB = $((r_{atual} - r_{dif}) * sin(a_{prox}), altura_{atual} + altura_{dif}, (r_{atual} - r_{dif}) * cos(a_{prox}))$
pC = $(r_{atual} * sin(a), altura_{atual}, r_{atual} * cos(a))$
pD = $(r_{atual} * sin(a_{prox}), altura_{atual}, r_{atual} * cos(a_{prox}))$

De notar que, em cada ciclo, a variável $altura_{atual}$ aumenta por $\frac{height}{stacks}$, sendo inicializada a 0, e a variável $altura_{atual}$ decrementa por $\frac{radius}{stacks}$ e o seu valor é igual ao valor do raio que é passado para a sua geração.

Assim, no final do ciclo, vamos ter várias secções constituídas por dois triângulos, que, juntas, formam o lado do cone.

De seguida apresentamos um cone com raio 1, altura 1, 20 *stacks* e 20 *slices*.

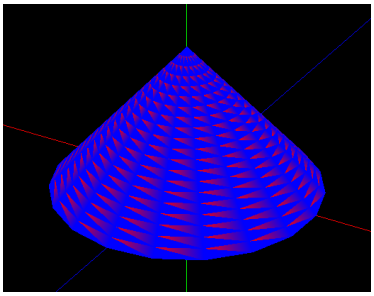

Figure 7: Cone gerado com o *Generator*

3.1.7 Escrita no Ficheiro

Para escrever no ficheiro, é utilizada a função *writePoint*. A esta é passado um *Point*, estrura explica na *Engine*, e um *ofstream*, correspondente ao ficheiro onde queremos escrever. Cada linha é escrita com o mesmo formato, guardando a informação de um ponto (mais uma vez, uma explicação detalhada do formato das linhas encontra-se na secção sobre a *Engine*).

O ficheiro em que escrevemos é aberto antes da geração dos pontos, e um apontador para o mesmo é utilizado ao longo do processo de geração, até este terminar, momento em que é fechado.

3.2 *Engine*

3.2.1 Leitura do ficheiro XML

Uma vez que a informação relativa aos modelos deve ser carregada para a *Engine* através da leitura de um ficheiro XML, nós utilizamos a biblioteca *TinyXML2* para poder ler o ficheiro mais facilmente. Note-se que inicialmente experimentamos a biblioteca que nos foi sugerida pelos docentes, a *TinyXML*, mas acabamos por optar pela *TinyXML2* porque tivemos alguns problemas a utilizar os métodos que a primeira oferecia.

O nome do ficheiro XML deverá ser passado como argumento para o programa, e este será processado pela função *loadFile*. Esta função irá guardar os nomes dos ficheiros com os modelos a serem lidos num *vector*<> de strings, que será posteriormente processado.

De notar que o programa não pode correr sem receber um ficheiro de input, de forma que, caso não seja recebido um ficheiro, o programa encerra com uma mensagem a informar que um ficheiro válido não foi disponibilizado.

```
<scene>
  <model file="/Users/tarly127/Desktop/teste_box.3d" />
  <model file="/Users/tarly127/Desktop/teste_plane.3d" />
  <model file="/Users/tarly127/Desktop/teste_sphere.3d" />
  <model file="/Users/tarly127/Desktop/teste_cone.3d" />
</scene>
```

Figure 8: Exemplo de um ficheiro XML.

3.2.2 Leitura dos Ficheiros dos Modelos

Depois da execução da função *loadFile*, será executada a função *loadModel*. Esta irá percorrer o vetor de strings anteriormente mencionado, abrindo cada um dos ficheiros referidos e lendo o seu conteúdo. Cada ficheiro conterá a informação relativa a uma figura, como tal, quando um ficheiro é lido, é instanciada a estrutura *Shape*, onde serão guardadas as informações referentes aos vértices que constituem a figura. Esses vértices serão guardados num *vector* de *Point*s, estrutura que contém três floats, referentes às coordenadas x, y e z de um ponto.

Dentro de cada ficheiro, em binário, note-se, está escrita em cada linha a informação de um único ponto, ou seja, o ficheiro terá a estrutura

```
x1 y1 z1\n\0x2 y2 z2\n\0 (...)
```

De notar que o '\0' surge no ficheiro por necessidade, devido à forma como uma linha é escrita no *Generator*, como explicado anteriormente. Esta forma de escrita foi assim decidida para ajudar no *debugging* do programa ao longo da sua conceção, uma vez que era apenas uma questão de alterar a forma de abertura no *Generator* para o ficheiro se tornar legível para um humano.

Quando se atingir um EOF, a *Shape* lida é guardada num *vector*, para ser posteriormente lida pelo programa.

3.3 Apresentação das figuras

O desenho de cada uma das figuras guardadas no *vector* de *Shape*s cabe à função *renderScene*. Para além de desenhar os eixos, esta função também percorre o *vector* de *Shape*s, desenhando os seus vértices um a um. A cada três vértices desenhados cria um triângulo, com alternância de cores a cada triângulo para facilitar a visualização das formas geométricas.

3.3.1 *Inputs* do Teclado

3.3.1.1 Câmara A câmara é movida utilizando as setas do teclado, cujo resultado do input é ditado pela função *specialKeys*. Nesta o movimento das teclas referidas aumenta ou diminui os valores de duas variáveis, *a* e *b*, por $\pi/30$ rad, o que movimenta a câmara com base na definição dos valores de x, y e z (de aqui em diante referidos como px, py e pz, respetivamente) no sistema de coordenadas esféricas, ou seja:

$$px = radius * cos(b) * sin(a)$$
$$py = radius * sin(b)$$
$$pz = radius * cos(b) * cos(a)$$

Note-se que aqui, por defeito, escolhemos radius = 5, uma vez que é o valor que estamos habituados a utilizar nas aulas práticas, e é um bom valor de referência para a distância da câmara ao centro.

As mudanças às três variáveis vão ser depois relevantes dentro da função *renderScene*, através da sua aplicação na função *glLookAt*, nos parâmetros *eyeX, eyeY* e *eyeZ*, respetivamente.

3.3.2 Rotação da Figura

O modelo gerado pode ser rodado utilizando as teclas W, A, S e D, cujos inputs correspondentes estão definidos na função *keys*, onde duas variáveis, *angleX* e *angleY* são alteradas por 5 com cada input. Isto vai-se refletir na função *glRotatef*, onde a posição da figura vai ser alterada consoante o valor passado nas variáveis que referimos.

4 Conclusão

Primeiramente, acreditamos que atingimos os objetivos principais do trabalho. Esta primeira fase do projeto permitiu-nos estar mais à vontade com as funcionalidades mais básicas que o GLUT nos oferece, bem como com outros fatores que condicionaram o trabalho, como a utilização de uma linguagem nova e de *parsers* de XML.

As maiores dificuldades que enfrentamos estiveram maioritariamente relacionadas com o facto de nunca termos trabalhado com C++, que, apesar de ser semelhante a C, tem as suas nuances específicas que o distinguem do C. Para além disto, ao início enfrentamos alguns problemas com a biblioteca para fazer *parsing* do XML, de forma que acabamos por abandonar a biblioteca que usamos inicialmente. No entanto, este não foi um grande entrave no desenvolvimento desta fase, uma vez que o XML constituía uma parte pequena do trabalho.

Quanto às funcionalidades extra do trabalho, conseguimos realizar algumas, nomeadamente o movimento da câmara. No entanto, teria sido interessante experimentar algo relacionado com uma câmara em *first-person* ou mesmo com controlo através de *inputs* do rato. No entanto, devido a limitações impostas pelo tempo, não conseguimos implementar tais funcionalidades.

Em suma, apesar de termos tido algumas dificuldades, e de não termos ido mais além com funcionalidades extra, atingimos os objetivos mais importantes do projeto, sendo que o nosso trabalho refletiu-se em duas aplicações em C++ que produzem os resultados esperados nesta primeira fase.