Computação Gráfica
3º ano de LCC/MIEI

1º Fase de Entrega

Relatório de Desenvolvimento

Graphical primitives

João Gomes

a70400

João Dias

A72095

Joel Morais

A70841

Luís Ventuzelos

a73002

5 de Março de 2017

Resumo

Este documento apresenta o desenvolvimento do projeto de *Computação Gráfica* numa colaboração entre o Mestrado Integrado de Engenharia Informática *(MIEI)* e a Licenciatura de Ciências de Computaçao *(LCC)* correspondente ao ano lectivo *2016/2017*, analisando as estratégias utilizadas e as principais funções implementadas em cada uma das tarefas propostas.

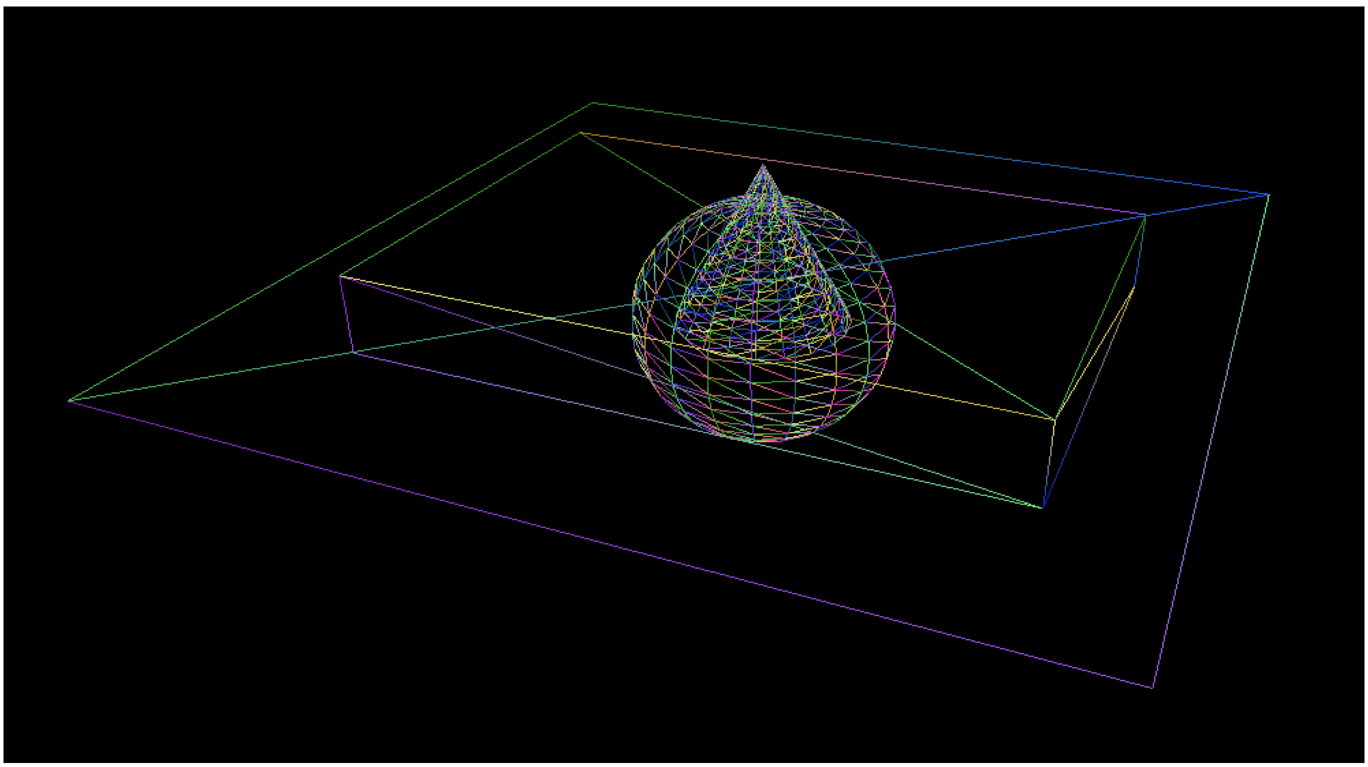

Figura 1: Modelo

Capítulo 1

Introdução

No âmbito da unidade curricular de *Computação Gráfica* foi-nos proposto o desenvolvimento de um projeto dividido em quatro fases. Este relatório refere-se à fase inicial, na qual o objetivo é a criação de primitivas gráficas. Para a resolução desta fase precisamos de desenvolver duas aplicações:

- A primeira é um gerador de pontos o qual nos cria os pontos necessários da figura pretendida a partir dos parâmetros que recebe, guardando esses valores num ficheiro .3d para o motor ler.
- A segunda, como referenciado, é a leitura pelo motor dum ficheiro em .xml , que contem os nomes das figuras ao qual tem de desenhar, para isso usa os pontos criados pelo gerador. Ao longo deste relatório iremos descrever detalhadamente todo o processo que desenvolvemos desde o gerador até motor, explicando as funções e estruturas utilizadas.

Capítulo 2

Generator

2.1 Plano

Um plano é formado pela concatenação de 2 triângulos, para isso são necessários 6 pontos, ou seja, 3 para cada triângulo. Foi instruído que este plano teria de estar centrado na origem.

Para se desenhar o plano necessitamos de um valor de X *(comprimento)* e Z *(largura)* do plano, dividimos depois esses valores por dois e declaramos os pontos (-X/2,0,Z/2), (X/2,0,Z/2), (-X/2,0,-Z/2) obtendo assim um dos triângulos e (X/2,0,Z/2), (X/2,0,-Z/2), (-X/2,0,- Z/2) que devolve o outro.

O valor de Y será afixado em 0 com o intuito de satisfazer a condição desse plano se centrar na origem.

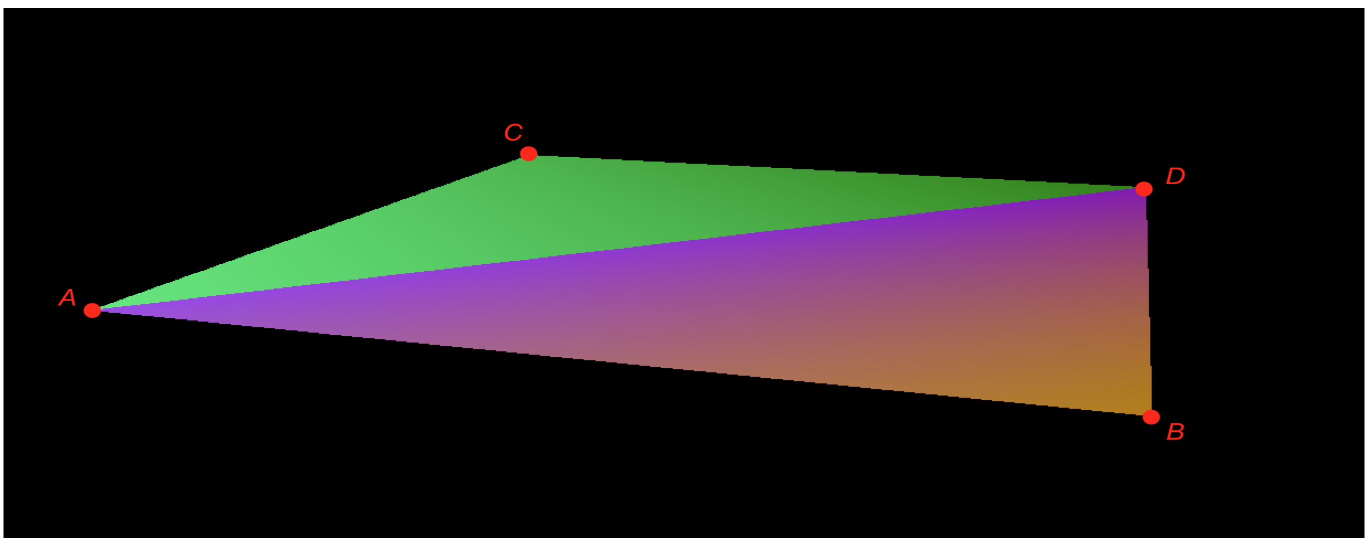

Figura 2.1: Plano

- **A**: (-X/2,0,Z/2);
- **B**: (X/2,0,Z/2);
- **C**: (-X/2,0,-Z/2);
- **D**: (X/2,0,-Z/2);

De modo a facilitar a representação do nosso plano atribuímos letras a cada ponto para que alguém leigo ao projeto tenha uma maior eficiência na sua compreensão, informação e cuidado também presente nos comentários do código.

Para gerar um plano basta então fazer:

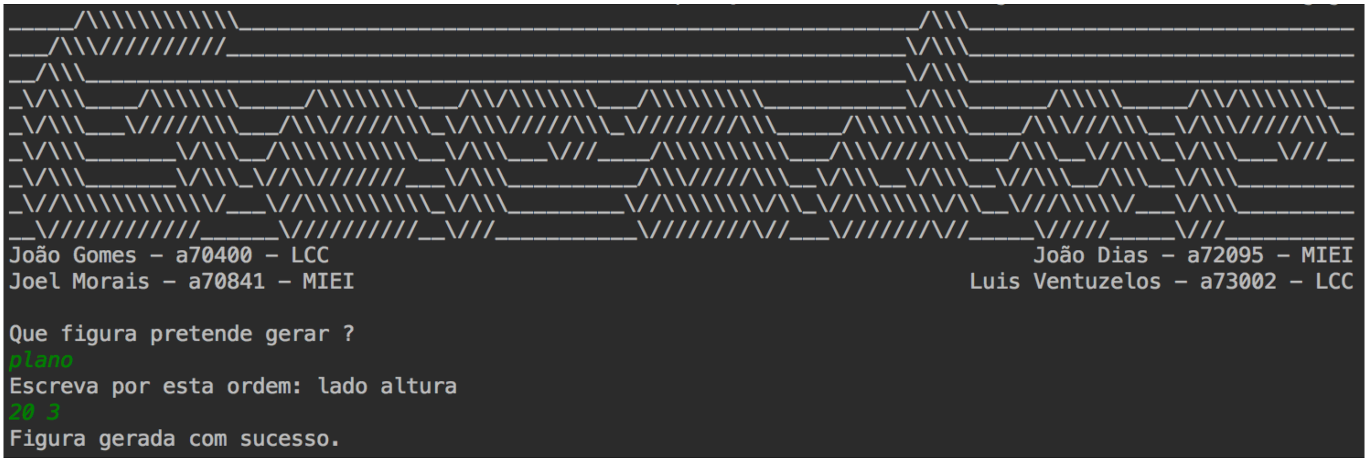

Figura 2.2: Linha de comando para gerar plano

2.2 Box

Uma caixa é desenhada usando 6 planos onde cada plano é desenhado com 2 triângulos ,previamente explicitado. Uma caixa necessita de comprimento (**X**), largura (**Z**) e altura (**Y**), para a centrar na origem vamos então dividir essas coordenadas por 2. Os pontos que usamos para desenhar a caixa foram:

- **A**:(-X/2,-Y/2, Z/2);
- **B**:(-X/2, Y/2, Z/2);
- **C**:(-X/2, Y/2,-Z/2);
- **D**:(-X/2,-Y/2,-Z/2);
- **E**:(X/2,-Y/2, Z/2);
- **F**:(X/2, Y/2, Z/2);
- **G**:(X/2,-Y/2,-Z/2);
- **H**:(X/2, Y/2,-Z/2);

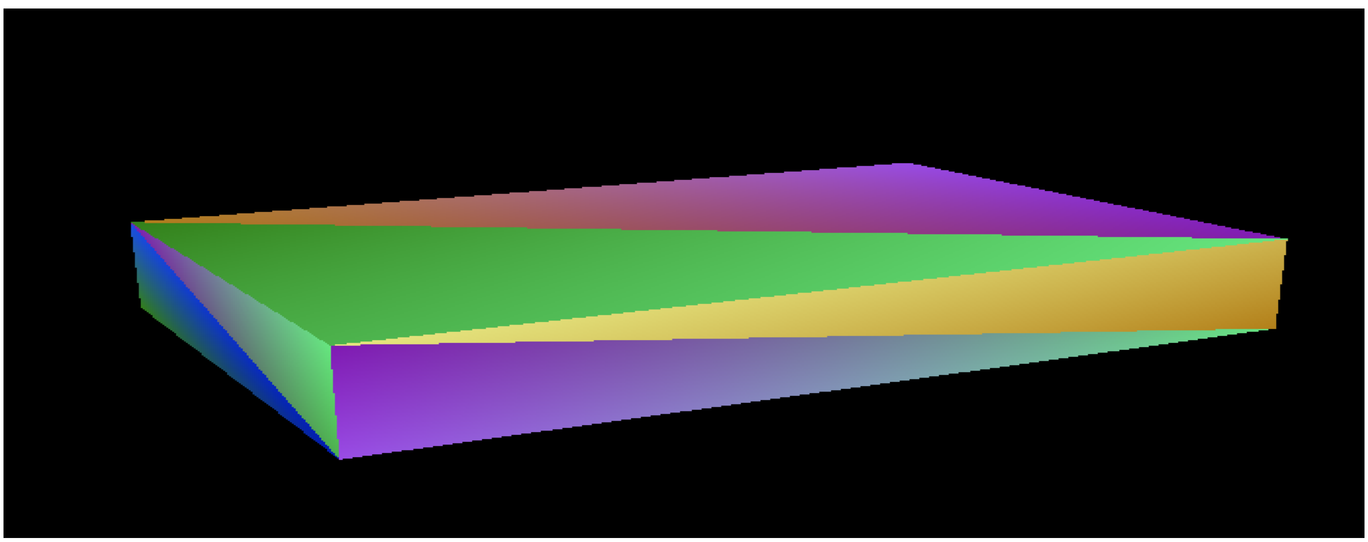

Figura 2.3: Caixa

Para gerar uma caixa basta fazer:

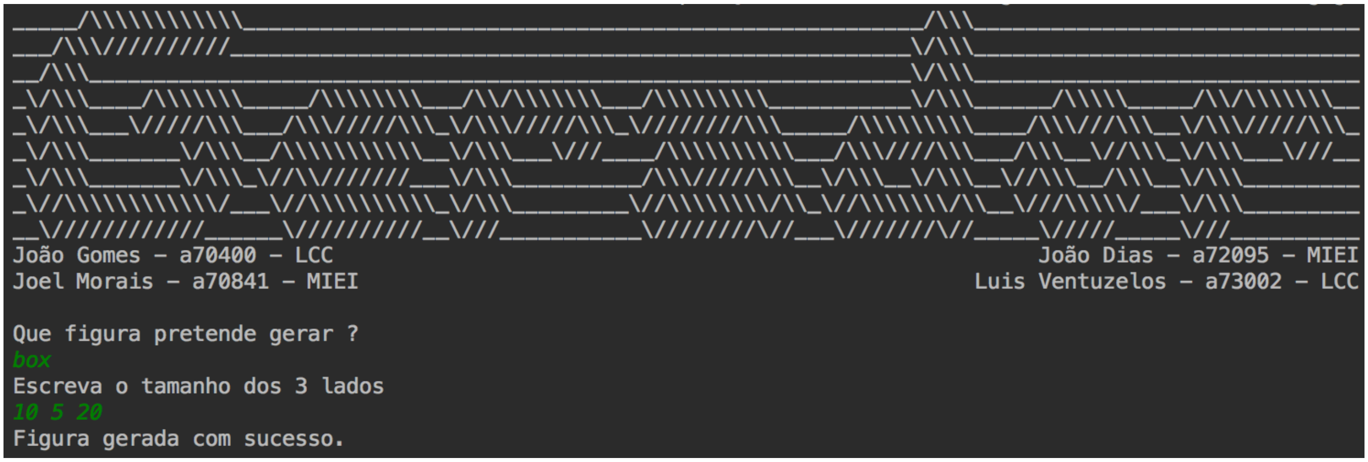

Figura 2.4: Linha de comando para gerar caixa

2.3 Cone

Para a representação dum cone será necessário que a sua base (uma *circunferência* centrada na origem) seja partida em n fatias. Para representarmos pontos na base do cone, usamos as equações cartesianas da *circunferência*, isto é, um ponto p=(**X**,**Y**,**Z**) que esteja na base do cone tem as seguintes coordenadas, com **alpha** a variar entre 0 e 2π:

- **X** = raio∗sin(**alpha**);
- **Y** = 0;
- **Z** = raio*cos(**alpha**);

Como previamente referido, uma parte da estratégia para a construção do cone foi a divisão do cone em fatias(verticais) e, além disso, em fatias horizontais que iremos denominar de camadas.

O objetivo das fatias é gerarem a base do cone e por cada uma gerada, geramos também todas as camadas correspondentes a essa fatia, ou seja, cada fatia é formada por 1 triângulo na base e por várias camadas na sua superfície lateral. Cada triângulo da base é formato por 3 pontos, 1 ponto no centro da base, e 2 na periferia circulo, como podemos observar:

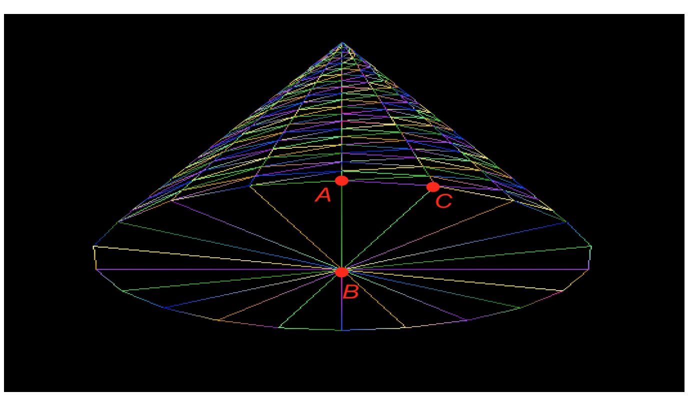

Figura 2.5: Pontos para a fatia do cone

Os pontos representados na figura serão da seguinte forma:

- **A** = (raio∗sin(i),0.0,raio∗cos(i))
- **B** = (0.0,0.0,0.0)
- **C** = (raio∗sin(i+1),0.0,raio∗cos(i+1))

Para desenharmos a superfície curva do cone temos um ciclo que faz slices+1 iterações. Suponhamos que o ponto **A** é o ponto correspondente à iteração **i**, para sabermos o ponto **C** usamos exatamente a mesma estratégia do ponto **A**, mas para **i+1**. Assim, terminada esta iteração, o ponto **C** será o ponto "inicial"da próxima iteração e, assim sucessivamente construimos a nossa base.
Para gerarmos as camadas de cada fatia seguimos os seguintes passos:

1. Calculamos a diferença de altura entre cada camada;
2. Calculamos a proporção do raio em relação à altura do cone;

Para explicarmos os passos seguidos, vamos ter em conta a seguinte figura, e as suas variáveis:

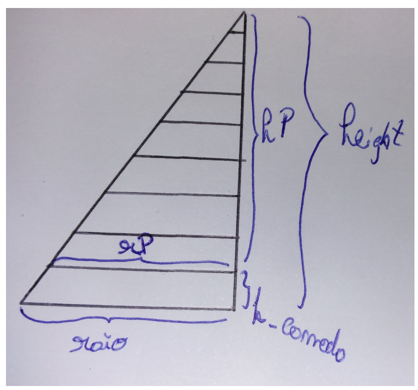

Figura 2.6: Vista de uma stack

Passo 1:
Para definir a altura de cada camada, podemos interpretar como a "espessura"dessa camada e, nesse caso, vai ser **h_camada** = height/stacks

Passo 2:
A proporção do raio em relação à altura do cone tem como objetivo saber o tamanho do raio de cada camada à medida que nos aproximamos do topo do cone, isto pois o raio de cada camada vai diminuindo, de forma constante, tal como a altura relativa a essa camada em relação ao topo do cone, por isso, a proporção entre cada raio e a respetiva altura vai ser sempre constante para todas as camadas. Para o cálculo dessa proporção usamos uma simples regra de 3 simples: **p** = (1*raio)/height

Por último, podemos ver neste exemplo como gerar um cone, sendo necessário passar 4 argumentos (raio,altura,slices,stacks):

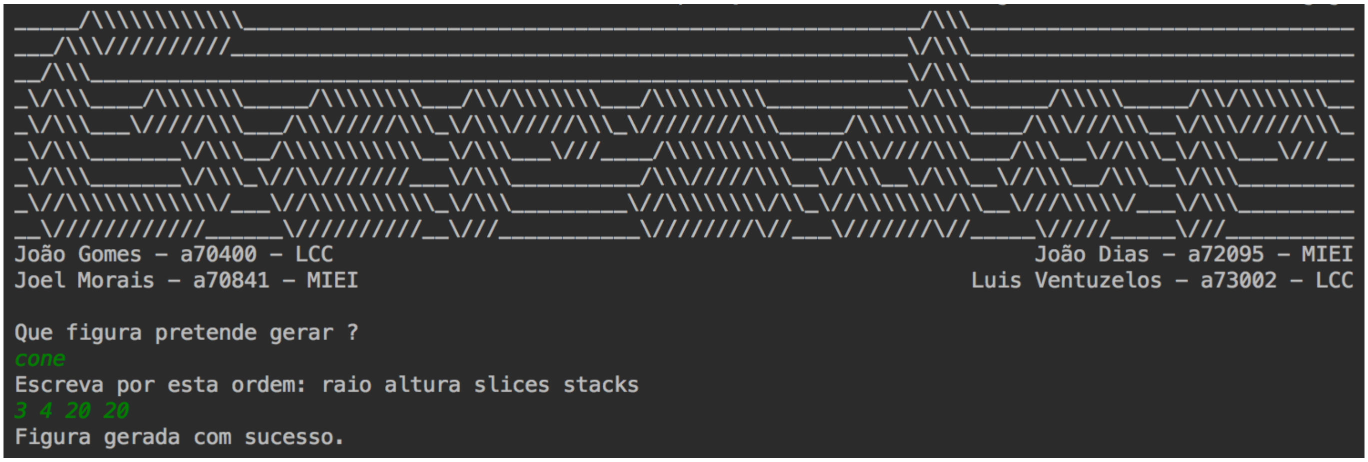

Figura 2.7: Linha de comandos cone

2.4 Esfera

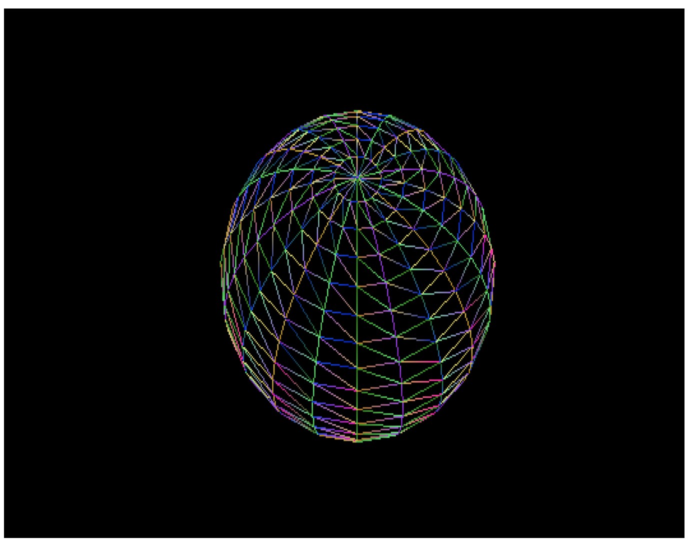

Figura 2.8: Esfera

Antes de começarmos a desenhar a nossa esfera, é necessário sabermos as coordenadas dos pontos que irão estar na superfície da mesma. A estratégia utilizada foi a seguinte:

Passo 1:
Definimos dois ângulos:

1. **beta** que varia entre -$\pi/2$ e $\pi/2$
2. **alpha** que varia entre 0 e 2*π

Passo 2:
Usando estes ângulos e as equações cartesianas da superfície da esfera, ficamos com as seguintes igualdades, que nos vão permitir desenhar pontos na superfície da mesma:

- **X** = raio$*cos($**beta**$)*sin($**alpha**$)$;
- **Y** = raio$*sin($**beta**$)$;
- **Z** = raio$*cos($**beta**$)*cos($**alpha**$)$;

Passo 3:
Para além destes, é necessário definirmos mais um ângulo (**angF**) que irá determinar a distância entre o ponto do inicio da nossa stack e o do fim dessa mesma stack, que será o ângulo que vamos ter em conta para calcular os pontos do inicio da próxima stack e assim sucessivamente.

Após definidas estas partes, visto que a nossa esfera será feita através de fatias e camadas, com a interceção das duas iremos obter pequenos quadrados, sendo estes definidos através de quatro pontos e dois triângulos, tal como representado na figura seguinte:

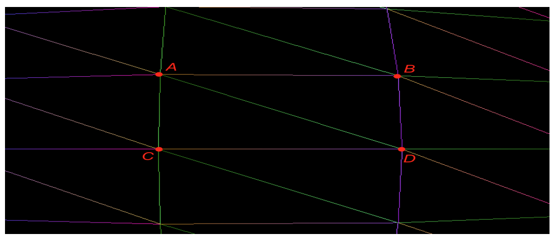

Figura 2.9: Pontos na superfície da esfera

É necessário calcularmos os valores de **X Y Z** de cada ponto acima mencionado. Como pode ser visto no nosso código, estamos a utilizar dois ciclos aninhados para este efeito, um dos ciclos para as fatias e outro para as camadas. Para isto temos #fatias iterações, desde i=0 até i=#fatias. Quando temos (i+1) significa que o ângulo **alpha** é o valor do ângulo **alpha** em i mais (2π/fatias). O j começa em $-\pi/2$ e quando temos (j+1) significa que o ângulo **beta** é o valor do ângulo **beta** em j mais (π/camadas). Depois de descobrirmos os ângulos de cada ponto, conseguimos então obter as suas coordenadas e através delas criar o retângulo que resulta dos dois triângulos definidos por esses pontos.

Por ultimo, podemos ver neste exemplo como criar uma esfera com raio 3 (que será centrada na origem), com 20 fatias e 20 camadas:

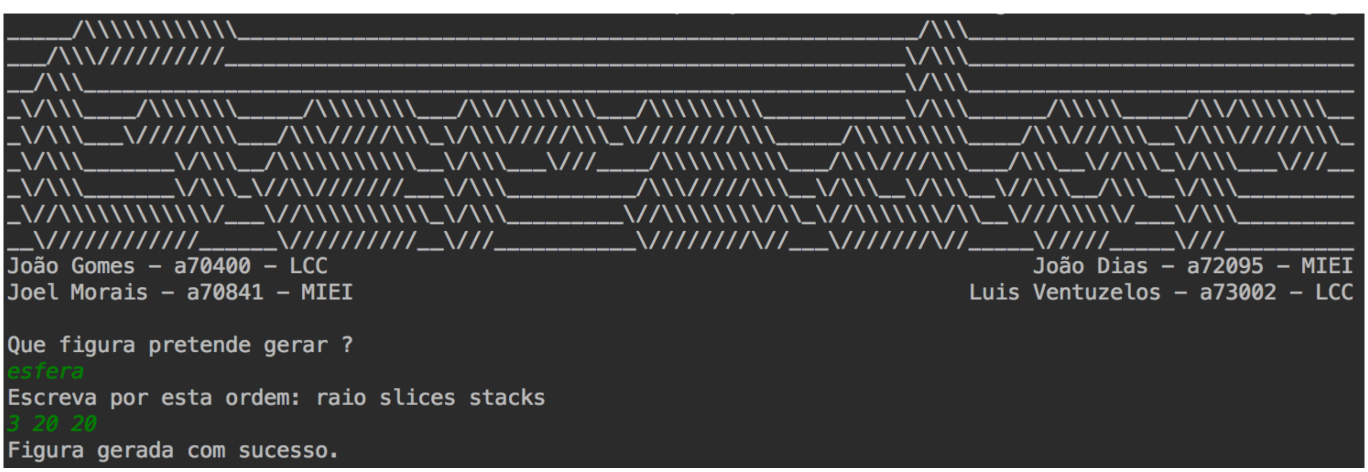

Figura 2.10: Linha de comando para gerar esfera

Capítulo 3

Motor

O Motor tem a função de desenhar as figuras. Para isso, lê um ficheiro XML escolhido pelo utilizador (que tem que ser do formato que apresentamos de seguida) e desenha todas as figuras que lá estão referenciadas.

3.1 Ficheiros XML

O ficheiro XML é escolhido pelo utilizador, dentro dos disponíveis, e desenha todas as figuras correspondentes ao ficheiro passado como string, no atributo do campo **file**, como podemos observar:

```
<scene>
    <Model file = "fig1.3d" />
    <Model file = "fig2.3d" />
    <Model file = "fig3.3d" />
    <Model file = "fig4.3d" />
</scene>
```

Para fazermos o parsing dos ficheiros XML usamos a biblioteca TinyXml.
De maneira a navegar até as linhas que contêm o campo Model usamos:

```
TiXmlElement *scene = doc.FirstChildElement("scene");
if(!scene){
   cerr << "Failed to load file: No root element." << endl;
doc.Clear();
return EXIT_FAILURE;
}

TiXmlElement *linha = scene->FirstChildElement("Model")->ToElement();
```

O **scene** é usado aqui para controle de erros de maneira a garantir que o nosso ficheiro XML está bem estruturado, como se pode ver pelo **if** acima documentado. O TiXmlElement vai nos construir um nodo que, neste caso, tem por nome linha. De maneira a iterar por cada filho do nosso nodo principal (o nodo linha) usamos:

```
for (; linha != NULL; linha = linha->NextSiblingElement())
```

Para extrairmos os atributos do file fazemos:

```
file = linha->Attribute("file");
```

Para finalizar armazenamos o nome do ficheiro **.3d** no final de um vetor declarado como variável global.

```
xmlF.push_back(file);
```

3.2 Ficheiros .3d

Nos ficheiros **.3d** são guardados os pontos gerados pelo generator que vai armazenar e mais tarde usados para desenhar as figuras.

A estrutura do ficheiro é simples, em cada linha vamos ter os pontos da nossa figura. A quantidade de pontos em cada linha varia conforme a figura. Por exemplo no caso da box, do plano e do cone o ficheiro **.3d** vai ter por linha 3 pontos definidos:

- coordenada1 espaço coordenada1 espaço coordenada1 espaço coordenada2 espaço coordenada2 espaço coordenada2 espaço coordenada3 espaço coordenada3 espaço coordenada3

No caso da esfera irá ser um ponto por linha:

- coordenada1 espaço coordenada1 espaço coordenada1
- coordenada2 espaço coordenada2 espaço coordenada2
- ...

3.3 Função drawFigures()

Na função **drawFigures()** vamos desenhar as nossas figuras, definidos por pontos no nosso ficheiro **.3d**.

Para tal vamos ler o nosso vetor **xmlF** (previamente declarado como variável global), aceder à sua primeira posição e armazenar o seu valor na variável nome.

```
for (nome = xmlF.begin(); nome != xmlF.end(); ++nome)
```

De seguida verificamos qual das figuras pretendemos desenhar e procedemos ao desenho das mesmas, como ilustrado no exemplo seguinte:

```
if (*nome == "box.3d"){
ifstream input_file("box.3d");
glBegin(GL_TRIANGLES);
while (input_file >> a >> b >> c){ // le 3 inteiros de cada vez
glColor3f((GLfloat) fmod(red, 1.0), (GLfloat) fmod(green, 1.0), (GLfloat) fmod(blue, 1.0));
glVertex3f(a, b, c);
red+=0.1;green+=0.2;blue+=0.2;
}
glEnd();
```

3.4 Câmara

Para podermos ver e interagir com os modelos que foram desenhados precisamos de usar algumas funções fornecidas pelo **GLUT**, como por exemplo **glutKeyboardFunc()** que é responsável pelo reconhecimento de teclas do teclado, isto é, utilizando as teclas [A,S,D,W,Q,E].
O que estas teclas nos permitem fazer é o seguinte:

- **A** - roda em relação ao eixo **y**;
- **S** - zoom-out;
- **D** - roda em relação ao eixo **y**;
- **W** - zoom-int;
- **Q** - roda em relação ao eixo **x**;
- **E** - roda em relação ao eixo **x**;

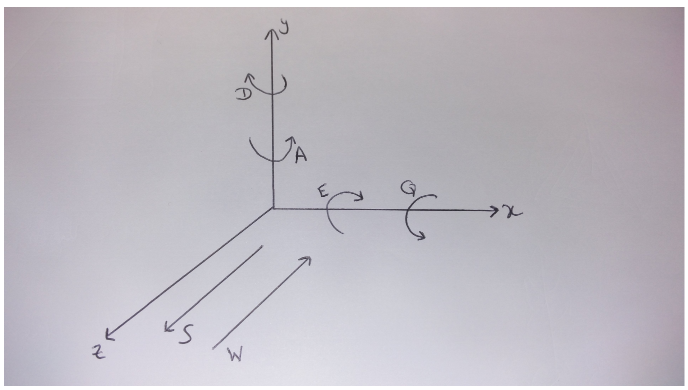

Figura 3.1: Reconhecimento Teclado

Capítulo 4

Conclusão

Primeiramente, atingimos o objetivo inicial de criar um gerador funcional que cumpre com os requisitos propostos pelo docente,ou seja, um gerador de pontos especifico para cada tipo de figura.

Em segundo lugar, o desenvolvimento de um motor que aplica esses pontos na criação das diversas figuras geométricas. Relativamente a essas mesmas figuras, no caso do plano achámos bastante trivial a sua construção, visto que seria apenas necessária a criação de dois triângulos; Passando para a caixa, visto que inicialmente estruturámos cada vértice com papel e lápis, a construção desta também se tornou simples; Considerámos o cone a figura mais difícil, pois iniciamos o seu desenvolvimento a partir duma estratégia desadequada, não conseguindo implementar as fatias. Problema este que foi superado com sucesso. Por último, relativamente à esfera, debatemo-nos com alguns problemas, todavia também esses obstáculos foram ultrapassados.

Porém, apesar de atingirmos todos os objetivos propostos, certos detalhes, alguns sugeridos pelo professor e outros que, na nossa opinião, melhorariam o projeto, não foram implementados. Exemplo disso é a criação de divisões na caixa e ainda, a criação do ficheiro *.xml* automaticamente na pasta relativa ao motor ou o seu acesso direto neste através de uma **systemcall** (pesquisas levaram-nos ao método *realpath()*) foram infrutíferas.

Em suma, consideramos o projeto um sucesso pois cumprimos todos os requisitos dele esperados.

Bibliografia

[1] http://www.grinninglizard.com/tinyxmldocs/annotated.html

[2] http://www.cplusplus.com

[3] Documentação fornecida pelo professor