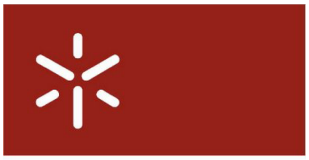
Universidade do Minho

Departamento de Informática

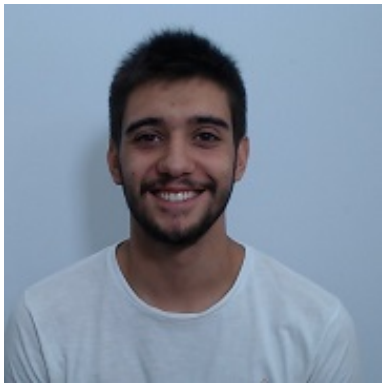


Universidade do Minho

Computação Gráfica

Fase 1 - Primitivas Gráficas

Grupo 4



André Gonçalves Vieira A90166

março de 2022

Conteúdo

1 Introdução

1.1 Contextualização

No âmbito da Unidade curricular de Computação Gráfica, foi proposto a o desenvolvimento de modelos 3D, através da utilização de *OpenGL* com base na biblioteca *GLUT* e desenvolvido na linguagem C++.

Esta é a primeira de quatro fases deste trabalho prático, que tem como objetivo a criação de algumas primitivas gráficas.

1.2 Resumo

As primitivas gráficas que foram elaboradas nesta fase são um **Plano**, uma **Caixa**, um **Cone** e uma **Esfera**, de acordo com os requisitos apresentados no enunciado do trabalho prático. Desta forma, serão apresentadas, ao longo deste relatório, as estratégias elaboradas de forma a realizar este projeto.

2 Estrutura do projeto

2.1 Aplicações

Nesta fase do projeto foram criadas duas aplicações que são essenciais para o seu funcionamento:

- **Generator** - Gera informação dos modelos e armazena os seus vértices num ficheiro especificado.
- **Engine** - Lê a configuração de um ficheiro XML e exibe os modelos indicados.

2.1.1 Generator

O **generator.cpp**, tal como dito acima, gera as figuras necessárias armazenando os vertices num ficheiro especificado. Para tal, necessita de receber o tipo de primitivas que se pretende gerar, todas as informações necessárias para as gerar e ainda o nome do ficheiro onde se pretende guardar a informação.

2.1.2 Engine

No **engine.cpp** são feitas a interpretação e leitura do ficheiro XML, onde estão contidas as informações referentes à camera, os modelos das primitivas gráficas e o armazenamento em memória dos vértices dos modelos.

2.1.3 Classes

Foram criadas duas classes **Ponto** e **Camera** de forma a simplificar a utilização da informação.

- **Ponto.cpp** - Classe utilizada para armazenar a informação necessária à criação de um ponto pertencente a um triângulo.

- **Camera.cpp** - Classe utilizada para armazenar a informação necessária para as definições de camera fornecidas nos ficheiros de teste.

3 Primitivas Geométricas

3.1 Plano

Para obtermos um plano, é necessária a criação de dois triângulos que partilham dois vértices entre eles. Uma vez que pretendemos criar uma espécie de grelha quadrada com *side* unidades de comprimento de lado e dividiao em *grid* divisões ao longo de cada eixo, cada destas divisões irá ser constituída por dois triângulos.

Como a figura está contida no plano xz, qualquer que seja o vértice de qualquer um dos planos constituintes desta "grelha", a coordenada y terá o valor 0. Em primeiro lugar, calculamos o comprimento de cada divisão desta grelha (*tlado*) que será *side / grid*. Posteriormente, calculamos as coordenadas dos pontos pelo qual começaremos a desenhar a primeira divisão da grelha que, uma vez que o plano é centrado na origem, serão:

- ($\frac{side}{2}$, 0 ,-$\frac{side}{2}$)
- ($\frac{side}{2}$ - *tlado*, 0 ,-$\frac{side}{2}$)
- ($\frac{side}{2}$ - *tlado*, 0 ,-$\frac{side}{2}$ + *tlado*)
- ($\frac{side}{2}$, 0 ,-$\frac{side}{2}$ + *tlado*)

A partir desta fase, desenhamos todas as divisões desta fila da "grelha"de forma paralela ao eixo do x. Chegando ao fim desta fila, passamos para a próxima até obtermos uma grelha com *grid* filas e *grid* colunas.

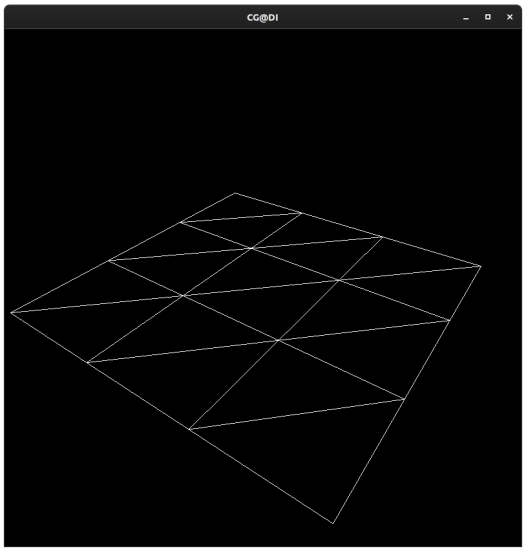


Figura 1: Plano com 3 unidades de comprimento e 3 divisões por eixo

3.2 Caixa

Esta primitiva geométrica é formada por 6 faces, em que cada face segue a mesma estratégia utilizada anteriormente no plano. Desta forma, para ser construída necessita do comprimento da aresta (*aresta*) e do número de divisões (*grid*), sendo cada face dividida numa grelha de *grid*x*grid* divisões. A construção de cada face é feita da mesma forma que foi feito o plano anteriormente referido, sendo desenhadas faces opostas ao mesmo tempo.

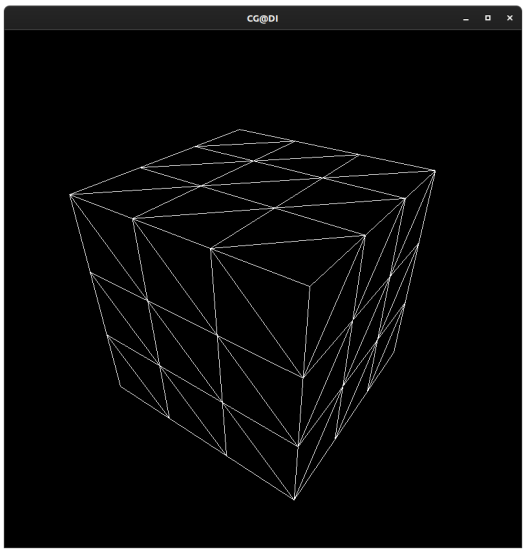


Figura 2: Caixa com 3 unidades de comprimento e 3x3 divisões por face

3.3 Cone

Para construir um cone é necessário um raio da sua base (*raio*), uma altura (*height*) e o número de divisões da base (*slices*) e da altura (*stacks*). O desenho desta primitiva pode ser separado em duas etapas: o **desenho da base** e o **desenho do plano curvo**.

Inicialmente, é feito o **desenho da base**, e uma vez que esta se encontra centrada na origem, fixámos um ponto inicial (0, 0, 0) e, depois, para sabermos o deslocamento angular, definimos o ângulo de cada divisão da base pelo seguinte cálculo: $espca = (2 * \pi) / slices$, uma vez que sabemos que a circunferência da base tem 2π radianos. Daqui obtemos às expressões que fornecem as coordenadas de um ponto atual e de um próximo ponto da circunferência iterando a variável i até percorrer toda a circunferência. Estes ponto em conjunto com o0 centro da circunferência formam os pontos de 2 triângulos

As coordenadas destes pontos são:

- **Ponto atual :** ($raio$*sin(i) , 0 , $raio$*cos(i))
- **Próximo Ponto :** ($raio$*sin(i+$espca$) , 0 , $raio$*cos(i+$espca$))

Posteriormente, para efetuar o **desenho do plano curvo**, observa-se que o raio decresce em cada divisão, desde a base até à altura máxima onde se torna 0. O espaçamento da altura (*esph*) é obtido dividindo a altura do cone pelo número de divisões verticais (*esph = height / stacks*). A variável *proxRaio* representa valor do raio da próxima divisão vertical do cone.

Através destes valores conseguimos obter as expressões que nos fornecem as coordenadas de uns pontos do cone a cada iteração, ou seja, cada vez que percorremos a circunferência (incremento do i) e que convergimos de encontro à altura pretendida (incremento do j):

1. (*raio* * sin(alpha) , *esph***i* , *raio* * cos(alpha))
2. (*raio* * sin(alpha + *espca*) , *esph**i , *raio* * cos(alpha + *espca*))
3. (*proxRaio* * sin(alpha) , *esph**(i + 1) , *proxRaio* * cos(alpha))
4. (*proxRaio* * sin(alpha + *espca*) , *esph**(i + 1) , *proxRaio* * cos(alpha + *espca*))

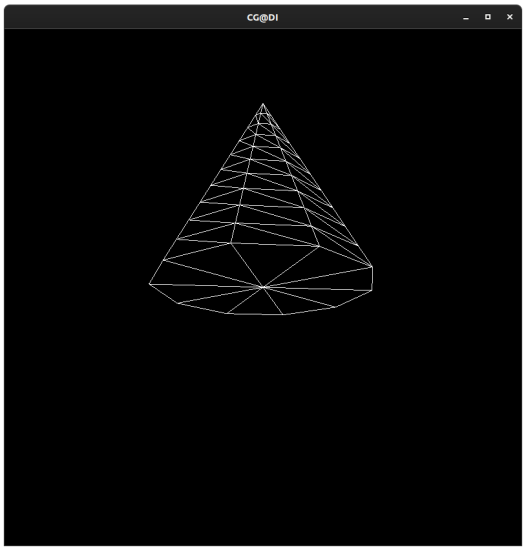


Figura 3: Cone com 1 unidade de raio, 2 unidades de altura, 10 divisões de base e 10 divisões de altura

3.4 Esfera

A esfera consiste num sólido geométrico cuja conjunto de pontos da sua superfície estão à mesma distância do centro. Para tal, o algoritmo para a construir necessita dos parâmetros: raio (*raio*), divisões da base (*slices*) e divisões da altura (*stacks*).

Inicialmente, foram calculados os deslocamentos que iriam ser úteis para o problema, ou seja, o horizontal (*espca*) que é dado pela expressão $espca = (2 * \pi)/slices$ e o vertical (*espcb*) que tem como expressão $espcb = \pi/stacks$.

Deste modo, começamos a desenhar no topo da esfera fatia a fatia, isto é, fixamos um ponto inicial e um próximo ponto, com y constante e x e z que variam o seu valor conforme o deslocamento horizontal. Como o ângulo alfa vai incrementando com as iterações do ciclo, quando este atingir o valor $2 * \pi$, passa a ser o ângulo beta a sofrer alterações, passando assim a atuar o deslocamento vertical.

As coordenadas dos pontos que constituem a esfera são do tipo :

1. (*raio**sin(alpha)*cos(beta) , *raio**sin(beta) , *raio**cos(alpha)*cos(beta))
2. (*raio**sin(alpha+*espca*)*cos(beta-*espcb*) , *raio**sin(beta-*espcb*) , *raio**cos(alpha+*espca*)*cos(beta-*espcb*))
3. (*raio**sin(alpha)*cos(beta-*espcb*) , *raio**sin(beta-*espcb*) , *raio**cos(alpha)*cos(beta-*espcb*))
4. (*raio**sin(alpha+*espca*)*cos(beta) , *raio**sin(beta) , *raio**cos(alpha + *espca*)*cos(beta))

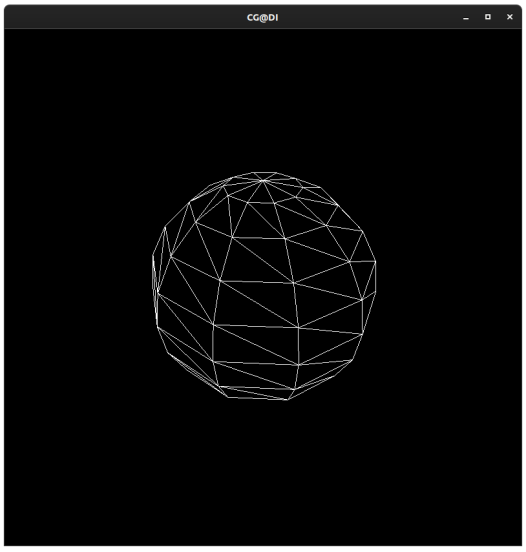


Figura 4: Esfera com 1 unidade de raio, 10 divisões de base e 10 divisões de altura

4 Utilização do Programa

4.1 Generator

Para utilizarmos o generator devemos inserir o comando referente à figura que queremos que surja, as suas dimensões e o nome do ficheiro resultante. De seguida estão os comandos e os argumentos para formar as primitivas.

- **Plano:** ./gen plano *dimensãoDoLado divisões ficheiroResultante*
- **Caixa:** ./gen box *dimensãoDaAresta divisões ficheiroResultante*
- **Cone:** ./gen cone *raio altura divisõesDaBase divisõesDaAltura ficheiroResultante*
- **Esfera:** ./gen sphere *dimensão divisõesDaBase divisõesDaAltura ficheiroResultante*

Desta forma, são criados os ficheiros output do programa, para este caso, com a extensão "*.3d*", não existindo obrigatoriedade de utilizar esta extensão.

4.2 Engine

O engine, como já referido anteriormente,é responsável pela leitura de ficheiros XML e por gerar os modelos 3D correspondentes. Para ler o ficheiro no terminal e obter o output em GLUT do modelo 3D desejado utiliza-se o comando :

./engine *NomeFicheiroXML*

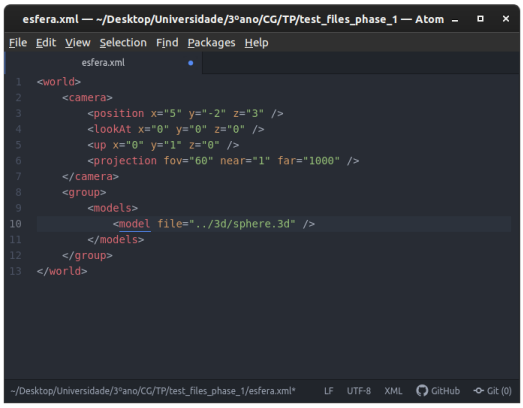


Figura 5: Exemplo de um ficheiro XML

Após a apresentação do modelo 3D, é possível utilizar comandos como W, A, S e D que movem a figura para observarmos o topo, a parte de baixo, a esquerda e a direita, respetivamente.

5 Conclusão

Considero que esta fase foi produtiva, visto que nos trouxe muita experiência na utilização de alguns recursos como OpenGlut e o GLUT, que serão úteis para a continuação do trabalho.

Um ponto importante a referir é o facto de utilizar para realizar o trabalho uma linguagem com a qual nunca utilizei em trabalhos práticos anteriores despertando, deste modo, interesse e vontade de conhecer esta nova linguagem.