

**Universidade do Minho**

Licenciatura em Ciências da Computação

**Unidade Curricular de**

**Computação Gráfica**

Ano Letivo de 2018/2019

**José Boticas A81241, Gonçalo Costa A82405, Ricardo Vieira A81640**

Março, 2019

**Índice**

[1. Introdução 3](#_Toc3802074)

[1.1. Descrição do problema 3](#_Toc3802075)

[2 Aplicações 4](#_Toc3802076)

[2.1 *Generator* 4](#_Toc3802077)

[2.2 *Engine* 4](#_Toc3802078)

[3 Implementação 5](#_Toc3802079)

[3.1 Fase 1 5](#_Toc3802080)

[3.1.1 Plano 5](#_Toc3802081)

[3.1.2 Caixa 5](#_Toc3802082)

[3.1.3 Esfera 6](#_Toc3802083)

[3.1.4 Cone 7](#_Toc3802084)

[3.2 Fase 2 8](#_Toc3802085)

[3.2.1 Ficheiro XML 8](#_Toc3802086)

[3.2.2 Aplicação das transformações 8](#_Toc3802087)

[4 Conclusões 9](#_Toc3802088)

## Introdução

O objetivo deste trabalho é o desenvolvimento de um motor gráfico 3D e mostrar os vários exemplos que mostram o seu potencial, utilizando o *OpenGL* e a biblioteca *GLUT*.

### Descrição do problema

Foi-nos então pedido o desenvolvimento de duas aplicações: um gerador de ficheiros com a informação de vários modelos, neste caso, só os vértices, e um motor que lê o ficheiro, escrito em XML, e que desenhe o modelo.

Para criar um ficheiro com o modelo, o gerador vai receber como parâmetros o tipo da primitiva gráfica, os outros parâmetros necessários para criação, por exemplo as dimensões, e o ficheiro onde os vértices vão ser guardados.

As primitivas gráficas pedidas são:

* **Plano** (um quadrado no plano *XZ*, com centro na origem, feito com 2 triângulos);
* **Caixa** (requer *X*, *Y* e *Z* como dimensões e opcionalmente o número de divisões);
* **Esfera** (requer o raio e o número de *slices* e *stacks*);
* **Cone** (requer o raio da circunferência de baixo, altura e número de *slices* e *stacks).*

## Aplicações

## *Generator*

O programa *generator* gera os ficheiros dos modelos. Para efeito criaram-se as funções *drawPlane*, *drawBox*, *drawCone*, *drawSphere*, que escrevem, nos ficheiros acima mencionados, as coordenadas dos pontos pertencentes a um conjunto de triângulos. Cada linha do ficheiro contém um conjunto de três números, correspondentes a um vértice. Cada triângulo é constituído por três vértices.

## *Engine*

O *engine* irá ler a configuração do ficheiro, em *XML* e mostrar os respetivos modelos 3D.

Para tal, recorremos à biblioteca *TinyXML2* para processar os ficheiros *XML*. O nº raiz destes ficheiros corresponde à *tag* *<scene>*, que contém os elementos *<model>* que, por sua vez, possuem como atributo o caminho para o ficheiro de pontos aí representado.

De seguida, são carregados para a memória os pontos do ficheiro supracitado e desenhados no ecrã os triângulos, formando o modelo 3D desejado.

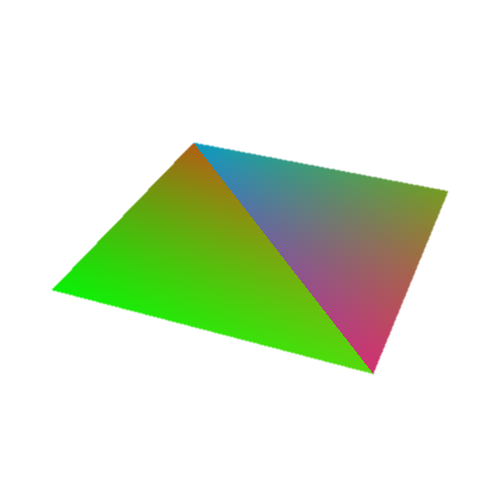
## Implementação

Neste capítulo, são abordadas as implementações essenciais ao desenvolvimento de cada uma das fases do projeto.

### Fase 1

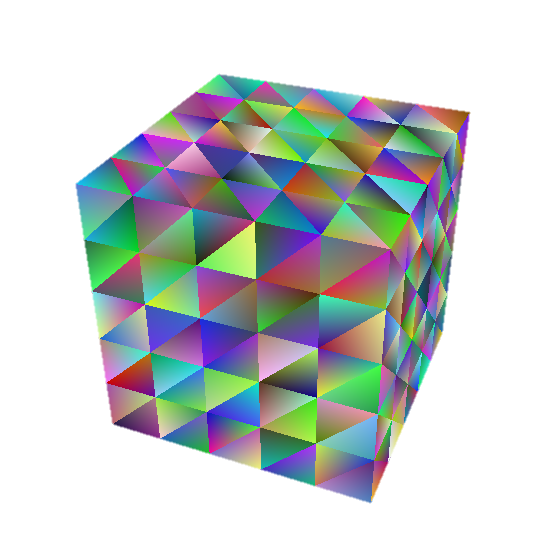
A primeira fase prende-se pela implementação de todas as primitivas pedidas. De notar que sempre que se falar em desenhar um triângulo, quer-se, na verdade, falar em guardar os vértices de um triângulo, já que o desenho é feito pelo motor desenvolvido.

### Plano

Para desenhar um quadrado/retângulo no plano *XZ*, desenham-se dois triângulos. Estes têm dois vértices em comum (os vértices de uma das diagonais do quadrado). Para que o plano possa ser visto quer de cima, quer de baixo, decidimos desenhá-lo por duas vezes, sendo assim criados quatro triângulos.

### Caixa

Para desenhar a caixa, escreveram-se dois ciclos que iteram até ao número de divisões pedidas (1 por *default*). Vendo o exemplo da face superior, mantém-se o *y* constante e calculam-se as coordenadas de *x* e *z*. Para isso, estas o *x* e o *z* são iniciados com o valor mínimo do mesmo, ou seja, -*x/2* e -*z/*2. O valor de *x* é incrementado em *x/divisões* a cada iteração do ciclo interior e restaurado para o seu valor inicial, mencionado atrás, a cada iteração do ciclo exterior. O valor de *z* é incrementado em *z/divisões* a cada iteração do ciclo exterior. Assim, conseguimos desenhar a face superior com as divisões pedidas.

O desenho das restantes faces é análogo. Importa referir que, por comodidade, são guardadas duas variáveis correspondentes a *x*, *y* e *z*.

### Esfera

No desenvolvimento da esfera, começou por se pensar apenas na metade superior da mesma. O processo iniciou-se com o desenho de tantas circunferências quantas as *stacks* pedidas e depois ajustaram-se as suas alturas e raios das mesmas, de forma a termos uma estrutura e percebermos quais seriam os vértices dos triângulos.

A partir daqui tudo ficou mais facilitado e tivemos apenas de ir desenhando uma a uma as *stacks*, guardando, para isso, dois raios e duas alturas (do topo e da base da *stack*).

As *slices* determinaram quantos triângulos seriam desenhados por cada *stack*.

Foram então desenvolvidos dois ciclos *for* aninhados, um para controlar o número de *stacks* e outro para se desenharem os triângulos de cada uma delas. Os raios e as alturas eram atualizados a cada iteração do primeiro ciclo.

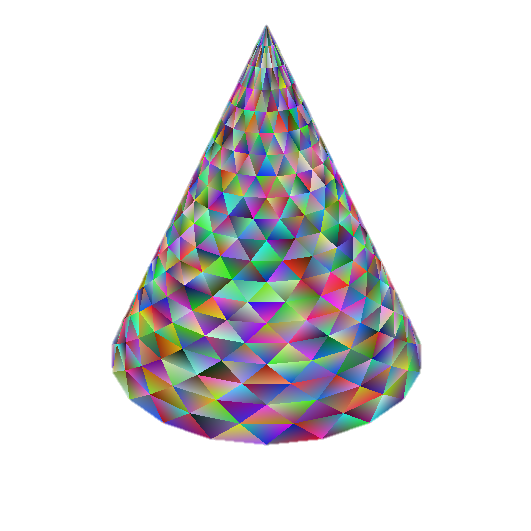
Uma imagem com balão, aeronaves, transporte

Descrição gerada com confiança muito altaA metade inferior da esfera é uma cópia da metade superior em que se alteram apenas os valores de *y* para o seu simétrico, ou seja, para -*y*. Tem também de se ter em conta a ordem por que são escritos os vértices.

### Cone

Para o desenho do cone, foi utilizada a mesma estratégia usada no desenvolvimento da esfera. Começou-se então por se pensar em várias circunferências. Desta vez, o raio de cada uma das circunferências diminui de forma constante. Esta diminuição é igual à divisão do raio da base pelo número de *stacks*. As alturas das circunferências calculam-se da mesma forma (começa em 0 e aumenta de forma constante e igual à divisão entre a altura e o número de *stacks*). Mais uma vez, o número de *slices* é igual ao número de triângulos de cada *stack.*

Assim, começamos por desenhar a base do cone, que consiste numa circunferência desenhada com o número de triângulos igual ao número de slices. De seguida, desenham-se as *stacks*, uma a uma.

Foi então escrito um ciclo *for* para o desenho da base e outros dois ciclos *for* aninhados, seguindo a mesma estratégia do desenho da esfera. De realçar que a diferença entre estas duas figuras se prende pela forma como eram calculados os raios das circunferências que serviram de base para o pensamento.

### Fase 2

A segunda fase passa por tratar as transformações geométricas permitindo a construção de um Sistema Solar estático.

### Ficheiro XML

As principais alterações nesta fase dão-se a nível do ficheiro XML. Este passa a conter a informação de cada planeta num grupo (figura 1 #METER), desde as transformações nele aplicadas (scale, rotate, translate) até a outros grupos que podem representar os seus satélites, naturais ou artificiais. De notar que a ordem das transformações associadas a cada grupo tem influência no resultado e que estas também são aplicadas nos subgrupos do respetivo grupo.

### Aplicação das transformações

Foi alterada a aplicação Engine de modo a guardar as informações do ficheiro XML. Para isso criaram-se novas estruturas de dados: uma para guardar transformações e outra para guardar os grupos, sendo que esta última inclui, para além da estrutura de vértices já existente, a primeira.

Esta leitura é feita linha a linha, identificando as tags e guardando as transformações, vértices e subgrupos, grupo a grupo.

#MELHORAR ABAIXO

O desenho é feito grupo a grupo desenhando os vértices e aplicando as transformações pela ordem em que estes são lidos. Para o desenho dos subgrupos segue-se a mesma lógica, uma vez que se trata de um grupo, sendo que estes aplicam, no entanto, as transformações do grupo “pai”. Para isto fazemos uso das funções glPushMatrix() e glPopMatrix() no início e fim de cada grupo, respetivamente.

## Conclusões

* 1. **Fase 1**

O desenvolvimento deste pequeno trabalho permitiu o desenvolvimento de algumas primitivas gráficas, essenciais para o que virá a ser o projeto final. Permitiu ainda, aos elementos deste grupo, compreender algumas noções da computação gráfica e desenvolver capacidades na utilização das ferramentas utilizadas para a aprendizagem da mesma.

Por outro lado, os códigos de geração dos modelos foi onde surgiu uma maior dificuldade, sobretudo na construção dos algoritmos, em especial no da esfera. Contudo, após debate de grupo, conseguiu-se finalizar esta fase.