

UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

**TRABALHO PRÁTICO – Fase 4**

# Normals and Texture Coordinates

Computação Gráfica  
Maio de 2021

## Autores:

Ana Filipa Pereira A89589

Carolina Santejo A89500

Raquel Costa A89464

Sara Marques A89477

# Índice

---

Introdução e Principais Objetivos .....	3
Arquitetura do Programa .....	4
Motor .....	5
VBO'S com índice .....	5
Iluminação.....	5
Texturas .....	5
Estrutura de Dados – Atualização .....	6
Leitura e Parse do Ficheiro XML – Atualização .....	6
Leitura do Ficheiro tipo 3D .....	6
Keybinds.....	6
Gerador .....	7
Vetores Normais .....	7
Plano .....	7
Caixa.....	8
Esfera .....	8
Cone .....	9
Torus .....	9
Bezier Patches.....	10
Coordenadas de Textura.....	10
Plano .....	10
Box .....	10
Esfera .....	11
Cone .....	12
Torus .....	12
Bezier Patches.....	12
Geração do Ficheiro do tipo 3D – Atualização.....	12
Demos .....	13
Sistema Solar.....	13
Plano .....	13
Esfera .....	14
Torus .....	15
Cone.....	16
Caixa.....	17
Teapot.....	17

## Introdução e Principais Objetivos

---

Nesta quarta e última fase do projeto prático da unidade de Computação Gráfica, foi pedido que tornássemos o sistema solar desenvolvido nas fases passadas mais realista através da adição de texturas e de iluminação. Para tal, foi necessário modificar o *generator* e o *motor*. No gerador, para cada primitiva, passaram a ser calculados os pontos de textura e os vetores normais. O *engine* foi alterado de forma a conseguir ler os ficheiros gerados. Além disto, foi preciso modificar também os *XML*.

Assim sendo, ao longo deste relatório será descrito todo o processo de desenvolvimento da fase além de serem explicadas todas as decisões tomadas.

# Arquitetura do Programa

---

Nesta fase do projeto, foi adicionada a classe `model`, responsável por gerar VBOs, carregar imagens de textura e gerir os diferentes tipos de luz existentes (difusa, ambiente, especular, emissiva) e as suas cores.

Foi também adicionada em `Demos` a pasta `texImages`, que contém todas as imagens a serem utilizadas como texturas pelo nosso programa.

# Motor

---

## VBO'S com índice

Tendo implementadas as estruturas VBO's na fase anterior, o grupo chegou à conclusão que, por um lado, a leitura dos ficheiros .3d estava a ser demorada devido à existência de um número elevado de pontos por cada figura e por outro, estava a ser gasto desnecessariamente demasiado espaço na memória da placa gráfica. Nesta fase, a necessidade da inclusão de vetores normais e pontos de textura justificou a conveniência da implementação de VBO's com índices.

Com esta alteração, todos os pontos passam a ser identificados pelos seus índices, o que leva à redução do número de vértices e consequentemente, do número de vetores normais e pontos de textura a guardar. Desta forma, apesar de atualmente se armazenar mais informação foi notável uma maior rapidez principalmente na leitura dos ficheiros .3d.

Para gerar os buffers respetivos para cada modelo lido foi utilizada a função `generateVBOs` da classe `model` que para cada conjunto de valores utiliza as funções `glGenBuffers`, `glBindBuffer` e `glBufferData` para alocar e copiar os dados para a memória da placa gráfica.

## Iluminação

Para definir a iluminação de uma cena gerada pelo nosso programa foi necessário caracterizar, não só a luz que ilumina todo o espaço, como também a iluminação que cada uma das figuras desenhadas pode possuir.

Para a luz principal, foi admitido que pode ser de três tipos: posicional, direcional ou ser um foco de luz. Para a primeira só foi necessário definir a sua posição, para o caso da segunda é estabelecido o vetor da direção e para a última são necessários ambos os parâmetros das duas primeiras. Para ativar a iluminação foram utilizadas as funções do OpenGL: `glEnable(GL_LIGHTING)` e `glEnable(GL_LIGHT0)`. Consequentemente, para definir o tipo de luz utilizou-se a função `glLightfv`.

Quanto à iluminação dos materiais admitiu-se que podem possuir todas as quatro componentes da cor: difusa, especular, ambiente e emissiva. Para cada uma delas apenas foi necessário definir a sua cor com recurso ao método `glMaterialfv`, tendo em conta que para a especular se definiu por default o valor do `shininess` a 128.

## Texturas

Dada a necessidade de aplicação de texturas às figuras desenhadas foi necessário carregar as todas imagens necessárias para a memória da placa gráfica. Com recurso à função `loadTexture` da classe `model`, foi carregada a imagem de cada um dos modelos dado o seu nome indicado previamente no ficheiro XML.

Para o seu desenho, inicialmente foram ativadas as funcionalidades das texturas do OpenGL chamando as funções `glEnable(GL_TEXTURE_2D)` e

`glEnableClientState(GL_TEXTURE_COORD_ARRAY)` e logo a seguir foram ativados tanto o buffer com os dados da imagem como também os respetivos pontos de textura previamente lidos no ficheiro .3d que relacionam as posições do referencial e da imagem.

## Estrutura de Dados – Atualização

A adição de iluminação e texturas e a alteração para estruturas VBO's com índices nesta fase do projeto implicou a alteração da estrutura de dados que armazena a todas as informações necessárias ao desenho de todas as figuras.

Ao contrario da fase anterior onde era possível agrupar todos os vértices das figuras no mesmo array e ir desenhando parcialmente de acordo com o grupo que estivessemos a ler, nesta foi necessário, dentro de cada group separar os dados constituintes de cada modelo.

Deste modo foi criada uma nova classe model que guarda em primeiro lugar todos os valores dos vértices, índices, vetores normais e pontos de textura lidos a partir do ficheiro .3d, assim como o nome da imagem da textura (se não possuir textura é uma string vazia) referenciado no XML.

Uma vez que estamos perante a utilização de VBO's foi também necessário gerar e copiar todos os dados para a placa gráfica, logo a classe model constitui também 5 buffers do tipo `GLuint`, em que os primeiros quatro incluem os dados previamente guardados já referidos e o ultimo é a imagem da textura já carregada para a memória. A geração dos buffers e respetiva copia dos dados é feita apenas uma vez no momento em que for desenhada a cena pela primeira vez.

Assim sendo, a classe group é constituída pelos vetores das transformações (estáticas e dinâmicas) e dos grupos filhos tal como na fase anterior e é agora adicionado também um vetor de elementos da classe model.

## Leitura e Parse do Ficheiro XML – Atualização

Raquel + lights!

## Leitura do Ficheiro tipo 3D

Raquel

## Keybinds

Com recurso ao método `glutKeyboardFunc` foram atribuídas algumas funcionalidades a certas teclas do teclado quando o motor é executado. Cada uma delas pode ser especificada na tabela seguinte:

Tecla(s)	Ação
<b>1</b>	Ativar camara 1
<b>2</b>	Ativar camara 2
<b>x</b>	Mostrar/ocultar eixos do referencial
<b>o</b>	Mostrar/ocultar órbitas das curvas de <i>catmull-rom</i>
<b>b</b>	Mostrar/ocultar imagem de fundo
<b>Camara 1</b>	
<b>Botão esquerdo do rato</b>	Pressionar e arrastar os lados para rodar a camara

<b>Botão direito do rato</b>	Pressionar e arrastar para cima/baixo para aproximar/afastar a camara
<b>Camara 2</b>	
<b>w</b>	Mover para a frente da direção do olhar
<b>s</b>	Mover para trás da direção do olhar
<b>a</b>	Mover para a esquerda da direção do olhar
<b>d</b>	Mover para a direita da direção do olhar
<b>Botão esquerdo do rato</b>	Pressionar e arrastar para rodar a camara

## Gerador

Nesta fase o gerador foi alterado para, além de fornecer as coordenadas de cada ponto das primitivas (plane, box, sphere, cylinder, torus) e os índices dos VBOs, gerar também as coordenadas normais e as coordenadas de textura de cada ponto. Todos estes dados serão então guardados num ficheiro que será lido pelo motor.

As normais dos pontos serão necessárias para que o OpenGL possa desenhar sombras correspondente às fontes de luz presentes em cada primitiva. A normal de cada ponto é calculada por face, logo, pontos que pertencem a múltiplas faces (como vértices ou pontos em arestas) terão as suas coordenadas registadas n vezes, sendo n o número de faces ao qual o ponto pertence.

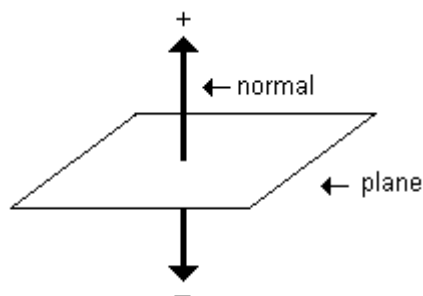
As coordenadas de textura serão utilizadas para aplicar texturas às primitivas, sendo estas representadas por um ponto (x,y,z). As coordenadas (0,0,0) correspondem ao canto inferior esquerdo de uma imagem de textura. Ao percorrer a imagem horizontal ou verticalmente, x e y aumentam respetivamente até atingirem o valor máximo 1, sendo que z se mantém sempre 0, sendo uma coordenada inútil nestas circunstâncias. O OpenGL irá, depois, utilizar estas coordenadas para saber que ponto da imagem de textura corresponde a que ponto da primitiva em questão.

## Vetores Normais

Nesta fase, cada uma das primitivas, passou a ter um vetor, no qual são colocadas todas as suas normais calculadas.

### Plano

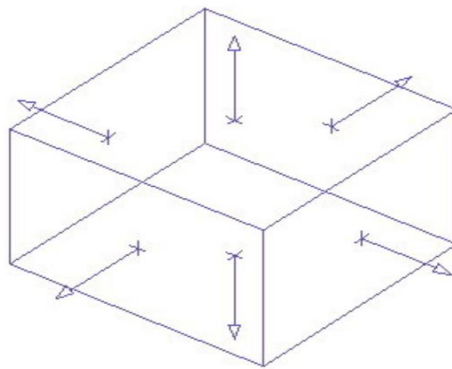
O cálculo das normais num plano, é bastante simples. Sendo que esta primitiva se encontra no plano XZ, os vetores normais serão (0,1,0) para todos os pontos da face voltada para cima e (0,-1,0) para todos os pontos da face voltada para baixo.



## Caixa

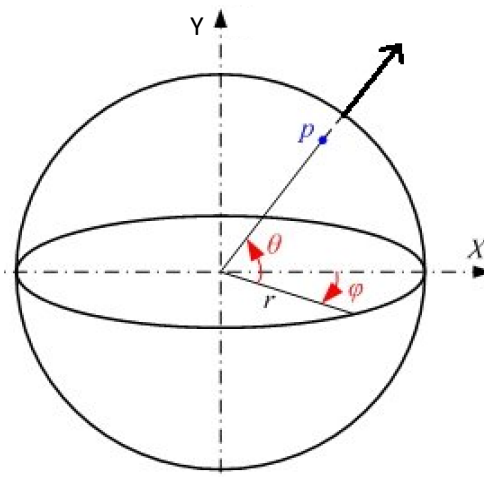
Visto que o cubo possui seis faces planas e que para todos os pontos de uma face o vetor normal é o mesmo, concluímos o seguinte:

- o vetor normal dos pontos da face da frente é  $(0,0,1)$
- o vetor normal dos pontos da face de trás é  $(0,0,-1)$
- o vetor normal dos pontos da face da direita é  $(1,0,0)$
- o vetor normal dos pontos da face da esquerda é  $(-1,0,0)$
- o vetor normal dos pontos da face de cima é  $(0,1,0)$
- o vetor normal dos pontos da face de baixo é  $(0,-1,0)$



## Esfera

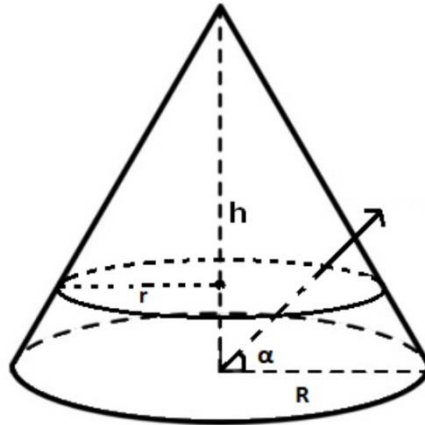
No caso da esfera, o vetor normal num dado ponto é igual ao vetor normalizado do vetor que vai do centro da esfera até ao ponto em questão. Assim sendo, a normal em qualquer ponto é a seguinte:  $(\cos(\theta) * \sin(\varphi), \sin(\varphi), \cos(\varphi) * \cos(\theta))$ ;





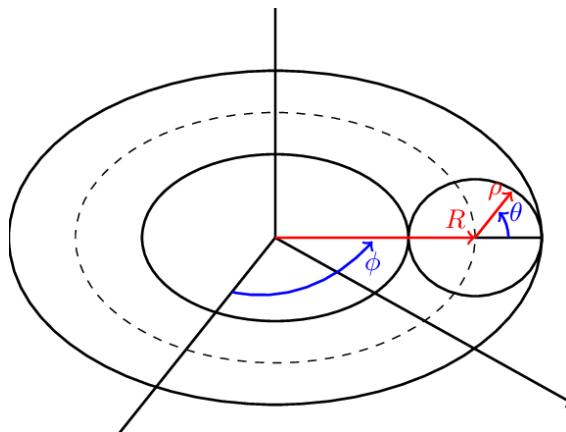
## Cone

Para a base do cone, que se encontra no plano XZ, o vetor normal é o mesmo para todos os pontos e é igual a  $(0, -1, 0)$ . Por outro lado, o vetor normal para qualquer ponto da superfície lateral do cone pode ser obtido da seguinte forma:  $(\sin(\alpha), \cos(\arctan(h/R)), \cos(\alpha))$ ;



## Torus

No caso do *Torus*, o vetor normal para qualquer ponto, pode ser obtido da seguinte forma:  $(\cos(\varphi) * \cos(\theta), \cos(\varphi) * \sin(\theta), \sin(\varphi))$ ;



## Bezier Patches

No caso dos *patches* de *Bezier* as normais podem ser calculadas através o produto externo entre as derivadas de  $u$  (segunda fórmula) e  $v$  (terceira fórmula) e normalizando o resultado obtido. A matriz  $M$  corresponde à matriz de *Bezier*, e  $P$  aos pontos de controlos.

$$p(u, v) = [u^3 \quad u^2 \quad u \quad 1] M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

$$\frac{\partial p(u, v)}{\partial u} = [3u^2 \quad 2u \quad 1 \quad 0] M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T V^T$$

$$\frac{\partial p(u, v)}{\partial v} = U M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} 3v^2 \\ 2v \\ 1 \\ 0 \end{bmatrix}$$

## Coordenadas de Textura

Nesta fase, cada primitiva passou a ter também um vetor no qual são armazenadas as coordenadas de textura que lhe poderão ser aplicadas.

### Plano

No plano, o cálculo das coordenadas de textura é bastante simples. Para tal basta considerar o seguinte:

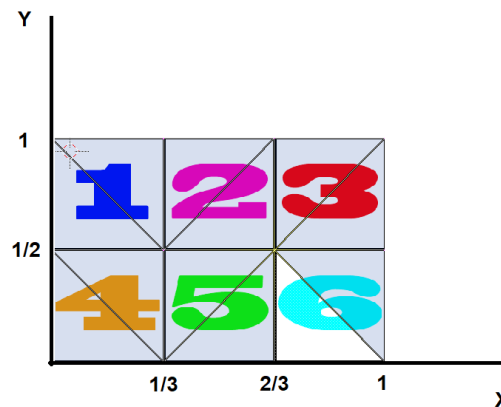
- O canto superior esquerdo corresponde ao ponto de textura com as coordenadas (0,1);
- O canto superior direito corresponde ao ponto de textura com as coordenadas (1,1);
- O canto inferior esquerdo corresponde ao ponto de textura com as coordenadas (0,0);
- O canto inferior direito corresponde ao ponto de textura com as coordenadas (1,0);

### Box

Para o cálculo dos pontos de textura na caixa, inicialmente foi preciso considerar os valores de proporções apresentados na figura a baixo, sendo que cada número corresponde à parte da imagem que ficará numa determinada face do cubo. No entanto, é necessário ter em conta que a *box* é desenhada em camadas. Assim temos o seguinte:

- Uma divisão é a “interceção” entre uma camada vertical e horizontal;

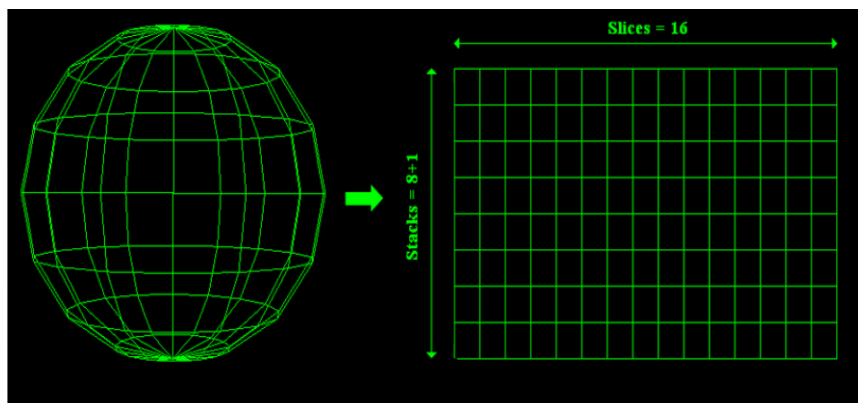
- $textH = (1/3)/(num\_camadas)$ , que simboliza o “comprimento” de uma divisão numa face;
- $textV = (1/2)/(num\_camadas)$ , que simboliza a “altura” de uma divisão numa face;
- À medida que calculamos os pontos de uma face calculamos também as coordenadas de textura (três por triângulo);
- A cada iteração pelas camadas horizontais o x aumenta  $textH$  e a cada iteração pelas camadas verticais o y aumenta  $textV$ ;
- Na face 4, o iniciamos o cálculo das coordenadas de textura no ponto (0,0);
- Na face 5, o iniciamos o cálculo das coordenadas de textura no ponto (1/3,0);
- Na face 6, o iniciamos o cálculo das coordenadas de textura no ponto (2/3,0);
- Na face 1, o iniciamos o cálculo das coordenadas de textura no ponto (0,1/2);
- Na face 2, o iniciamos o cálculo das coordenadas de textura no ponto (1/3,1/2);
- Na face 3, o iniciamos o cálculo das coordenadas de textura no ponto (2/3,1/2);
- Os valores das coordenadas de textura variam entre 0 e 1 (inclusive);



## Esfera

No caso da esfera, foi necessário dividir a imagem 2D, pelo número *stacks* e *slices* que a primitiva possui, considerando que o valor máximo da sua largura e comprimento é 1. Assim sendo temos o seguinte:

- Uma divisão da esfera é a “interseção” entre uma *stack* e uma *slice*;
- Para cada divisão da esfera, calculamos 6 coordenadas de textura (3 por triângulo);
- O comprimento de uma divisão da esfera é  $textH = 1/slices$ ;
- A largura de uma divisão da esfera é  $textV = 1/stacks$ ;
- A cada iteração pelas *stacks* a coordenada y varia  $textV$ ;
- A cada iteração pelas *slices* a coordenada x varia  $textH$ ;

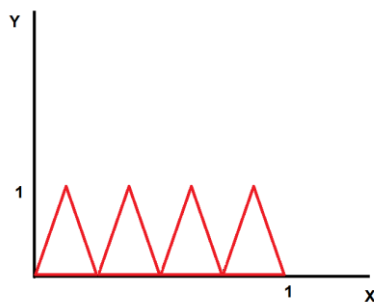


## Cone

No caso do cone, o cálculo das coordenadas de texturas é dividido em duas partes: uma dedicada à base da primitiva e outra à lateral;

Para base do cone, podemos considerar que a “decomposição” de uma circunferência em vários triângulos é aproximada da forma como é mostrado na figura 1. É de realçar que o comprimento da base de cada um dos triângulos é igual ao perímetro da circunferência a dividir pelo número de *slices*, ou seja,  $dist = 2 * PI * Raio / slices$ .

Assim sendo, basta para cada iteração pelas *slices*, determinar as coordenadas de textura, que são:  $(i * dist, 0, 0)$ ,  $((i+1) * dist, 0, 0)$  e  $(dist/2, 1, 0)$ , onde  $i$  é o número da *slice* em questão.



Por outro lado, o raciocínio utilizado para o cálculo das coordenadas de textura, é semelhante ao da esfera.

## Torus

Para o caso desta primitiva, o raciocínio é semelhante ao da esfera.

## Bezier Patches

Para o caso dos *patches* de *Bezier*, as coordenadas dos pontos de textura são dadas por  $(1-v, 1-u)$ , onde  $v$  e  $u$  são valores de tecelagem.

## Geração do Ficheiro do tipo 3D – Atualização

A adição de vetores normais e pontos de textura e a alteração para estruturas *VBO's* com índices nesta fase do projeto implicou a alteração da estrutura dos ficheiros *.3d* criados pelo gerador.

Nesta fase, cada ficheiro gerado possui uma estrutura bem definida onde a primeira linha representa todos os índices dos pontos a desenhar separados por vírgula. De seguida temos uma linha constituída apenas pelo número de vértices (N) sendo que para cada uma das N linhas seguintes temos os três valores das coordenadas (x, y e z) de cada ponto. De modo análogo, estão listados também tanto os vetores normais de cada ponto como os respetivos pontos de textura de cada figura sendo que para os últimos cada linha possui apenas dois valores (x e y).

```
1 0,1,2,2,3,0,4,5,6,6,7,4,
2 8
3 1 0 1
4 1 0 -1
5 -1 0 -1
6 -1 0 1
7 -1 0 -1
8 1 0 -1
9 1 0 1
10 -1 0 1
11 8
12 0 1 0
13 0 1 0
14 0 1 0
15 0 1 0
16 0 -1 0
17 0 -1 0
18 0 -1 0
19 0 -1 0
20 8
21 1 0
22 1 1
23 0 1
24 0 0
25 1 1
26 0 1
27 0 0
28 1 0
29
```

Figura 1 - Ficheiro 3d de um plano

## Demos

---

### Sistema Solar

-add cintura + planetas e luas a rodar sob o próprio eixo + xml

### Plano

```

<scene>
  <lights>
    <light type="POINT" posX="0" posY="1" posZ="1" />
  </lights>
  <group>
    <models>
      <model file="plane.3d" texture="de.png"/>
    </models>
  </group>
</scene>

```

Figura 2 - demo1.xml



Figura 3 - Output demo2

## Esfera

```

<scene>
  <lights>
    <light type="POINT" posX="0" posY="10" posZ="10" />
  </lights>
  <group>
    <rotate angle="-90" axisX="1" axisY="0" axisZ="0"/>
    <models>
      <model file="sphere.3d" diffR="1" diffG="0.8" diffB="0" ambR="0.7" ambG="0.8" ambB="0"/>
    </models>
  </group>
</scene>

```

Figura 4 - demo3.xml

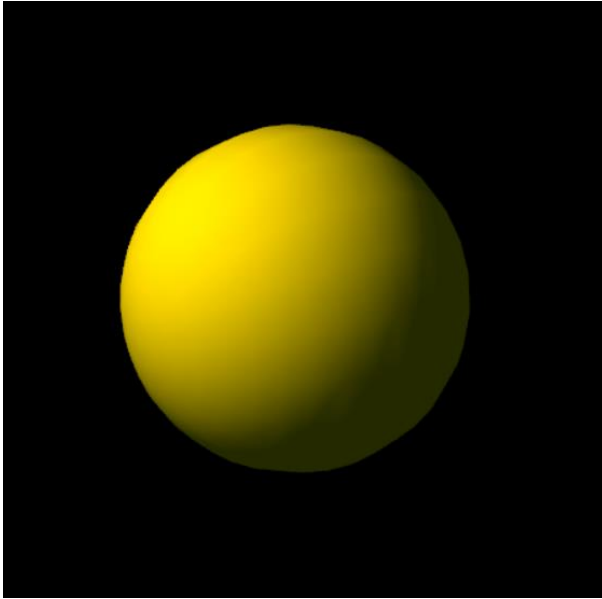


Figura 5 - Output demo3

## Torus

```
<scene>
  <lights>
    <light type="DIRECTIONAL" dirX="0" dirY="10" dirZ="10" />
  </lights>
  <group>
    <rotate angle="-90" axisX="1" axisY="0" axisZ="0"/>
    <models>
      <model file="torusDemo4.3d" texture="demo4Img.png"/>
    </models>
  </group>
</scene>
```

Figura 6 - demo4.xml



Figura 7 - Output demo4

## Cone

```
<scene>
  <lights>
    <light type="POINT" posX="0" posY="0" posZ="10" />
  </lights>
  <group>
    <rotate angle="-60" axisX="1" axisY="0" axisZ="1"/>
    <models>
      <model file="cone.3d" texture="demo5Img.png"/>
    </models>
  </group>
</scene>
```

Figura 8 - demo5.xml

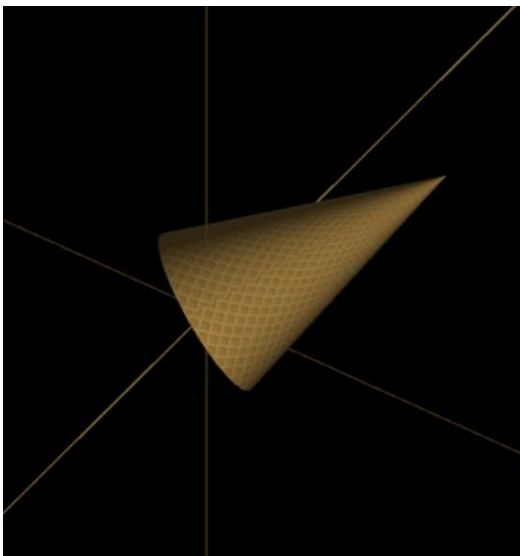


Figura 9 - Output demo5



## Caixa

```
<scene>
  <lights>
    <light type="SPOT" posX="0" posY="0" posZ="10" dirX="0" dirY="0" dirZ="-1"/>
  </lights>
  <group>
    <rotate angle="-20" axisX="0" axisY="1" axisZ="0"/>
    <models>
      <model file="box.3d" texture="img5.png"/>
    </models>
  </group>
</scene>
```

Figura 10 - demo6.xml

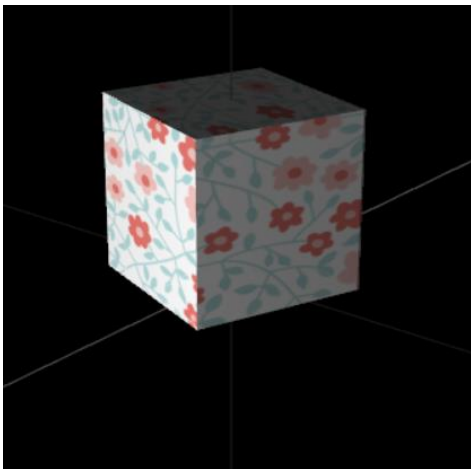


Figura 11 - Output demo6

## Teapot

```
<scene>
  <lights>
    <light type="POINT" posX="0" posY="10" posZ="10" />
  </lights>
  <group>
    <rotate angle="-90" axisX="1" axisY="0" axisZ="0"/>
    <models>
      <model file="teapot.3d" texture="teapot.jpg" ambR="1" ambG="1" ambB="1"/>
    </models>
  </group>
</scene>
```

Figura 12 - demo1.xml



Figura 13 - Output demo1

## Conclusão

---

Ao longo desta quarta e última fase do trabalho prático, foi possível aplicar e consolidar todo o conhecimento adquirido nas aulas práticas e teóricas da unidade curricular de Computação Gráfica.

O grupo considera que realizou tanto esta fase como o projeto na totalidade com sucesso, uma vez que apresentamos um sistema solar dinâmico e realista, com todas as funcionalidades pedidas no enunciado.

No entanto, há sempre espaço para melhorias, sendo que como trabalhos futuros poder-se-ia implementar a *View Frustum Culling*.

