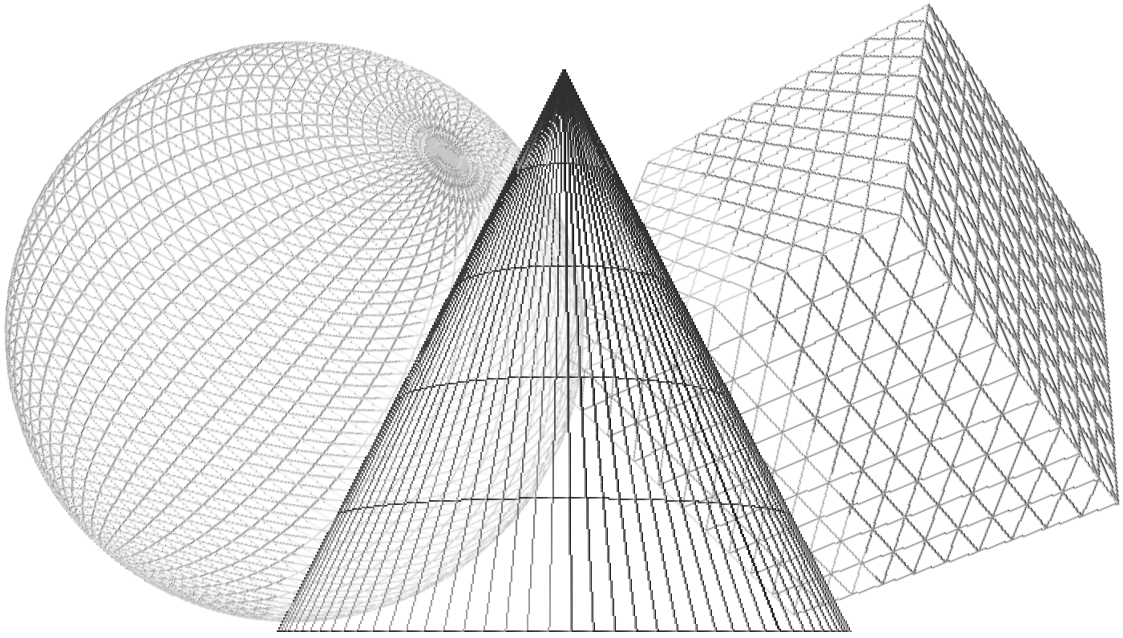
Trabalho de Computação Gráfica

FASE 1



Grupo:

- João Ferreira, A50193
- Luís Azevedo, A66704
- Luís Albuquerque, A79010
- Rafaela Pinho, A77293.

Docente: António José Borba Ramires Fernandes

Ano Letivo: 2017/2018

março de 18

Índice

Índice de figuras

Introdução

Foi proposto, no âmbito da unidade curricular computação gráfica, a elaboração de um projeto que ambiciona a criação de um software fabricante de objetos, ambientes e cenários 3D. É sugerida a utilização de ferramentas como OpenGL, Glut e Glew que com base na linguagem c++ facilitem a chegada ao objetivo final do projeto.
Este relatório, visa fundamentar os passos, escolhas, dificuldades e soluções que o grupo encontrou no caminho da concretização da primeira de quatro fases deste projeto semestral.

Começamos por partir o "problema" em fatias. As duas principais fatias óbvias desta fase foram a criação do "gerador" de ficheiros de configuração dos objetos 3D, e a outra fase do "motor" que consome os ficheiros de configuração gerados pelo "gerador" e desenha em ambiente gráfico esses mesmos objetos.

A abordagem ao "gerador" é repartida pelos vários objetos a serem produzidos para o ficheiro de configuração (um a cada execução do gerador). Quanto ao "motor", reparte trabalhos por 3 partes essenciais. Pela leitura e parse de um ficheiro XML que consome com a enunciação de todos os objetos a serem desenhados; pelo preenchimento dos buffers de desenho; e finalmente pelo desenho em si dos objetos.

Gerador

Recebe como argumentos os parâmetros necessários à criação das figuras geométricas e um nome, e cria um ficheiro de texto com o nome passado como parâmetro, onde guarda as coordenadas dos vários pontos que a figura necessita para ser gerada.

Neste momento suporta a criação das figuras *Sphere*, *Box*, *Cone* e *Plane*.

1. Possibilidades de utilização

- ./generator Plane x z [filename]
- ./generator Box x y z d [filename]
- ./generator Sphere radius slices stacks [filename]
- ./generator Cone radius height slices stacks [filename]

Em que **x**, **y**, **z** são dimensões e **d** é o número de divisões.
Note-se que apenas são permitidos valores positivos nos parâmetros numéricos.

1.1. Exemplo

./generator Cone 1 2 20 20 cone

A linha acima verifica se o número dos argumentos passados está correto, se os seus valores estão no intervalo permitido e, em caso afirmativo, cria um ficheiro de nome **cone.3d** na pasta onde está o executável.

```
0 0 1
0.309017 0 0.951057
0.0 2 0.0
0 0.1 0.95
0.293566 0.1 0.903504
0.0 2 0.0
0 0.2 0.9
0.278115 0.2 0.855951
0.0 2 0.0
0 0.3 0.85
0.262664 0.3 0.808398
0.0 2 0.0
0 0.4 0.8
0.247214 0.4 0.760845
0.0 2 0.0
0 0.5 0.75
0.231763 0.5 0.713292
0.0 2 0.0
0 0.6 0.7
0.216312 0.6 0.66574
0.0 2 0.0
0 0.7 0.65
0.200861 0.7 0.618187
```

Figura 1- Ficheiro de texto com as coordenadas do cone

2. Plane

Para criação do plano é invocada a função ***drawPlane*** que recebe como parâmetro o nome do ficheiro onde as coordenadas serão guardadas e, opcionalmente, as dimensões **x** e **z**. Caso não sejam passadas dimensões, o plano é criado com dimensão x=2 e z=2. [Ver criação do plano, pag.10]

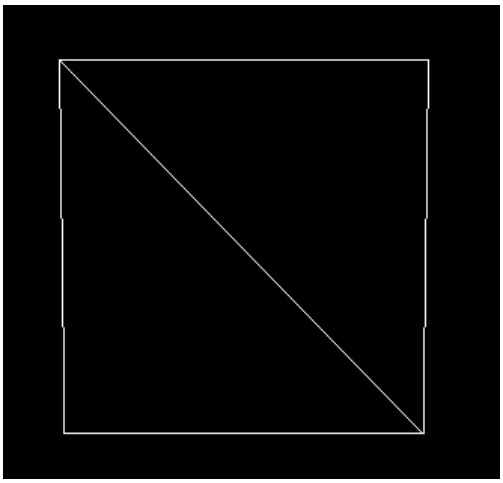


Figura 2- Plano gerado com x e z iguais a 2

3. Box

Para criação da caixa é invocada a função ***drawBox*** que recebe como parâmetros as dimensões ***X, Y, Z*** e o nome do ficheiro onde as coordenadas serão guardadas.
[Ver criação da caixa, pag.11]

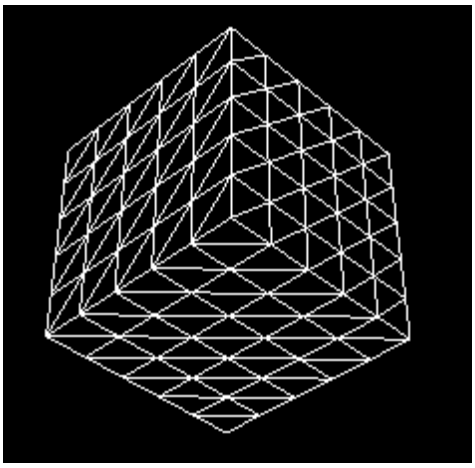


Figura 3- Box de 1 x 1 x 1 e 5 divisões

4. Sphere

Para criação da esfera é invocada a função ***drawSphere*** que recebe como parâmetros ***radius***, ***slices***, ***stacks*** e o nome do ficheiro onde as coordenadas serão guardadas. [Ver criação da esfera, pag12]

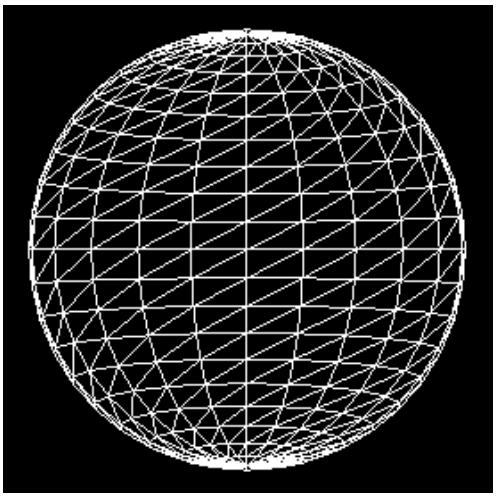


Figura 4- Esfera de raio 1, com 20 slices e 20 stacks

5. Cone

Caso a figura pretendida seja um cone é invocada a função *drawCone* que recebe como parâmetros *radius*, *height*, *slices*, *stacks* e o nome do ficheiro onde as coordenadas serão guardadas. [ver criação do cone, pag.13]

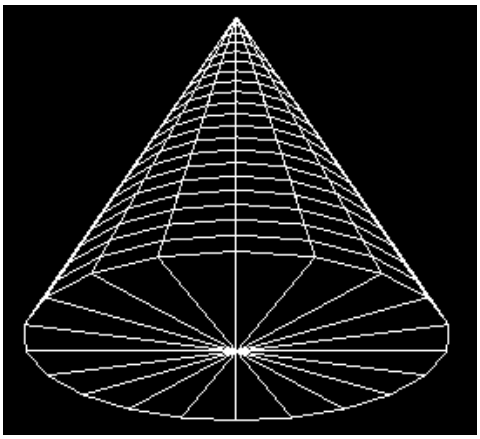


Figura 5- Cone de raio 1, altura 2, 20 slices e 20 stacks

Engine

Recebe um único parâmetro, o nome de um ficheiro com extensão *XML* previamente criado, onde estão os nomes dos ficheiros texto que contêm as coordenadas para criação das figuras geométricas.

Neste caso temos o ficheiro ***config.xml*** e para correr o *engine* fazemos:

- ./engine config.xml

À medida que o XML é lido, com recurso ao *parser* ***TinyXML***, os ficheiros que contêm as coordenadas dos pontos são abertos e estas são armazenadas em três vetores através da função ***readCoord***.

Quando todos os ficheiros tiverem sido lidos e as coordenadas armazenadas, começamos então gerar os pontos que compõem os triângulos necessários à criação das figuras geométricas pretendidas. Os triângulos que formam as figuras são criados através do ***GLUT*** na função ***renderScene***. É utilizado um simples ciclo que percorre todas as posições dos vetores e vai criando os pontos com a função ***glVertex3f***.

```
1  <scene>
2      <model file="box.3d"/>
3      <model file="plane.3d"/>
4      <model file="cone.3d"/>
5      <model file="sphere.3d"/>
6  </scene>
```

Figura 6- Ficheiro XML

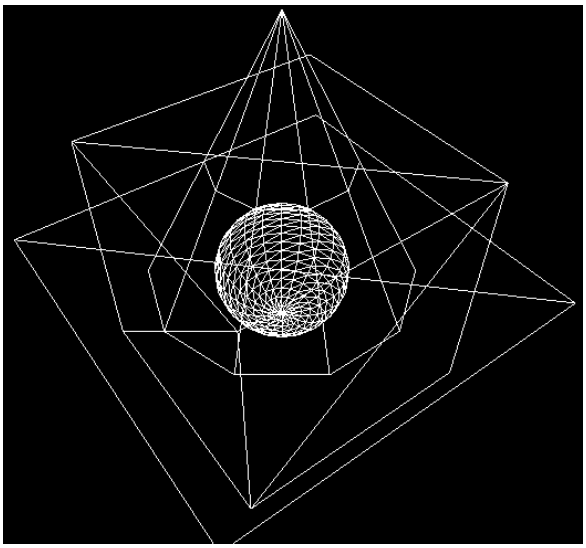


Figura 7- Todas as figuras geradas

Compilação

Para além dos ficheiros necessários à criação das figuras é também fornecida uma *Makefile* para compilação e execução do projeto.

Para executar o ***engine*** basta fazer "*make exe*"

Plane

Um plano é uma figura geométrica finita de duas dimensões, que pode ser determinado através de, por exemplo:

- Um ponto e um vetor;
- Duas retas;
- Três pontos não colineares;

Nesta parte do trabalho o objetivo era construir um plano no eixo XZ, centrado na origem, a partir de dois triângulos.

Construímos uma função, que pode receber argumentos ou não e que terá sempre coordenada do y a 0, chamada **drawPlane.** Quando a chamamos sem argumentos, drawPlane(), vai desenhar por omissão quadrado que tem os lados de tamanho 20. Se os valores não forem por omissão a função poderá receber no máximo 2 parâmetros, drawPlane(float X , float Z), ficando assim o tamanho de dois lados com o valor X e outros dois com o valor de Z, caso o Y seja diferente de 0 (zero) o plano fica centrado no ponto (0,Y,0).

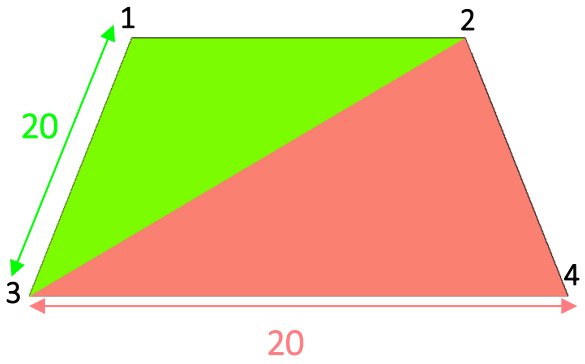


Figura 8- Plano por omissão

Para construir os dois triângulos começamos por dividir em 2 definir as coordenadas dos pontos, depois definimos a cor para os triângulos utilizando o *glColor3f* e por fim usando o *glBegin*(GL_TRIANGLES), 6 vezes *glVertex3f* e o *glEnd*() construímos os dois triângulos. Se quisermos que cada triângulo tenha a sua cor depois de desenhar três vértices definimos uma nova cor. Também definimos da mesma forma os triângulos por trás.

Para definir o triângulo a rosa na figura 8, usamos primeiro o ponto 2, depois o 3 e por fim o 4. Para o triângulo verde usamos o 3, o 2 e o 1. Para a parte de trás usamos para o triangulo rosa o ponto 3, depois o 2 e o 4, para o verde usamos o 2, o 3 e o 1.

- Ponto 1(-x, y, -z)
- Ponto 2(x, 0, -z)
- Ponto 3(-x, 0, z)
- Ponto 4(x, 0, z)

NOTA: o x e o z têm como valor metade do X e do Z, respetivamente

BOX

A "box" é constituída por 6 faces em que são idênticas e paralelas duas a duas.

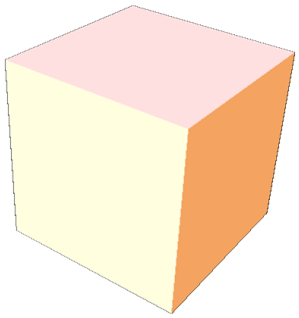


Figura 9- "Box"

Para construirmos a caixa necessitamos do tamanho da largura (eixo do X), da altura (eixo do Y) e do comprimento (eixo do Z). A nossa função, **drawBox**, recebe esses 3 parâmetros e recebe também um parâmetro d para a divisão da caixa. Como a nossa caixa vai ser centrada na origem vamos ter de dividir por 2 a largura, o comprimento e a altura. O número de divisões significa que cada face da caixa vai estar dividida em d^2.

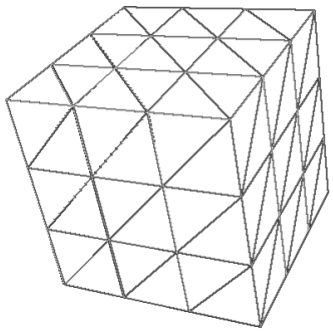


Figura 10- Caixa com lados todos e iguais e d=3

A nossa função **drawBox** depois de definir o x, o y e o z, define um dx, dy e dz. A partir daí cria 2 ciclos um para percorrer as linhas e outro as colunas e de cada vez que entra no ciclo cria dois triângulos para cada uma das faces.

Por exemplo, as coordenadas de dois triângulos da face da frente (Figura 11):

- Ponto1(x - (dx*(c+1)), y - (dy*l), z)
- Ponto2(x - (dx*c), y - (dy*l), z)
- Ponto3(x - (dx*(c + 1), y - (dy*(l + 1)), z)
- Ponto4(x - (dx*c), y - (dy*(l + 1)), z)

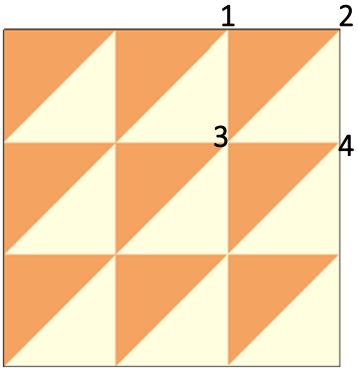


Figura 11- Face frontal da caixa

Sphere

A esfera é um sólido geométrico composto por uma superfície curva, na qual os pontos encontram-se à mesma distância do centro. Para construímos a esfera é necessário utilizar coordenadas esféricas (Figura 12).

Coordenadas esféricas

(α, β, r) e -90 < β < 90

Coordenadas cartesianas

px= r x cos(β) x sin(α)

py= r x sin(β) r'= r x cos(β)

pz= r x cos(β) x cos(α)

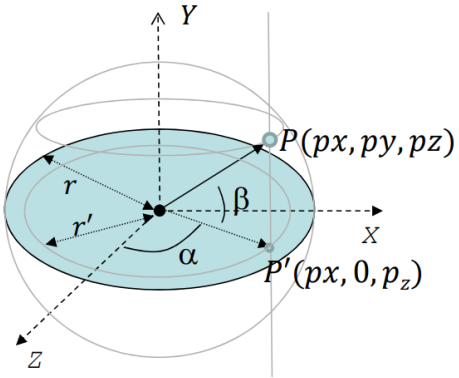


Figura 12- Coordenadas esféricas

Para construímos a esfera criamos uma função, **drawSphere**, que recebe como parâmetros o raio, o número de "slices" e de "stacks". A esfera vai estar centrada na origem, logo cada ponto terá de coordenadas:

- x = raio x cos((stacks / 2 - j) x β) x sin(i*α)
- y = raio x sin((stacks / 2 - j) x β)
- z = raio x cos((stacks / 2 - j) x β)*cos(i*α)

NOTA:

β = π / stacks α = (2 x π) / slices

i é o slice onde se encontra o ponto

j é a stack onde se encontra o ponto

Para desenhar a esfera vamos fazer dois ciclos, um que diz em que "slice" estamos e outro que diz a "stack". Dentro dos ciclos vamos construir dois triângulos, começando pelo ponto 3 depois o 4 e de seguida o 2. Para o outro utilizamos o 3, o 2 e o 1. (Figura13)

- Ponto1(raio x cos((stacks / 2 – (j + 1)) x β) x sin(i x α), raio x sin((stacks / 2 – (j + 1)) x β), raio x cos((stacks / 2 – (j+1)) x β) x cos(i x α))
- Ponto2(raio x cos((stacks / 2 – (j + 1)) x β) x sin(i x α), raio x sin((stacks / 2 – (j + 1)) x β), raio x cos((stacks / 2 – (j+1)) x β)*cos(i x α))
- Ponto3(raio x cos((stacks / 2 – (j + 1)) x β) x sin(i x α), raio x sin((stacks / 2 – (j + 1)) x β), raio x cos((stacks / 2 – (j+1)) x β) x cos(i x α))
- Ponto4 (raio x cos((stacks / 2 – (j + 1)) x β) x sin(i x α), raio x sin((stacks / 2 – (j + 1)) x β), raio x cos((stacks / 2 – (j+1)) x β) x cos(i x α)))

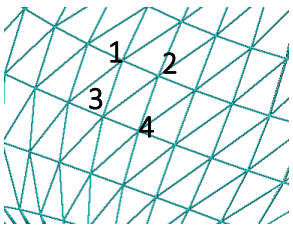


Figura 13- Recorte da esfera desenhada com linhas

Cone

O cone é um sólido geométrico, constituído essencialmente por duas partes, a base e o corpo do cone.

Para a construção do cone necessitamos das coordenadas polares (Figura 14).

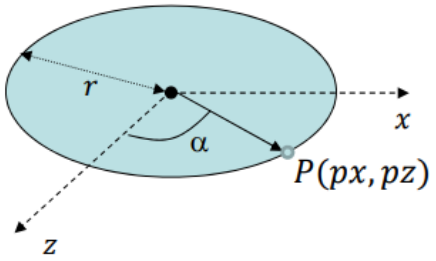


Figura 14- Coordenadas Polares

Coordenadas esféricas
(α, r)

Coordenadas cartesianas
px= r x sin (α)
pz= r x cos(α)

A base é feita através de um conjunto considerável de triângulos, que juntos formam um círculo, quantos mais triângulos, mais arredondado é a base. Para que os triângulos fiquem alinhados desta forma definimos um ângulo, que o seu valor é calculado dividindo 2π / slices. O corpo é feito de triângulos, em que cada triângulo, tem 2 pontos, (pontos que se encontram no raio da base), que já foram usados para fazer a base, e um outro que se encontra no ponto (0,altura,0).

A nossa função, **drawCone,** recebe como argumentos o raio, a altura do cone, o número de "slices" e o número de "stacks". A função utiliza um ciclo que percorre o número de "slices" e dentro deste existe outro que percorre as "stacks" e que vai desenhando cada triângulo da base. Dentro do ciclo das "stacks" desenha-se um triângulo, onde o raio é menor por cada iteração do ciclo. O cone é desenhado centrado na origem.

α = 2π / slices que quanto maior, mais arredondado fica, ou seja, mais fica parecido a um cone, figura geométrica.

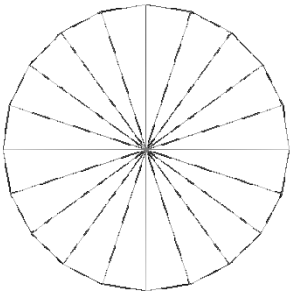


Figura 15- Base do cone

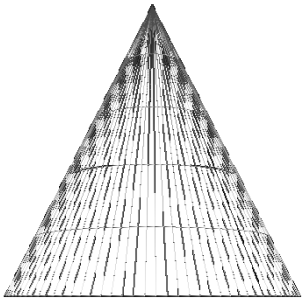


Figura 16- Cone visto de frente

Exemplo de um triângulo da base (Figura 17):

(ordem de desenhar os pontos, primeiro o 2 depois o 3 e depois o 1)

- Ponto1 (raio*sin(x*α), 0, raio*sin(x*α))
- Ponto2(raio*sin((x+1)*α), 0, raio*sin((x+1)*α))
- Ponto3(0,0,0)

Exemplo de um triangulo do corpo(Figura18):

(ordem de desenhar os pontos, primeiro o 2 depois o 1 e depois o 3)

- Ponto 1((raio*j)/stacks *sin((x+1)*α), 0, (raio*j)/stacks *sin((x+1)*α))
- Ponto2((raio*j)/stacks*sin(x*α), 0, (raio*j)/stacks *sin(x*α))
- Ponto3(0,altura,0)

Nota:

x – diz em que slice está o ponto

j – diz em que stack estamos

α = (2 x π) /slices

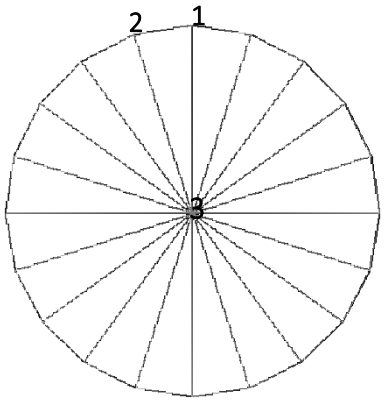


Figura 18- Base

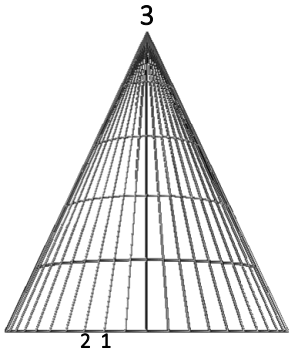


Figura 17- Corpo do cone

Conclusão

Com a realização desta primeira fase do trabalho prático aprofundamos conhecimento do que tinha sido dado em aulas práticas, openGL e GLUT, e para além disso, adquirimos novos conhecimentos quanto à utilização e manipulação de ficheiros XML utilizando uma ferramenta com que até agora não tinhamos trabalhado, o parser tinyXML.

Tivemos algumas dificuldades com a criação da esfera e do cone por também terem como parâmetro as "stacks", mas ainda fomos capazes de as ultrapassar a tempo de entrega.

Inicialmente previámos adicionar, para além das figuras pedidas, o Cilindro mas já não o conseguimos fazer em tempo útil. Caso numa próxima fase o consigamos gerar, adicioná-lo-emos.

Como trabalho futuro gostaríamos de adicionar os índices.