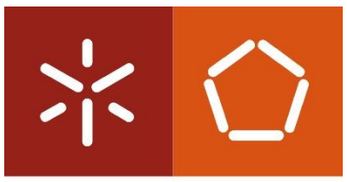


Universidade do Minho

Computação Gráfica
Fase 1

Benjamim Coelho, Henrique Neto, Leonardo Marreiros e
Júlio Alves

e-mail: {a89616,a89618,a89537,a89468}@alunos.uminho.pt

14 de março - 2021

Conteúdo

Introdução

O objetivo deste trabalho é desenvolver um motor gráfico capaz de representar diferentes primitivas ou modelos genéricos a três dimensões. Para isto, serão necessárias duas aplicações: o *engine* e o *generator* que têm objetivos diferentes. O *generator* é encarregue de gerar os pontos referentes a uma dada primitiva. Quanto ao *engine*, este tem como função representar os modelos criados a partir da leitura de ficheiros XML. Este projeto foi dividido em quatro partes.

Concretamente, nesta fase, foi desenvolvida a primeira versão do *generator* capaz de gerar os vértices das seguintes primitivas: plano, caixa, esfera e cone, tendo em consideração diferentes valores de entrada para parâmetros como a altura, largura, raio, número de divisões e número de *slices*. Do mesmo modo, foi desenvolvido um *engine* capaz de exibir a respetiva resposta gráfica a partir da leitura de ficheiros XML.

Arquitetura

2.1 Engine

De forma a renderizar as primitivas desejadas, utilizamos o engine. Este executável, com recurso ao GLUT e ao OpenGL, é responsável por dar parsing aos ficheiros XML que nos indicam quais os ficheiros .3d que contêm a informação relativa aos vértices necessários representar de forma a gerarmos a cena indicada. Para fazer o parsing dos XML, utilizamos o TinyXML2. Com este auxiliar, retiramos o nome dos ficheiros .3d que pretendemos. A partir daí, lemos os ficheiros .3d e guardamos toda a sua informação em memória.

Após termos toda a informação da cena em memória, só nos resta apresentar o resultado no ecrã. Para executar o engine, é necessário utilizar o comando "./engine (nome do ficheiro) [-axis]", sem esquecer que temos de indicar a extensão do ficheiro, .xml. Se o utilizador pretender ver os eixos do referencial com a cena basta adicionar a flag -axis"a seguir ao nome do ficheiro.

2.1.1 Models

Este ficheiro tem como função representar o produto dos vários ficheiros .3D criados anteriormente no *generator*. Para isto, para cada ficheiro, a partir de cada linha lida (cada linha é constituida por três floats) é criado um Point que é adicionado a um vector de pontos. A partir desse vector de pontos são depois desenhados todos os triângulos referentes à respetiva primitiva na função *draw_ model()*.

2.2 Generator

Como referido anteriormente, o gerador está encarregue de calcular os dados associados a uma dada primitiva. Para tal, este modulo distribui as suas funções por vários ficheiros que disponibilizam funções estruturais e computacionais do programa.

Para executar o *generator* é necessário utilizar um comando do tipo "./generator [primitiva] [argumentos necessários à primitiva] [nome do ficheiro .3d] [-xml]"O *generator* encarrega-se então de ler o argumento referente à primitiva e criar os ficheiros de coordenadas com base nos parâmetros das primitivas inseridos.

Adicionalmente este módulo também pode gerar um ficheiro XML com uma cena do objeto criado desde que seja indicado com a flag -xml". Desta forma programa invocará a função *createXML(char *string)* que utiliza a API do *tinyxml2* para construir o ficheiro com o formato pretendido com o mesmo nome que o ficheiro objeto produzido e respectiva extensão *.xml*.

2.2.1 Primitive

Esta classe tem como função escrever nos ficheiros .3D as coordenadas já calculadas para cada primitiva. Cada linha de ficheiros .3D é constituída pelas coordenadas x, y e z de um ponto, separadas por um espaço. Considerámos não ser necessário escrever o número de pontos como a primeira linha do ficheiro pois a sua leitura está a ser feita até o EOF pelo que a informação referente ao número de pontos seria redundante.

2.3 Utils

Esta secção constitui uma biblioteca comum e útil a ambas as aplicações. Nesta fase apenas contém a API do *tinyxml2* e a definição de uma classe Point que representa um ponto no espaço 3D.

Primitivas

3.1 Plano

Um plano é descrito matematicamente por uma primitiva geométrica infinita e bidimensional. Porém, dado as limitações computacionais existentes esta primitiva será aproximada a um quadrado com uma largura representativa do infinito que é definida pelo utilizador na invocação do programa. Desta forma o gerador calcula de forma direta quatro pontos no plano xOz nos quais forma dois triângulos. Por fim os triângulos resultantes formam o quadrado desejado. As figura 3.1 mostra um exemplo com comprimento L com os vértices $V_1 = (L; 0; L), V_2 = (-L; 0; -L), V_3 = (L; 0; -L)$ e $V_4 = (-L; 0; L)$ e os respectivos triângulos ($V1 \rightarrow V2 \rightarrow V4$ e $V1 \rightarrow V3 \rightarrow V2$) formados entre eles.

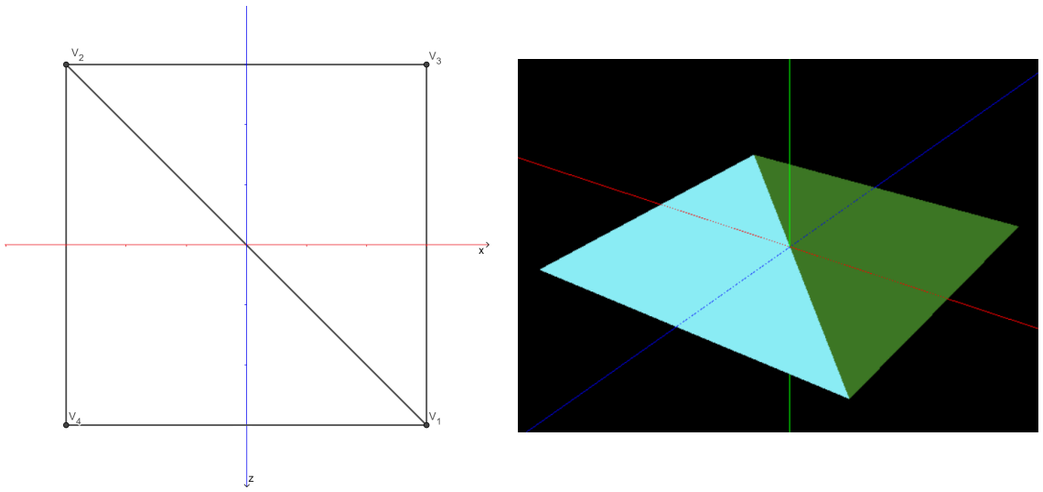


Figura 3.1: Aproximação de um plano

3.2 Caixa

Esta primitiva consiste na construção de um prisma retangular. Para esse efeito, recebe como argumentos as dimensões de comprimento, largura e altura. Opcionalmente, pode receber o número de divisões por aresta, que vai influenciar o nº de triângulos utilizados na sua construção (seja n o número de divisões, cada face será constituída por 2^n triângulos) . Por omissão, o número de divisões é 1. Por fim o algoritmo calcula iterativamente para cada divisão todos os triângulos associados a cada lado.

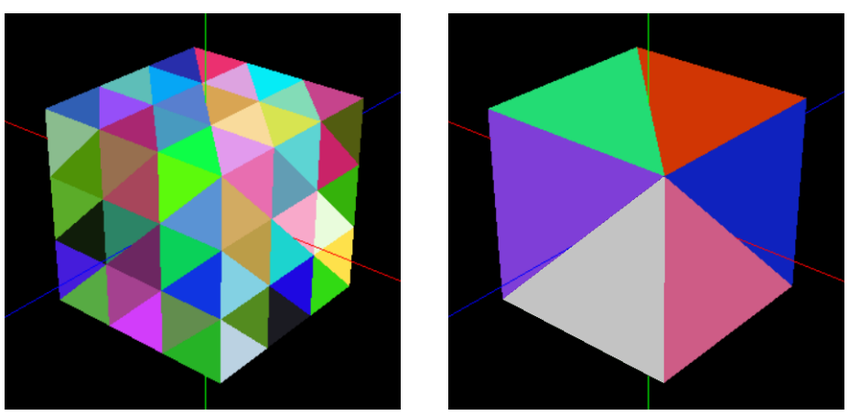
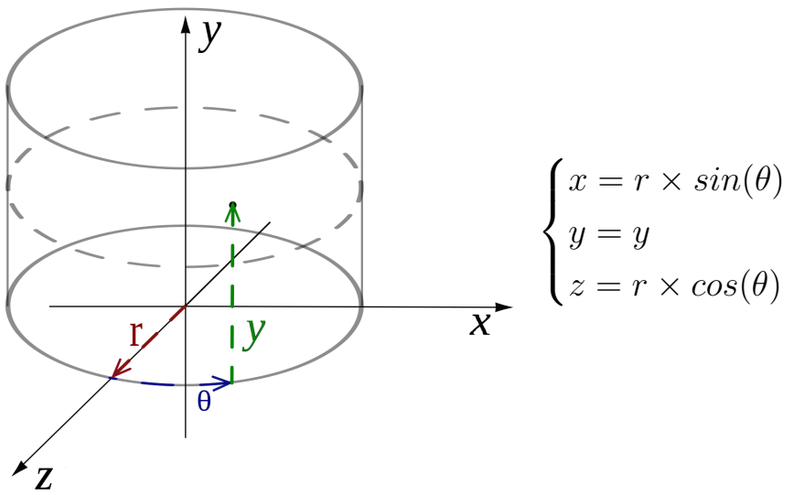


Figura 3.2: Exemplo de caixas com e sem divisões

3.3 Cone

Um cone é um polígono regular que representa uma pirâmide com base circular. Tendo isto em conta é evidente que seria necessário um número infinito de triângulos para representar as superfícies curvas consequentes. Ora como tal é impossível num ponto de vista computacional, iremos aproximar a base a um polígono regular, ou seja a um *n-ágono* com n lados e consequentemente dividiremos o polígono em varias n fatias (*slices*). Adicionalmente também iremos dividir o cone em várias pilhas(*stacks*) paralelas a base de forma a permitir um maior controlo da precisão na construção. Desta forma ao criar esta primitiva, o utilizador para além da altura e do raio do cone, terá também de indicar o número de *slices* e *stacks* que pretende.

O cone por fim é gerado *slice* a *slice* em que primeiro é calculado o triângulo da base seguido iterativamente pelos triângulos adjacentes até ao topo da fatia. É de notar que na construção desta primitiva o algoritmo gerador recorre a um sistema de coordenadas polares (exemplificadas na figura 3.3) para poder facilmente a calcular os pontos intervenientes apenas com o raio e a altura do cone.



$$\begin{cases} x = r \times sin(\theta) \\ y = y \\ z = r \times cos(\theta) \end{cases}$$

Figura 3.3: Exemplo de um sistema de coordenadas polares

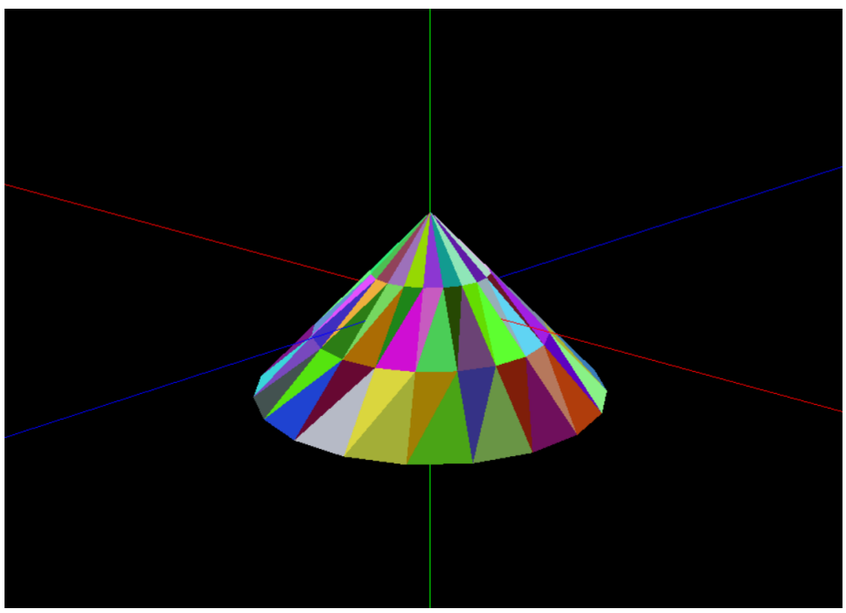


Figura 3.4: Exemplo de um cone com 15 slices e 3 divisões

3.4 Esfera

A esfera é um solido geométrico com uma superfície contínua em que todos os pontos encontram-se à mesma distância do centro. Esta distância denomina-se como o raio. Semelhante ao cone, como, na realidade, esta superfície é completamente curva, seria necessário um número infinito de triângulos para aproximar o modelo à realidade, o que é computacionalmente impossível. Com isto, o objeto desenhado será antes uma aproximação de uma esfera, ou seja consistirá de triângulos em que apenas os vértices destes cumprem as restrições do objecto. Estes vértices são definidos pela divisão da superfície esférica em várias fatias (*slices*) e pilhas (*stacks*). Desta forma ao criar esta primitiva, o utilizador para além do raio da esfera, terá também de indicar o número de *slices* e *stacks* pretendidos.

Por fim, a esfera é gerada *stack* a *stack* onde posteriormente cada *slice* é gerada sequencialmente. É de notar que na construção desta primitiva o algoritmo gerador recorre a um sistema de coordenadas esféricas (exemplificadas na figura 4.1) de modo a calcular, facilmente, os pontos que a constituem, apenas com o raio, a *stack* atual e a *slice* atual. De forma a diminuir o número de cálculos feitos, os valores trigonométricos associados a cada *slice* são pré-calculados e guardados antes do algoritmo de forma a não se repetirem cálculos.

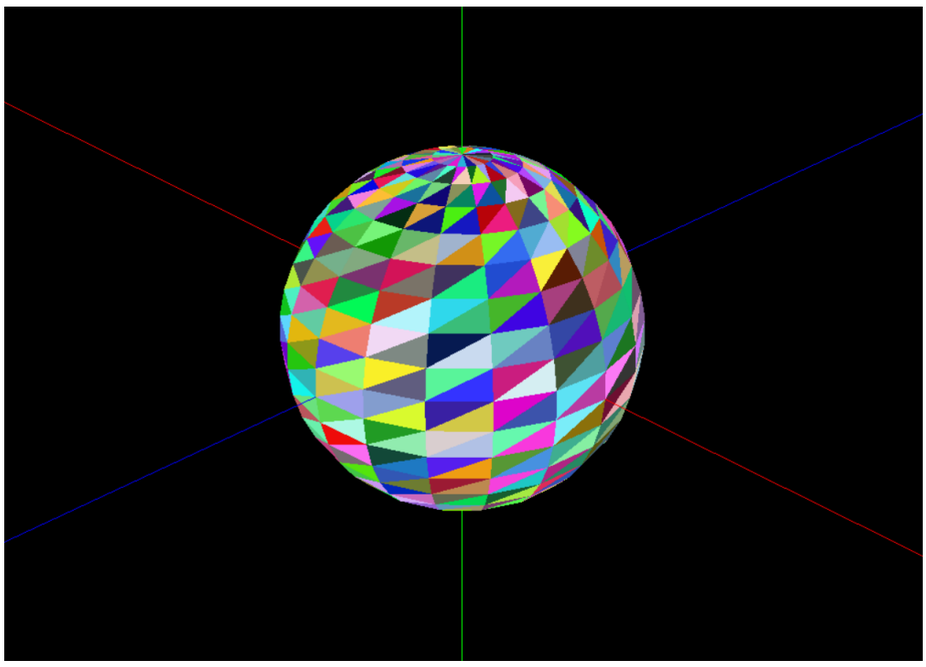


Figura 3.5: Exemplo de uma esfera com 20 stacks e 20 slices

Câmera

De modo a ser possível verificar a correta construção destas primitivas implementámos uma câmara em modo de explorador. Utilizando um sistema de coordenadas esféricas para o efeito, ao manter o ponto de observação na origem do plano, para mover a câmara basta mudar os ângulos beta e alpha 4.1.De seguida, podemos converter as coordenadas esféricas para coordenadas cartesianas e aplicar essas coordenadas na sua posição, criando assim uma câmara que se movimenta à volta do nosso modelo. Podemos também aproximarmos-nos e afastarmos-nos do modelo, incrementando ou decrementando o valor r (distância da câmara à origem). Os controlos são os seguintes:

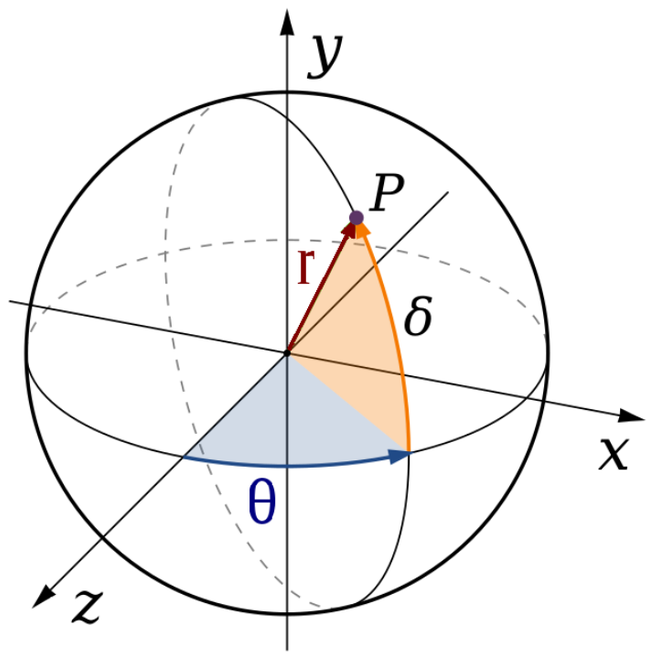


Figura 4.1: Exemplo de um sistema de coordenadas esféricas

- W - Mover a câmara para cima;
- S - Mover a câmara para baixo;
- A - Mover a câmara para a esquerda;
- D - Mover a câmara para a direita;
- ↑ - Aproximar a câmara da origem;
- ↓ - Afastar a câmara da origem.

Conclusão

Nesta primeira fase do projeto, conseguimos aprofundar o nosso conhecimento de C++, OpenGL e Glut, na medida em que fomos capazes de consolidar o que aprendemos nas aulas de Computação Gráfica e aplicar esses mesmos conhecimentos aos requisitos desta fase.

Na verdade, este trabalho apresentou-nos novos desafios, como foi o caso de aplicar o número de divisões na primitiva da box e o número de stacks nas primitivas da sphere e cone. Apesar destas dificuldades, acreditámos que conseguimos implementar todas as funcionalidades de forma bastante satisfatória.

Para além disso, este projeto obrigou-nos ao estudo da biblioteca tinyxml2, estudo esse que, apesar de desafiante, é um ótimo treino para uso de bibliotecas externas a uma linguagem de programação, o que é uma grande vantagem no mundo da programação.

Assim sendo, conseguimos fazer com que todas as primitivas sejam geradas corretamente e que o motor gráfico seja capaz de as representar genericamente pelo que podemos afirmar que foram cumpridos todos os objetivos propostos na primeira fase para além de termos implementado uma câmara em modo explorador por iniciativa própria.