Universidade do Minho

Mestrado Integrado em Engenharia Informática

Computação Gráfica

4ª Fase

Grupo 17

Daniela Fernandes	José Gomes	Ricardo Costa	Tiago Rodrigues
A73768	A82418	A85851	A84276

30 de maio de 2021

Conteúdo

1 Introdução

O trabalho prático da unidade curricular de Computação Gráfica tem como base a utilização do OpenGl, recorrendo à biblioteca GLUT, para a construção de modelos 3D.

A produção dos modelos envolve as diferentes temáticas abordadas nas aulas, como as transformações geométricas, curvas, superfícies cúbicas, iluminação e texturas. Com a realização deste projeto, dividido em quatro fases, tem como objetivo consolidar todos estes tópicos.

Este documento diz respeito à realização da quarta e última fase do trabalho prático da unidade curricular. Nesta fase a principal objetivo é a aplicação das Normais e Coordenadas de Textura.

Ao longo do relatório vamos explicar as decisões que tomamos.

2 Descrição do Probema

Nesta fase do projeto vamos ter que adicionar novas funcionalidades ao Gerador e Motor desenvolvidos anteriormente. O Gerador vai ter que ser capaz de calcular as normais e as coordenadas de textura das primitivas que definimos. Já o Motor vai que utilizar os parâmetros que vamos definir para aplicação de iluminação e texturas.

Os requisitos desta fase são:

- **Gerador:** os parâmetros que recebe são o tipo da primitiva gráfica, parâmetros relativos ao modelo e nome do ficheiro onde vão ser guardados os vértices.
  - calcula o vetores **normais** e as **coordenadas de textura** de cada uma das primitivas.
- **Motor:** vai ler o ficheiro *XML* que contém todos os modelos das primitivas, vai ler do ficheiro *XML* as características de iluminação e de textura e aplicar as **texturas** e **iluminação** ao modelo.

3 Resolução do Problema

Para a resolução desta fase vamos alterar o **gerador** e o **motor** que criamos para as fases anteriores.

3.1 Gerador

Para desenvolvermos os objetivos estabelecidos para esta fase, foi necessário modificar o **gerador** desenvolvido para as fases anteriores.

Deste modo, desenvolvemos o cálculo das normais e das coordenadas de textura para cada primitiva e escrevê-las no ficheiro.

3.1.1 Plane (Plano)

3.1.1.1 Normais

São utilizadas dois vetores normais, um para a face de cima e outro para a face de baixo.

Como tal, para qualquer ponto da face de cima o vetor normal vai estar no sentido positivo do eixo dos y (0,1,0). Já para os pontos da face de baixo o vetor vai apontar para o sentido negativo dos y (0,-1,0).

3.1.1.2 Coordenadas de Textura

As coordenadas de textura que definimos para o plano são (0,0), (1,1), (1,0) e (0,1) para cada ponto do plano tanto na face de cima como na de baixo. Isto porque as duas coordenadas de uma imagem (*u* e *v)* estão definidas entre 0 e 1.

3.1.2 Caixa (Box)

3.1.2.1 Normais

Cada uma das fases da caixa vai ter um vetor normal diferente das outras mas igual para todos os pontos da mesma.

Deste modo, para a face da frente a normal vai apontar no sentido positivo do eixo do z (0,0,1), a face de trás no sentido negativo do eixo do z (0,0,-1).

Na face de cima a normal vai estar a apontar no sentido positivo dos y (0,1,0) e a face de baixo vai estar a apontar no sentido negativo dos y (0,-1,0).

Por fim, a face da direita vai ter a normal a apontar no sentido positivo dos x (1,0,0), enquanto a da esquerda vai estar a apontar no sentido negativo (-1,0,0).

3.1.2.2 Coordenadas de Textura

Relativamente às coordenadas de textura, nesta fase decidimos simplificar o processo retirando a implementação das divisões e efetuando apenas a geração das coordenadas de textura para uma *box* com 0 divisões, ou seja, com apenas 2 triângulos por face. Isto permite facilitar este processo e gerar as coordenadas de um modo parecido com o que foi feito para o plano, mas para cada uma das faces da *box*. Isto foi feito para uma imagem como a do género:

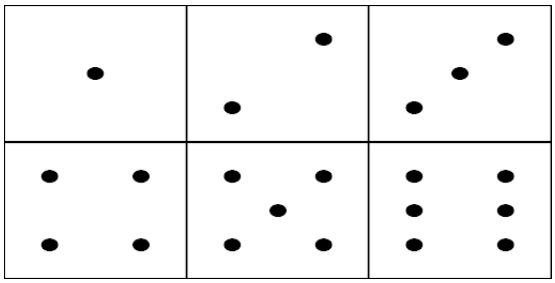

Figura 1: Imagem usada para a textura da *box*

Para realizar o processo, dividimos a imagem na horizontal por 3 e na vertical por 2. Com estes valores conseguimos associar cada face do cubo na imagem para os pontos respetivos da nossa primitiva. Podemos então somar 1/2 ao valor da coordenada de textura v (que corresponde à orientação vertical) para obter as faces de cima da imagem ou 1/3 e 2/3 ao valor da coordenada de textura u (que corresponde à direção horizontal) para obter as faces do centro ou da direita da imagem. Com a conjunção destes 2 valores conseguimos obter coordenadas para qualquer ponto da imagem.

3.1.3 Esfera (Sphere)

3.1.3.1 Normais

Para cada ponto da esfera vai ser definido o seu vetor normal, este vetor é perpendicular à superfície.

Vai ter a mesma direção que o ponto, como tal vamos ter em conta a fórmula do cálculo de um ponto e a única diferença que vai haver é que não vai haver a multiplicação pelo o valor do raio. Vamos considerar que o vetor tem módulo 1. Assim, as fórmulas são as seguintes:

$$normal\_x = cos(\beta) * sin(\alpha)$$
$$normal\_y = sin(\beta)$$
$$normal\_z = cos(\beta) * cos(\alpha)$$

3.1.3.2 Coordenadas de Textura

Para calcular as coordenadas de textura de uma esfera vamos pensar primeiro em dividir a imagem em *slices* e *stacks*, na horizontal e na vertical, respetivamente. Como depois já sabemos que as coordenadas estão entre 0 e 1, podemos dividir o valor da *slice* e *stack* onde estámos na respetiva iteração pelo número de *slices* e *stacks* total, respetivamente. Isto permite-nos obter pontos na nossa imagem que servir como textura para cada ponto respetivo que geramos para a primitiva esfera.

Em cada iteração vamos na mesma gerar 4 pontos entre 0 e 1 correspondentes às coordenadas de textura, já que, como vimos em fases anteriores, em cada iteração geramos 4 pontos para a primitiva, que irão servir para formar 2 triângulos.

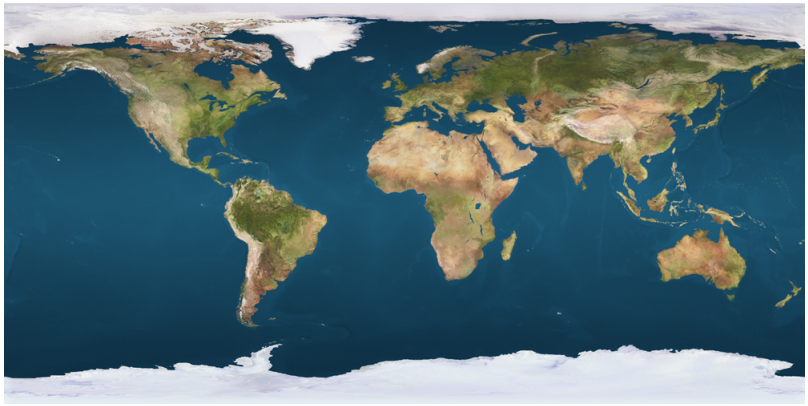

Figura 2: Imagem usada para a textura de uma das esferas, que irá ser dividida em *slices* e *stacks* como explicado

3.1.4 Cone

3.1.4.1 Normais

Relativamente à base do cone, a geração das normais é fácil: apenas é necessário para cada um dos pontos gerar um vetor normal igual a (0, -1, 0), que aponta para o sentido negativo do eixo do y.

Para os pontos do topo do cone os vetores normais são gerados através das seguintes fórmulas:

$$normal\_x = r * sin(\alpha)$$
$$normal\_y = sin(tan(radius/height))$$
$$normal\_z = r * cos(\alpha)$$

Nestas fórmulas, α corresponde à variação do ângulo na horizontal (que é atualizado para cada uma das iterações das *slices*), r corresponde ao valor do raio na altura (que depedendo

de qual dos 4 pontos numa dada iteração estámos a considerar, usámos r ou new_r, sendo este último usado para os pontos de cima) e *height* à altura do cone.

3.1.4.2 Coordenadas de Textura

Para gerar as coordenadas de textura decidimos considerar uma imagem circular e iremos usar a mesma imagem tanto para a base como para o topo do cone.

Figura 3: Imagem usada para a textura de um cone, usada para a base e para o topo do mesmo

Relativamente à base do cone, consideramos o raio como 0.5, já que as coordenadas estão sempre entre 0 e 1 e somamos 0.5 na fórmula, já que a circunferência está centrada no ponto (0.5, 0.5) da imagem. Depois é usado o cosseno e o seno, que correspondem à horizontal e vertical, respetivamente. Assim, as fórmulas são as seguintes:

$$tex\_u = cos(\alpha) * 0.5 + 0.5$$
$$tex\_v = sin(\alpha) * 0.5 + 0.5$$

Relativamente ao topo do cone, j corresponde ao número da *stack* que estámos a considerar e *stacks* ao número total de *stacks* existentes:

$$tex\_u = (j/stacks) * cos(\alpha) + 0.5$$
$$tex\_v = (j/stacks) * sin(\alpha) + 0.5$$

3.1.5 Anel (Ring)

3.1.5.1 Normais

Tal como utlizamos no plano, no anel apenas vamos utilizar duas normais, uma para a face de cima e outra para a face debaixo.

Para cada um dos pontos da face de cima, a normal vai apontar no sentido positivo do eixo dos y (0,1,0). Já para um ponto da face debaixo, a normal vai apontar no sentido negativo do mesmo eixo (0,-1,0).

3.1.5.2 Coordenadas de Textura

Iremos utilizar a imagem para a textura do anel um número de vezes igual ao número de *slices*. Isto porque iremos associar a imagem a cada par de triângulos que corresponde a uma *slice*.

Basicamente para os 4 pontos que foram uma *slice* do anel as coordenadas de textura serão as seguintes:

$$P1 = (0, 0)$$
$$P2 = (1, 0)$$
$$P3 = (1, 1)$$
$$P4 = (0, 1)$$

A imagem que irá ser utilizada para cada *slice* é a seguinte:

Figura 4: Imagem usada para cada uma das *slices* de um anel (usada na vertical)

Para a parte de baixo do anel é apenas necessário trocar as coordenadas dos 2 pontos interiores um pelo outro e fazer a mesma coisa para os pontos exteriores para a textura ficar igual em cima e em baixo.

3.1.6 *Bezier*

3.1.6.1 Normais

Para calcular as normais do *bezier* é necessário aplicar as seguintes fórmulas:

- $u = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix}$
- $u' = \begin{pmatrix} 3 * u^2 & 2 * u & 1 & 0 \end{pmatrix}$
- $v = \begin{pmatrix} v^3 \\ v^2 \\ v \\ 1 \end{pmatrix}$
- $v = \begin{pmatrix} 3 * v^2 \\ 2 * v \\ 1 \\ 0 \end{pmatrix}$

- $M = M^T = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$
- P corresponde aos pontos de controlo de uma dada coordenada (x, y ou z) - matriz 4x4
- $der\_U = u' * M * P * M^T * v^T$
- $der\_V = u * M * P * M^T * v'$

Algumas das variáveis já conhecemos da fase anterior. Obtidas as derivadas dos vetores u e v, falta apenas calcular o seu produto externo e normalizar para obtermos finalmente as normais.

3.1.6.2 Coordenadas de Textura

Para calcular as coordenadas de textura temos de pensar apenas que pensar que para cada ponto calculado na fase anterior para gerar a primitiva *teapot* temos duas variáveis: *u_step* e *v_step*. Também sabemos que o conjunto destas variáveis forma uma grelha e então, as coordenadas de textura u e v são simplesmente *v_step* e *u_step*, respetivamente. Isto significa que a imagem que irá ser usada como textura é como se fosse dividida a partir desta grelha tendo em conta os valores de *u_step* e *v_step* e podemos simplesmente usá-los para as coordenadas de textura.

3.2 Motor

Como no **motor** vai haver uma aplicação de texturas e iluminação, houve uma alteração e criação de novas classes para esse efeito.

3.2.1 Estruturas Definidas

3.2.1.1 Light

Esta estrutura foi criada por causa da **iluminação**, vai haver uma armazenamento das diferentes características da luz.

```
struct Light{
    int type;
    float *pos, *dir;
    float cutoff, linear, constant, quadratic, exponent;
};
```

3.2.1.2 Prim

Para esta fase foi criada uma nova estrutura para armazenar os pontos presentes nos ficheiros ".3d" e o respetivo VBO. Além destes parâmetros para esta fase foi também necessário criar 2 novos vetores e respetivos VBO's para armazenar os valores correspondentes às normais e às coordenadas de textura.

Foi necessário fazer isto numa estrutura nova e não na estrutura *Model* como faziamos na fase anterior, porque nesta fase já não podemos usar o mesmo VB0 para todos os modelos de um grupo como faziamos antes. Isto não é possível porque agora cada modelo tem também parâmetros diferentes entre si como vamos ver no tópico a seguir. Assim, criamos uma nova estrutura e agora cada VBO possui apenas os pontos referentes a um dado modelo e não a todos os modelos de um dado grupo.

```
struct Prim{
    vector<float> vertexB;
    vector<float> normalB;
    vector<float> text_coordsB;

    GLuint coords[1];
    GLuint normals[1];
    GLuint texCoord[1];

    GLuint num_vertices;
};
```

3.2.1.3 Model

Nesta fase, a estrutura Model está relacionada com um modelo existente no XML, e vai armazenar as características de iluminação e textura a ele associadas. Como podemos ver agora o *Model* possui novos parâmetros e estes podem variar de modelo para modelo dentro do mesmo grupo. Daí, como explicado em cima, já não podemos usar o mesmo VBO para armazenar os pontos de todos as primitivas de um dado grupo, pois na fase de desenho é necessário desenhá-los em separado para podermos aplicar os valores destas variáveis como iremos ver mais à frente.

```
struct Model{
    string file;
    float *amb, *diff, *spec, *emiss;
    float shininess;

    int id_textura;
};
```

3.2.2 *Parsing* do XML

Para armazenar as diferentes fontes de iluminação e os seus parâmetros que poderão existir foi criado um vetor global $vector < Light > lightsV$.

Relativamente aos diferentes parâmetros do *Model* presentes nesta fase estes foram também armazenados nas respetivas variáveis. De notar que caso não sejam especificados algum destes parâmetros no ficheiro de configuração ".xml" serão usados valores padrão especificados no *glMaterial*.

Relativamente à nova estrutura *Prim*, decidimos nesta fase criar um *map* global do seguinte modo: $map < string, Prim > primitives$, em que a chave é o nome do modelo. Ao usarmos um *map*, é possível não estar a carregar modelos que já tenham sido carregados anteriormente. Para isso, é só verificarmos se um dado modelo já existe no nosso *map* através do seu nome. Deste modo, apenas carregamos cada modelo e armazenamos os seus pontos uma única vez e tornamos o programa mais eficiente.

Para armazenar as texturas decidiu-se usar o mesmo raciocínio e criou-se um *map* global também: $map < string, int > textures\_map$ que associa um nome de um modelo a um identificador de textura dado pela função *loadTexture*. Assim, podemos aqui também verificar se a textura atribuída a um dado modelo já foi ou não carregada para memória. Se já tiver sido, é apenas associado o identificador de textura respetivo, que fica guardado na variável *id_textura* da estrutura *Model*. Se o modelo não possuir textura associada no ficheiro de configuração, ao seu identificador de textura é atribuído o valor 0. Este processo foi realizado da seguinte forma no *parsing* dos modelos:

```
texture = pListModels->Attribute("texture");

if (texture != nullptr){
    if (textures_map.find(texture) == textures_map.end()){
        m.id_textura = loadTexture(texture);
        textures_map.insert( pair<string, int> (texture, m.id_textura));
    }
    else {
        m.id_textura = textures_map.find(texture)->second;
    }
}
else {
    m.id_textura = 0;
}
```

3.2.3 Iluminação

Relativamente às fontes de iluminação permitiu-se o uso de diferentes tipos de fonte como veremos a seguir. Para aplicar cada uma começou-se por definir o que é comum a todas, as componentes *ambient* e *diffuse*.

```
float diff[4] = { 1.0f, 1.0f, 1.0f, 1.0f };
float spec[4] = { 1.0f, 1.0f, 1.0f, 1.0f };

glLightfv(i + GL_LIGHT0, GL_DIFFUSE, diff);
glLightfv(i + GL_LIGHT0, GL_SPECULAR, spec);
```

Agora, dependendo do tipo de luz irão ser definidos parâmetros diferentes:

- ***POINT*** : basta definir a posição da fonte.

  ```
  if (l.type == 0) {
      glLightfv(i + GL_LIGHT0, GL_POSITION, l.pos);
  }
  ```

  Não esquecer que neste caso o último parâmetro de *pos* deverá ser 1 visto tratar-se de um ponto.

- ***DIRECTIONAL*** : neste caso define-se a direção segundo *pos* mas em que o quarto parâmetro é 0, visto tratar-se de um vetor. Além disto, também se definem as componentes *linear*, *constant*, *quadratic*.

```
else if (l.type == 1) {

    // Quarto parâmetro a 0 já que neste caso é um vetor
    l.pos[3] = 0;

    glLightfv(i + GL_LIGHT0, GL_POSITION, l.pos);

    glLightf(i + GL_LIGHT0, GL_LINEAR_ATTENUATION, l.linear);
    glLightf(i + GL_LIGHT0, GL_CONSTANT_ATTENUATION, l.constant);
    glLightf(i + GL_LIGHT0, GL_QUADRATIC_ATTENUATION, l.quadratic);
}
```

- ***SPOT*** : neste caso, além de todas as componentes definidas para o tipo de luz *directional*, são definidas também as componentes *cutoff*, *exponent* e a direção da luz.

```
else {
    glLightfv(i + GL_LIGHT0, GL_POSITION, l.pos);

    glLightf(i + GL_LIGHT0, GL_LINEAR_ATTENUATION, l.linear);
    glLightf(i + GL_LIGHT0, GL_CONSTANT_ATTENUATION, l.constant);
    glLightf(i + GL_LIGHT0, GL_QUADRATIC_ATTENUATION, l.quadratic);

    glLightfv(i + GL_LIGHT0, GL_SPOT_DIRECTION, l.dir);
    glLightf(i + GL_LIGHT0, GL_SPOT_CUTOFF, l.cutoff);
    glLightf(i + GL_LIGHT0, GL_SPOT_EXPONENT, l.exponent);
}
```

Para tornar possível o uso de várias luzes se quiséssemos, é necessário somar a *GL_LIGHT0* o número de luzes que já existem (i) em cena.

Relativamente à parte dos modelos, ou seja, as características de iluminação de cada um, depois de armazenados os valores como já explicamos, é apenas necessário aplicá-los antes de desenhar a primitiva respetiva. Para isso é usado o seguinte:

```
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, model.diff);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, model.spec);
glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, model.emiss);
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, model.amb);
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, model.shininess);
```

Além disto, como para as trajetórias dos planetas e luas e também os eixos do referencial não existem normais, decidimos desativar a luz para eles antes de desenhá-los, do seguinte modo:

```
glDisable(GL_LIGHTING);

...

glEnable(GL_LIGHTING);
```

3.2.4 Textura

Após associar a textura a cada modelo e ter dado o *load* das texturas que ainda não tivessem sido através da função *loadTexture*, é apenas necessário aplicar estas texturas no momento em que as primitivas são desenhadas. Não esquecer que depois de aplicar a textura e se ter desenhado a primitiva respetiva é preciso voltar ao estado inicial, ou seja, sem nenhuma textura (0).

```
glBindTexture(GL_TEXTURE_2D, model.id_textura);

....

glBindTexture(GL_TEXTURE_2D, 0);
```

3.3 Modelo do Sistema Solar com Iluminação e Texturas

Para fazer uso do que foi implementado foi então definida uma fonte de luz do tipo *POINT* no ficheiro de configuração posicionada no ponto (0, 0, 0). Depois na parte dos modelos foi só definir as texturas no ficheiro de configuração e o resto dos parâmetros de cada modelo é usado os valores padrão. Isto exceto no modelo do Sol, em que são definidos os parâmetros de emissão todos a 1, para parecer que é o Sol que emite a luz.

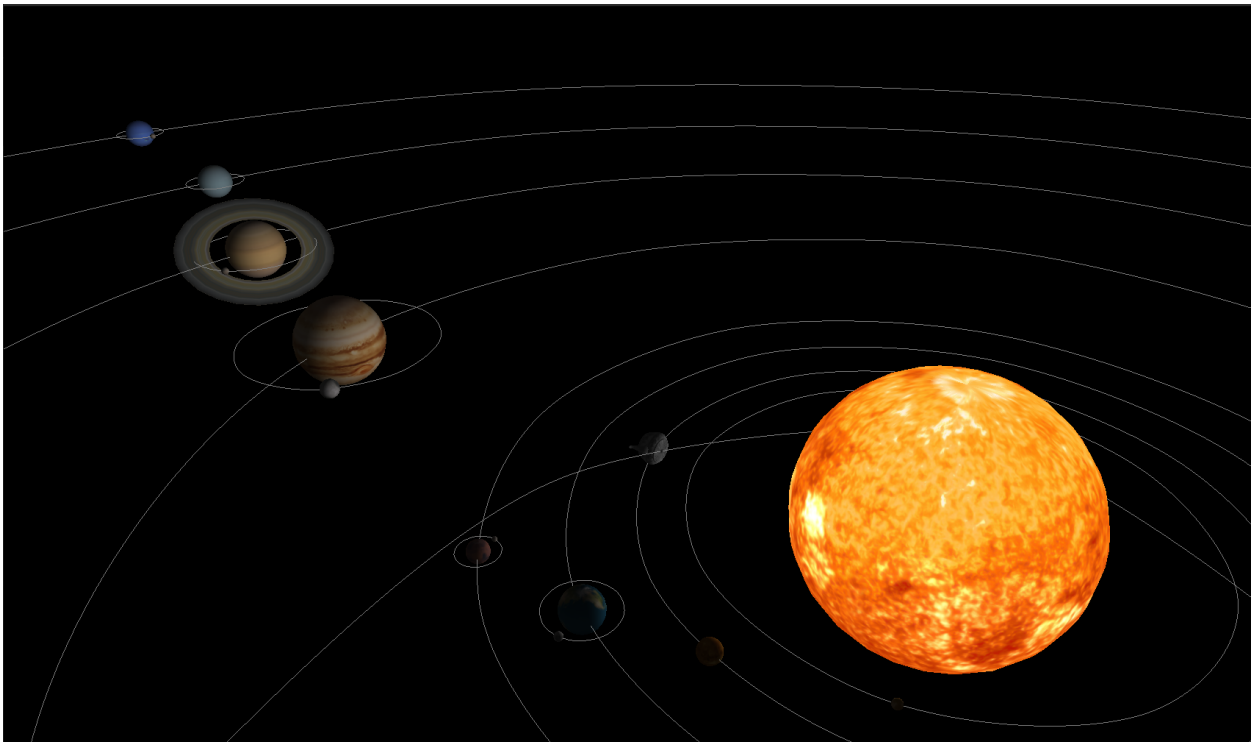

Figura 5: Modelo do Sistema Solar dinâmico com iluminação e texturas

4 Conclusão

De uma perspetiva geral, consideramos que a realização desta fase foi relativamente bem sucedida, visto que pensamos cumprir com todos os requisitos estabelecidos no enunciado.

Conseguimos implemantar bem a iluminação, apesar de no modelo final apenas termos utilizado o tipo de luz *POINT* e também as texturas de todas as primitivas, exceto a do cone, que apesar de na base a textura se encontrar correta, no topo a textura não está exatamente bem aplicada.

Criámos também um outro ficheiro de configuração, para demonstrar as normais e coordenadas de texturas das primitivas que não utilizámos no modelo do sistema solar, como o cubo e o plano.

Poderíamos melhorar este projeto implementando também outras funcionalidades extra, como um menu ou permitir movimentar a câmara com o rato, como vimos em alguns dos guiões realizados.

Em suma, após terminar a realização deste projeto, concluímos que as matérias lecionadas foram devidamente consumadas e aprofundadas com a concretização dos diferentes problemas que nos foram sendo propostos.