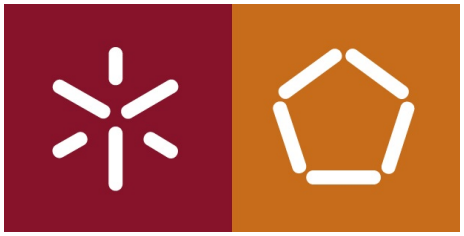
Computação Gráfica

Fase I

14 Março de 2020

---

a85517 Duarte Oliveira
a82098 Melânia Pereira
a81195 Tiago Barata
a67645 Sérgio Gomes



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1 Introdução

Para a primeira fase do trabalho prático da unidade curricular de Computação Gráfica foi necessário o desenvolvimento de duas aplicações, com o objetivo conjunto de desenhar o conjunto de primitivas do enunciado. Uma delas para gerar ficheiros que contenham informação sobre os vértices necessários para o desenho desses modelos e outra com a potencialidade de ler os ficheiros resultantes da anterior e permitir a visualização dos modelos. Estas aplicações serão denominadas *"Generator"* e *"Engine"*, respetivamente.

2 Aplicações

2.1 Generator

O objetivo deste programa é gerar ficheiros que contenham informação necessária para criar as primitivas gráficas requisitadas no enunciado desta fase do projeto. Assim, o *generator* vai receber o nome da primitiva, os parâmetros associados a ela e ainda a especificação do ficheiro no qual a informação desenvolvida vai ser guardada, ficheiro esse que terá o nome no seguinte formato: *nome_da_primitiva*.3d
Para desenvolver esta aplicação começou por se pensar em como se iria distinguir as primitivas, e rápido se chegou à conclusão de que seria pela comparação do primeiro argumento passado à aplicação, que identifica a primitiva a desenvolver.
Depois disso, seguiu-se o desenvolvimento de uma função para a geração dos vértices, estas funções recebem os parâmetros e calculam, então, os vértices referentes a cada um dos triângulos necessários para desenhar a primitiva em questão, escrevendo-os num ficheiro: um vértice por linha, sendo cada conjunto sucessivo de três linhas/vértices referente a um triângulo, e as linhas são escritas por ordem de desenho, ou seja, o primeiro vértice a ser desenhado é o constante da primeira linha, o segundo será o da segunda e assim sucessivamente, seguindo sempre a regra da mão direita.
O procedimento utilizado para o cálculo dos vértices será explicado mais a frente.

2.2 Engine

A aplicação *engine* irá ler e processar um ficheiro **XML** de configuração que contém o nome dos ficheiros .3d que deverão ser também lidos de seguida pela aplicação. Esse ficheiro é passado como argumento ao chamar a função, e o programa analisa a validade e existência desse ficheiro.
Para começar, como aqui será necessário desenhar as primitivas, ou seja, usar o *glut*, começou por se adicionar todas as funções necessárias, como a *renderScene()*, *changeSize()* e *main()*.
Depois, começou por se desenvolver uma função capaz de ler o ficheiro XML e, para isso usou-se o *tinyxml2*, que disponibiliza um conjunto de funções que facilitam essa leitura. Esta função recebe o nome do ficheiro a ler, e carrega-o para uma variável para poder ser acedido, depois, percorre os elementos filhos e, para cada elemento filho "*model*", guarda o valor do argumento "*file*"(que será o nome do ficheiro .3d a ler) noutra variável, sendo esta passada a outra função que irá ler e processar o novo ficheiro.
Essa função acede ao ficheiro e, por cada linha, guarda numa lista, um triplo de coordenadas, no formato *(x,y,z)*. Chegando a *EOF* procede-se ao desenho das primitivas. Este é realizado na função *renderscene()*, onde é chamada a função auxiliar *draw()* que itera sobre a lista de pontos, e desenha os triângulos correspondentes.

3 Raciocínio associado a cada primitiva gráfica

3.1 Plano

O plano requerido define-se como um quadrado no plano xz e centrado na origem, com um comprimento de lado fornecido pelo utilizador, ou seja, a coordenada y dos seus pontos será sempre igual a 0 e as outras coordenadas (x e z) serão sempre o comprimento recebido dividido por dois, ou o negativo disso, dependendo do quadrante em que estiver o ponto.
Para desenhar um quadrado são necessários quatro pontos, e a partir desses são desenhados dois triângulos (de forma a formarem um quadrado).

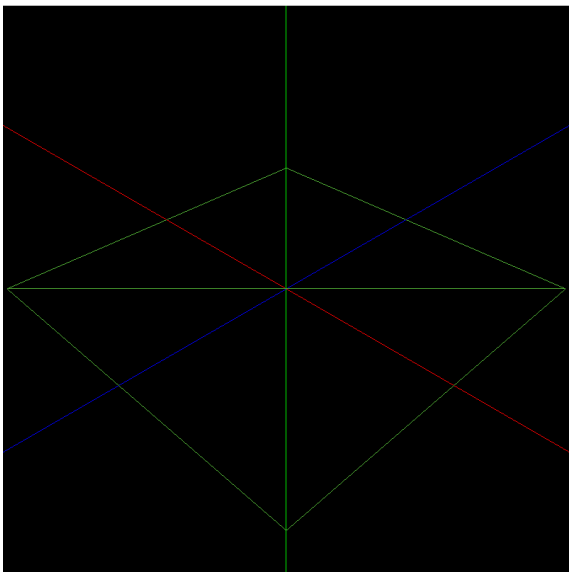
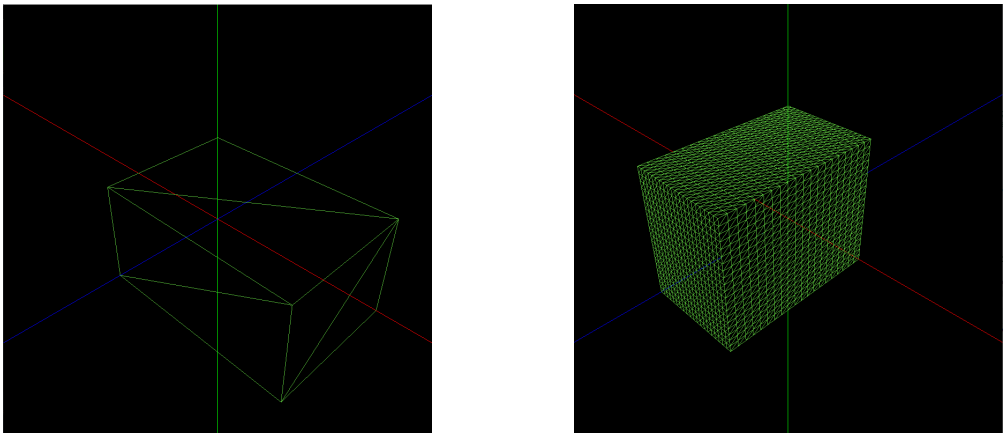


Figura 1: Plano

3.2 Caixa

Para o cálculo dos vértices da caixa, é necessário receber como argumento o seu comprimento, a sua largura, sua altura e, opcionalmente, o número de divisões por aresta. Decidiu-se, por conveniência e para cálculos mais rápidos, que os vértices seriam gerados de forma a que a caixa, depois de desenhada, ficasse no quadrante em que todas as coordenadas são positivas, por isso, o ponto inferior esquerdo da face de trás será o ponto origem (0,0,0).
Sendo que existe a opção de divisões por aresta, o tamanho dos lados terá que ser dividido pelo número recebido. Assim, usou-se um ciclo para percorrer cada quadrado dentro de cada face, em cada face haverá sempre uma coordenada que se mantém constante, por exemplo, na face da frente, a coordenada z será sempre igual, e as outras coordenadas serão calculadas de acordo com a iteração/divisão que se pretende desenhar.
Em cada quadrado tem-se quatro pontos, para cada um deles, as coordenadas que não sejam a que ficará constante, são sempre calculadas usando o número da iteração atual do ciclo, multiplicando essa pelo tamanho do lado já dividido, conseguindo assim o valor das coordenadas da divisão para a qual se pretende calcular os vértices, e, para apenas dois dos quatro pontos, adicionando ainda o valor do tamanho dividido.



(a) Caixa com 1 divisão

(b) Caixa com N divisões

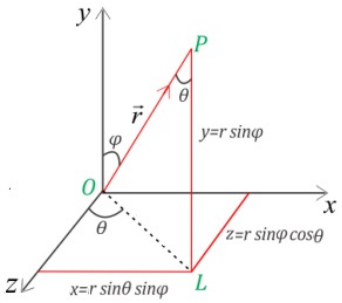
Figura 2: Diferentes caixas geradas

3.3 Esfera

A esfera, dada a sua natureza geométrica, revelou-se mais complexa do que as restantes figuras referidas até ao momento. Em primeiro lugar, temos que ter em consideração que terão que ser recebidos números de *slices* e *stacks*, para além do raio que se pretende que esta primitiva tenha. Estes dados vão ser o ponto de partida para se poder formar a figura, utilizando triângulos como unidade elementar para a construção da mesma, tal como foi feito até agora. É importante denotar que os valores destes argumentos detêm alguma importância visto que para haver coerência, os valores passados como argumento também o terão ter algum fundamento. Assim, é necessário que haja um valor relativamente alto de *slices* e *stacks*. A razão disto depende do fator crucial para se poder desenvolver esta esfera: são necessários 4 pontos iniciais, para poder-se derivar os restantes pontos que irão constituir esta primitiva. Esta derivação acontece pois é necessário se obterem coordenadas esféricas, em contraste com o que normalmente se fazia para outras primitivas geométricas. Para se obter essas coordenadas *(x,y,z)* devidamente convertidas utilizaram-se as seguintes fórmulas:

$$x = r \cdot sin(\theta) \cdot sin(\varphi)$$
$$y = r \cdot sin(\varphi)$$
$$z = r \cdot cos(\theta) \cdot sin(\varphi)$$

Onde , $\varphi = [0, \pi]$ e $\theta = [0, 2\pi[$



A partir daqui, foi relativamente fácil a construção da esfera. Cada ponto desloca-se relativamente ao ponto inicial (diga-se Ponto A), através da variação de θ e de φ com o respetivo deslocamento, que é calculado através dos limites superiores, a dividir pelas *slices* ou *stacks*. Este processo é realizado com dois ciclos encadeados, onde o ciclo exterior itera sobre as *slices* e o ciclo interior sobre *stacks*.

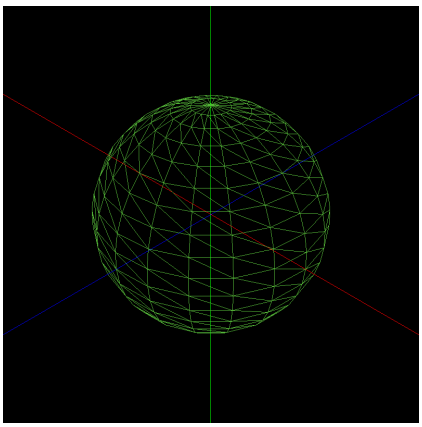


Figura 3: Esfera

3.4 Cone

Para o cone, começou por se pensar na sua base, que dada a sua natureza seria necessário usar coordenadas polares para o cálculo dos seus vértices, verificando-se o mesmo para os da face. Para tal definiu-se um ângulo **alfa** que inicia em 0 e vai até até 2π. Assim, de acordo com o número de *slices* que foram passadas como argumento, o ângulo vai incrementar em $\frac{2\pi}{slices}$ por cada iteração do ciclo.

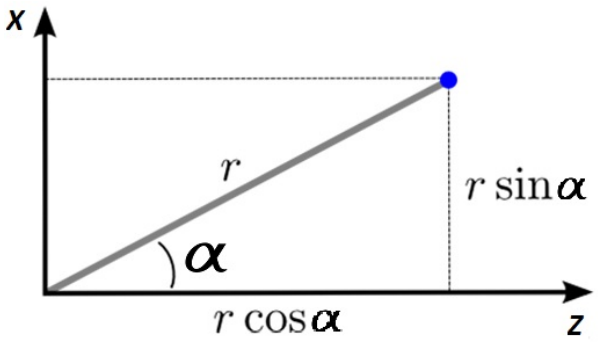


Figura 4: Coordenadas Polares

Dentro desse ciclo, terá também que se tratar dos vértices da face de cada *slice*. Existe aqui mais um parâmetro que é tomado em conta - as *stacks* - e por isso, ter-se-á outro ciclo. Tendo isto em conta, sempre que desenhamos um triângulo na face, há sempre um ou dois vértices que ficam com um raio menor e vice-versa.
Assim (e considerando i como a **"iteração atual"**) ao fazer os cálculos para os vértices percebeu-se que se teria que diminuir o raio em:

$$\frac{raio}{stacks} \times (i+1)$$

Obtendo assim o raio do ou dos **vértices de menor raio**, e diminuir em:

$$\frac{raio}{stacks} \times i$$

para obter o raio do ou dos vértices de maior raio.
O mesmo acontece para a altura dos vértices, para os mesmos em que o raio é menor, terá que se aumentar a altura em: $\frac{altura}{stacks}$.

Tal como anteriormente foi feito, a altura também terá que ser incrementada em cada iteração deste ciclo pois sempre que iteramos, significa que vamos escrever um vértice da *stack* seguinte/acima.

Finalmente, depois de encontrados todos os vértices de uma *slice*, o ângulo **alfa** tem que ser incrementado para que na iteração seguinte, se possa tratar da próxima *slice* do cone.

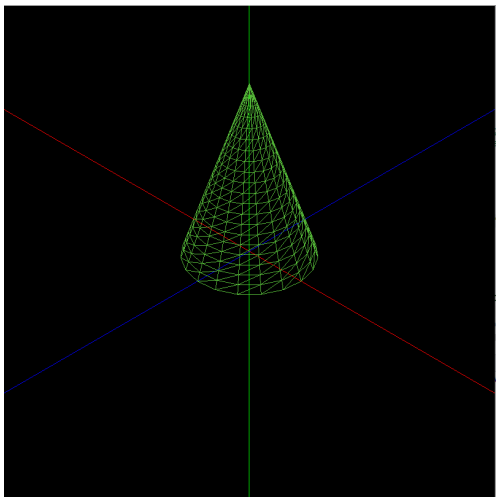


Figura 5: Cone

4 Notas e apreciações finais

Neste projeto para além do que nos foi proposto, decidimos também implementar comandos de transformações geométricas às figuras, permitindo manipular a posição, o ângulo e o tamanho, relativamente ao referencial. Para além disso foi um excelente método de consolidação da matéria lecionada nas aulas práticas desta Unidade Curricular.