Computação gráfica - Fase 1

Diogo Pires[a93239], Gonçalo Soares[a93286], Marco Costa[a93283], and Rita Teixeira,[a89494]

Universidade do Minho

1 Introdução

Tendo como objetivo desenvolver um *mini scene graph 3D engine*, nesta primeira fase focamos-nos, essencialmente, na construção de primitivas que nos ajudariam a chegar a esse propósito.

Assim, iremos descrever os vários tópicos desta fase, dando ênfase aos seguintes:

- Geração e renderização das diversas formas geométricas;
- Funcionamento e implementação do *engine*

Nos anexos apresentamos imagens das diversas formas desenhadas.

2 Generator

As diversas formas geométricas são formadas por um conjunto de triângulos, e escritas num formato definido por nós para um ficheiro que será lido, posteriormente, pelo *engine*. Este permitirá a visualização da forma geométrica gerada.

2.1 Leitura do Terminal

Tal como pedido no enunciado do projeto, o executável do programa recebe argumentos que o *generator* necessita para construir cada um dos modelos pedidos. O primeiro argumento identifica o tipo de modelo pretendido. Os outros argumentos diferem de modelo para modelo. Assim sendo, para os diferentes modelos são pedidos os seguintes tipos de argumentos:

- **Plano** - modelo (plane), comprimento, número de divisões e nome do ficheiro.
- **Cubo** - modelo (box), comprimento, número de divisões e nome do ficheiro.
- **Cone** - modelo (cone), raio da base, altura, *slices* (número de divisões no plano xOz), *stacks* (número de divisões no eixo y), e nome do ficheiro.
- **Esfera** - modelo (sphere), raio, *slices*, *stacks* e nome do ficheiro.

Este requisito foi relativamente simples de implementar, apenas tendo de implementar casos de *if...else* de maneira a verificar qual o modelo pretendido para assim podermos chamar a função que define essa mesma figura.

2.2 Plano

Para descrever o plano, começamos por pensar nele como um conjunto de quadrados, juntos lado a lado, como indica a figura seguinte, Fig. 1. Então, de forma a desenhar o plano, desenhamos os quadrados individuais, e para isso percorremos as esquinas inferiores esquerdas de cada um. A partir dessa esquina, desenhamos o quadrado, pela composição de dois triângulos.

As posições dos pontos azuis são calculadas a partir da origem, e para saber as suas posições avançamos segundo o número de divisões e comprimento que o utilizador escolhe. Esse cálculo parte da origem, e entre cada ponto avançamos $\frac{comprimento}{divisões}$ unidades.

O plano é desenhado a partir da origem do referencial, e é sempre no eixo xOz.

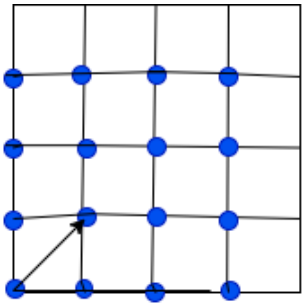

Fig. 1. Pontos iniciais na descrição do cubo.

2.3 Cubo

Para descrever o cubo, usámos o código desenvolvido para fazer um plano, alterando-o para cada um dos lados. O cubo tem uma das esquinas na origem do referencial, e a base é o plano xOz.

2.4 Esfera

Para descrever os pontos da esfera, utilizámos coordenadas polares, em radianos, com as seguintes caraterísticas:

1. **Alfa** - Descreve a posição nos eixos x e z, usando coordenadas polares para saber o ângulo, e o raio para saber a distância à origem.
2. **Beta** - Descreve a componente y do ponto, e o raio para saber a que distância deve estar da origem.
3. **Raio** - Na esfera, este é constante em todos os pontos, e é usado para calcular qualquer uma das componentes das coordenadas de um ponto.

De seguida avançamos a partir da linha central da esfera (idêntica ao *equador*), até às partes superior e inferior (pólos de cima e de baixo), por níveis, em função do número de *stacks*. Os níveis são analisados de forma simétrica, isto é, descrevemos ao mesmo tempo a parte superior e inferior da esfera. A imagem seguinte mostra dois níveis da esfera, com os círculos pretos que devem ser paralelos. A diferença dos ângulos beta entre os dois níveis é $\frac{\pi}{2 \times stacks}$, e varia entre 0 e $\pi/2$, na metade superior, e 0 e $-\pi/2$, na metade inferior da esfera.

Em cada nível, descrevemos os paralelogramos, assinalados a azul na figura seguinte, em função do número de *slices*. Para os descrever, dividimos cada um dos níveis em vários paralelogramos, e a diferença entre os alfa dos dois pontos da base e do topo é descrita por $\frac{2 \times \pi}{slices}$. O Alfa em cada nível varia entre 0 e $2 \times \pi$. Esses mesmo paralelogramos são depois decompostos em dois triângulos, linha vermelha que corta o paralelogramo em dois.

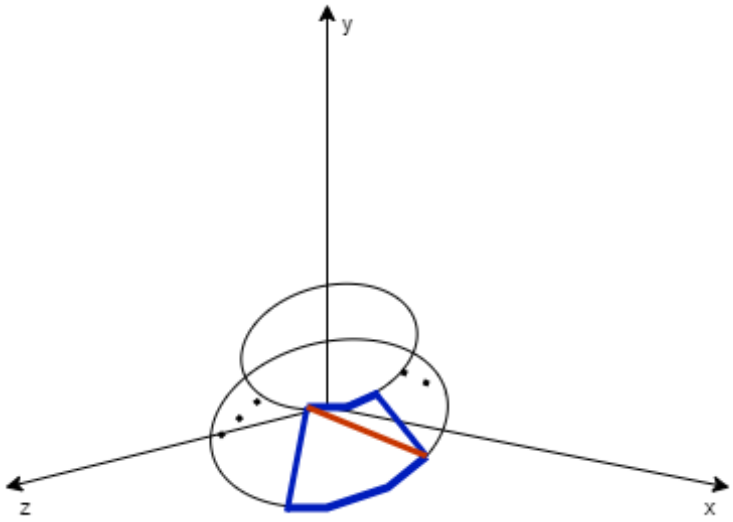

Fig. 2. Etapas para descrever a esfera.

2.5 Cone

Para descrever os pontos do cone, também utilizamos coordenadas polares. Primeiramente, definimos a base do cone. A função que realiza esta parte do modelo é relativamente simples, sendo que utilizamos as coordenadas polares anteriormente mencionadas para ver a base do cone em *slices* com grau alfa que varia entre 0 e $2 \times \pi$.

Quantos aos outros planos do cone, decidimos percorrer as várias *slices*, e em cada nível tratar de definir os triângulos. Da mesma forma que na esfera, desen-

hamos paralelogramos, dois triângulos, de cada vez. Para calcular o raio em cada posição, usámos semelhança de triângulos, usando a fórmula: $\frac{(Altura_2) \times raiodabase}{Altura_1}$.

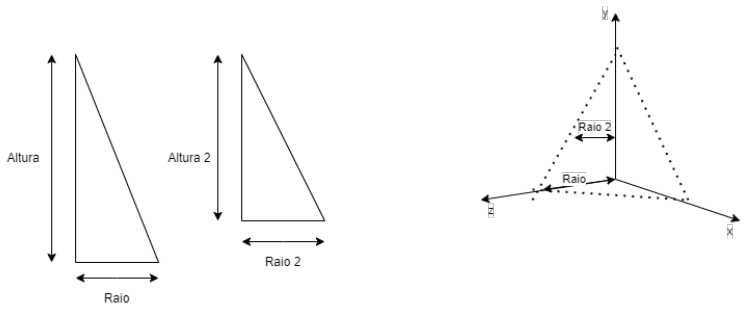

Fig. 3. Cálculo do raio nos diferentes níveis do cone.

2.6 Escrita no ficheiro .3d

Nesta última secção do *generator* tratámos de escrever os pontos necessários à construção pelo *engine* do modelo pedido. Através dos vários cálculos mencionados anteriormente, em vez de construir a figura, é criado um ficheiro do tipo .3d com o nome definido pelo utilizador no qual são escritos pelo programa os pontos que definem a figura. Por exemplo, na imagem em anexo, temos descrito um plano de comprimento 3 que é divido numa grelha de 4×4 quadrados. Uma caraterística comum a todos os modelos é eles serem compostos por triângulos, e cada triângulo é definido por 3 pontos. Por esta razão, decidimos que o ficheiro iria apresentar, em cada linha, os 3 pontos necessários à definição do triângulo, e em cada ponto as suas três coordenadas. A seguir, apresenta-se uma figura que demonstra um exemplo de como são apresentados os triângulos que constituem um modelo.

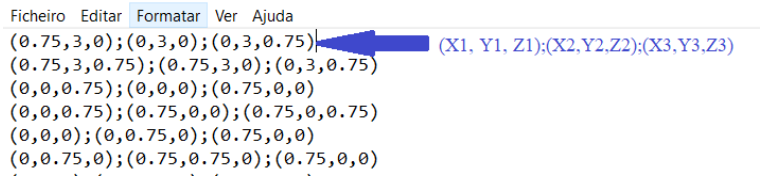

Fig. 4. Exemplo dum ficheiro tipo .3d.

3 Engine

3.1 Leitura do ficheiro XML

O caminho do ficheiro XML passado como argumento ao programa *Engine* é corretamente lido, ou seja, todos as suas *tags* que possuem nomes válidos no contexto do programa são aceites e os seus atributos e o seu texto são carregados. Dá se especial atenção que os *model's* especificados são guardados em memória durante toda a execução do programa, possibilitando assim que este ficheiro não necessite estar aberto em toda o ciclo de vida do *Engine*.

3.2 Cena

A cena é representada pela classe *Scene*. Esta contém toda a informação do estado do engine. Entre a informação guardada estão os vários modelos e as configurações da câmara, nomeadamente a posição, a rotação, o centro, o *up*, o campo de visão, o *near* e o *far*.

Durante a inicialização do programa, o estado e os modelos são lidos através dos ficheiros xml e 3d.

Depois de inicializado o *engine*, o utilizador é apresentado com o cenário descrito nos ficheiros. Agora o utilizador tem a possibilidade de mover e rodar os modelos e a câmara e aumentar ou diminuir o zoom desta usando o teclado.

4 Conclusão

Por fim, ficámos satisfeitos com o trabalho realizado, mas gostaríamos de ter melhorado a forma como descrevemos os pontos do cubo. Uma melhoria possível seria utilizar uma função que descreve um quadrado a partir de um ponto e um vetor. Dessa forma, seria possível usar a mesma função para as seis faces do cubo. A função que gostaríamos de ter implementado tem em conta se a face seria visível de cima ou de baixo.

5 Anexos

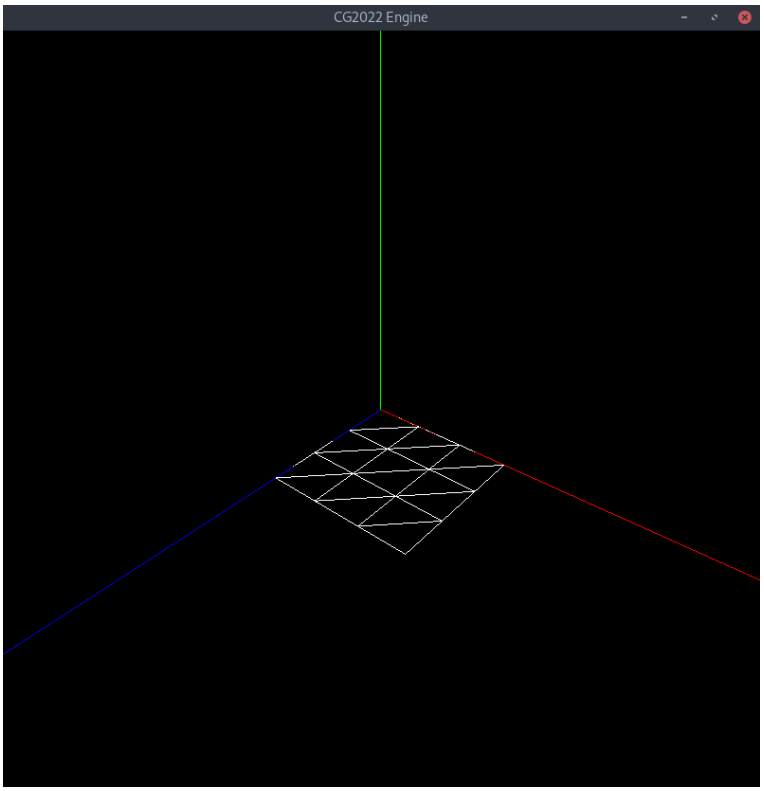

Fig. 5. Plano

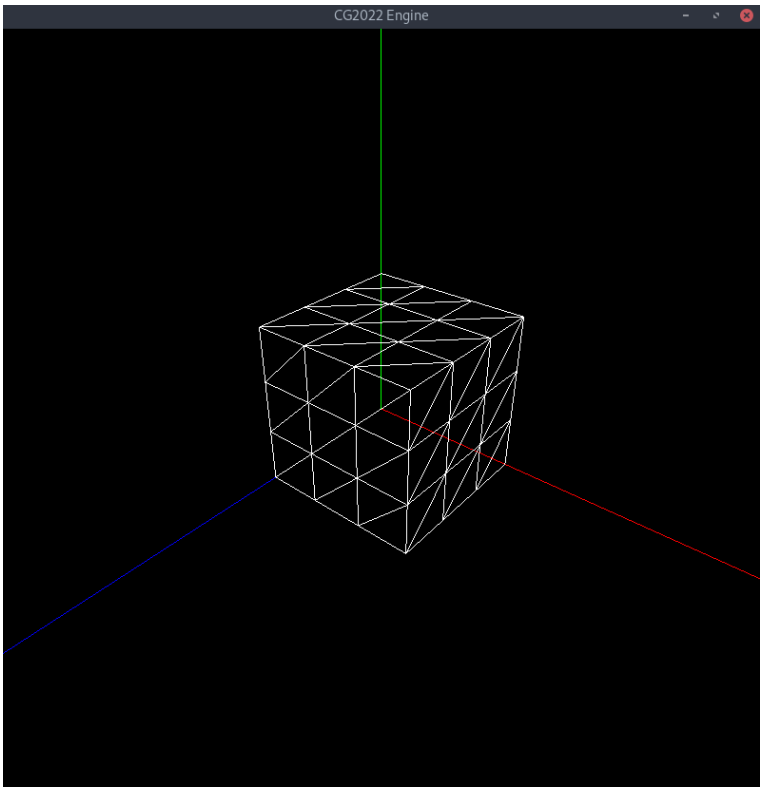

Fig. 6. Box

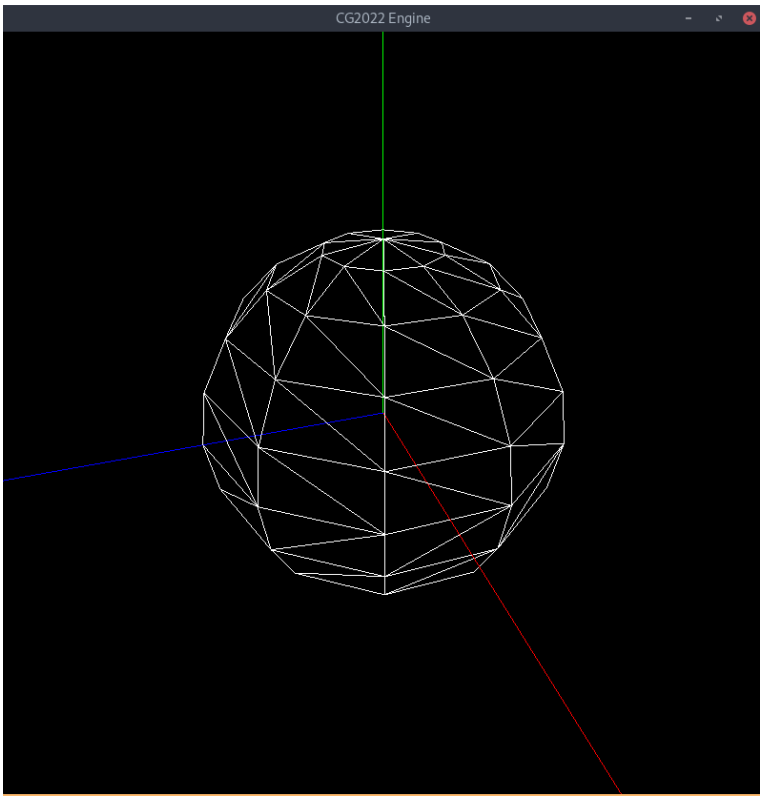

Fig. 7. Esfera

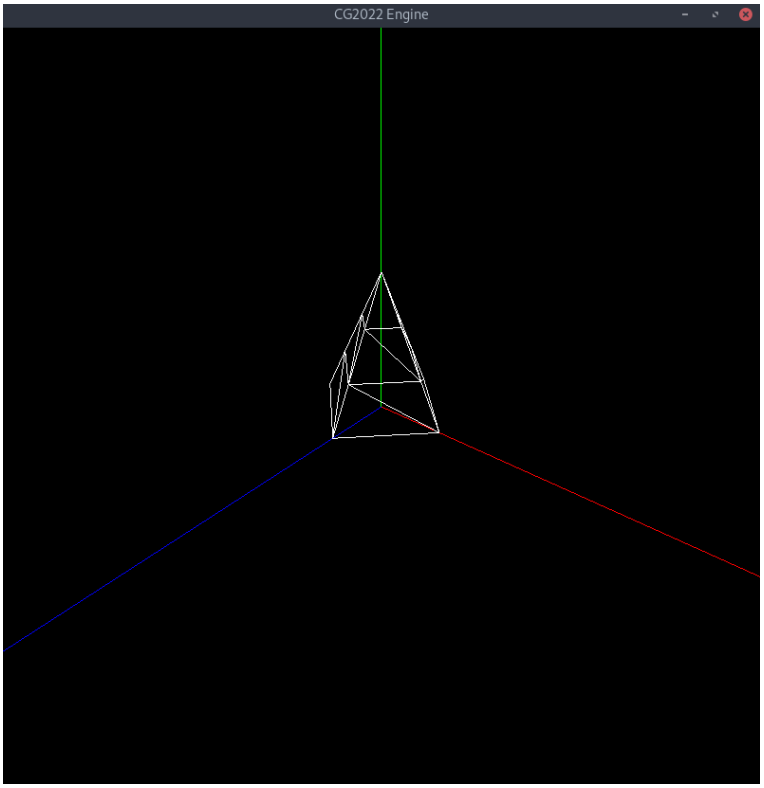

Fig. 8. Cone