

## Trabalho Prático - fase 4

### Texturas e Iluminação



Catarina Cruz  
A84011



Jorge Mota  
A85272



Maria Silva  
A83840



Mariana Marques  
A85171

06 de Março de 2020



Universidade do Minho

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Generator</b>	<b>4</b>
2.1	Vetores Normais e Coordenadas Textura . . . . .	4
2.1.1	Plano . . . . .	4
2.1.2	Box . . . . .	6
2.1.3	Esfera . . . . .	7
2.1.4	Cone . . . . .	8
2.1.5	Cilindro . . . . .	10
2.1.6	Torus . . . . .	12
2.1.7	<i>Bezier Patches</i> . . . . .	12
<b>3</b>	<b>Engine</b>	<b>14</b>
3.1	Material . . . . .	14
3.2	XML Parser . . . . .	14
3.3	Skybox . . . . .	15
3.4	Movimentação da câmera . . . . .	16
3.5	MipMapping . . . . .	17
<b>4</b>	<b>Lighting</b>	<b>17</b>
4.1	Light Types . . . . .	17
4.1.1	<i>Point Light</i> . . . . .	17
4.1.2	<i>Directional Light</i> . . . . .	18
4.1.3	<i>Spotlight</i> . . . . .	18
<b>5</b>	<b>Optimizações</b>	<b>18</b>
5.1	<i>View Frustum Culling</i> . . . . .	18
5.2	Remodelação RGB Color . . . . .	19
5.3	Debug Info . . . . .	20
<b>6</b>	<b>Cenários</b>	<b>21</b>
6.1	Sistema Solar . . . . .	24
<b>7</b>	<b>Conclusão</b>	<b>25</b>

# 1 Introdução

Nesta quarta fase do projeto estendemos as capacidades do motor gráfico anteriormente definido com a adição de luzes e texturas. Foram também feitas algumas otimizações ao modelo até agora criado.

O gerador de modelos foi também estendido para calcular para cada uma das primitivas os vetores normais e as coordenadas de textura.

Por último, o modelo de teste utilizado (Sistema Solar) foi alterado para demonstrar estas capacidades, fazendo uso de texturas para os planetas e Sol, assim como colocar iluminação neste. Foram ainda criados modelos extras com texturas e iluminação para termos vários cenários de teste.

## 2 Generator

O output do *generator* foi alterado para incluir as normais e as coordenadas de textura correspondentes aos pontos dos triângulos calculados.

De seguida, é explicado o processo do cálculo das normais e das coordenadas de textura para cada uma das primitivas.

### 2.1 Vetores Normais e Coordenadas Textura

Para todas as **texturas** é usado um sistema de coordenadas idêntico ao usado no mapeamento do OpenGL. Ou seja, o canto inferior esquerdo é a origem do referencial e este incrementa no x para a direita e no y para cima.

Apresentamos, para todas as primitivas, um esquema das texturas e em todos eles, a parte representada a preto é a não visível e desprezada no mapeamento do objeto.

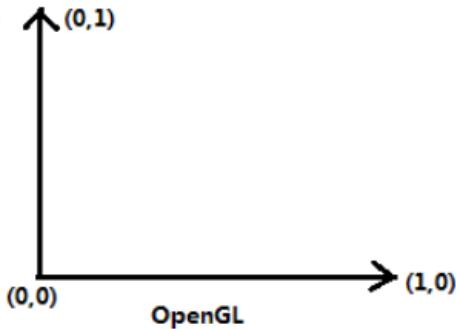


Figura 1: Referencial das coordenadas de texturas

#### 2.1.1 Plano



Figura 2: Plano com luz e textura

## Normais

Para calcular as normais de um plano, basta encontrar um **vetor unitário perpendicular** a este. Se o plano estiver situado no plano xOz, a normal será um vetor unitário com apenas com a componente y positiva (0,1,0).

## Coordenadas de Textura

As coordenadas de textura para o plano são calculadas de forma muito simples. Como o plano e a textura têm a mesma forma, o mapeamento é direto e apenas é necessário **mapear os cantos da textura para os cantos do plano**.

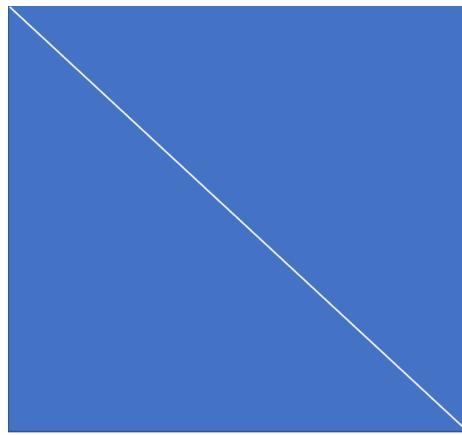


Figura 3: Esquema utilizado no mapeamento da textura do plano

### 2.1.2 Box



Figura 4: Box com luz e textura

#### Normais

As normais da *box* são calculadas para cada uma das faces. Como estas são paralelas a um plano, calcula-se de forma idêntica à do plano, aonde as normas de uma face são o **vetor unitário perpendicular ao plano** corresponde [ (1,0,0) ,(-1,0,0) ,(0,1,0) ,(0,-1,0) ,(0,0,1) ,(0,0,-1) ].

#### Coordenadas de Textura

Para as texturas definiu-se um esquema para mapear as coordenadas de cada face. Por isso, tal como representado na imagem a baixo e equivalente à **planificação de um cubo** são aplicadas contas para obter cada uma das face. Entre elas apenas é necessário aplicar um *shift* de  $\frac{1}{4} * \text{pos\_da\_face\_na\_textura}$  na direção horizontal pretendida ou um *shift* de  $\frac{1}{3} * \text{pos\_da\_face\_na\_textura}$  na vertical do segundo quadrado para obter as dimensão de uma nova face.

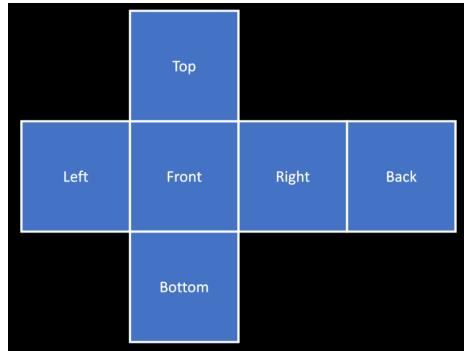


Figura 5: Esquema utilizado no mapeamento da textura de um cubo

Esta textura é usada não só para um cubo mas para qualquer paralelepípedo da mesma maneira.

### 2.1.3 Esfera



Figura 6: Esfera com luz e textura

#### Normais

Para calcular as normais da esfera, apenas é preciso **normalizar as coordenadas dos vértices** de forma a obter as normais. Como obtemos as coordenadas dos vértices por conversão de coordenadas esféricas então as normais são o mesmo mas com raio 1 (`spherical.to_cartesian(1,α,β)`).

#### Coordenadas de Textura

A textura usada para aplicar à esfera é retangular com triângulos na parte superior e inferior deste. Estes triângulos servem para criar um ambiente redondo e a parte triangular para cobrir o resto da esfera. A parte lateral da textura é dividida no mesmo número de slices e stacks que que a esfera e faz-se o mapeamento.

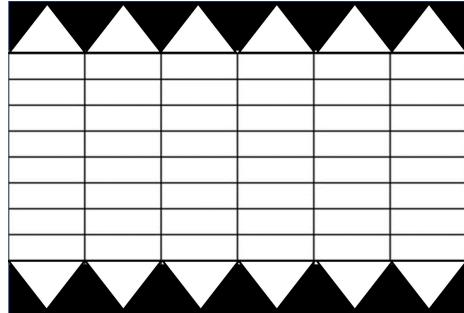


Figura 7: Esquema utilizado no mapeamento da textura de uma esfera

#### 2.1.4 Cone

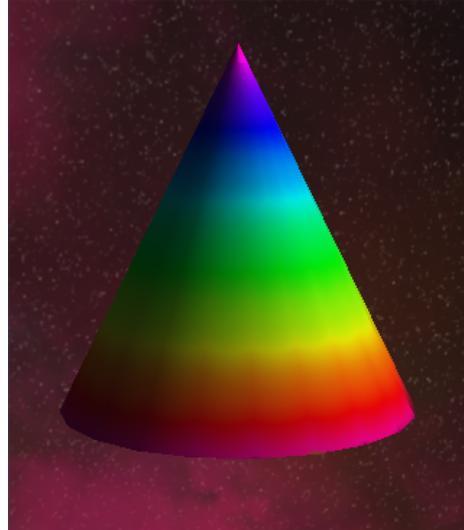


Figura 8: Cone com luz e textura

#### Normais

Para o cone é necessário calcular as **normais da base e do lado**. Para a base, tal como explicado anteriormente, basta calcular o **vetor unitário per-**

**pendicular ao plano** aonde a base de encontra, que costuma ser o  $(0, -1, 0)$ .

Para a parte lateral do cone, é mais complexo , visto que é preciso descobrir a componente y do ponto da lateral de um **vetor perpendicular à hipotenusa**. Para isso é necessário calcular o ângulo entre o ponto do lado do cone e a base do mesmo e determinar o y de imediato com a seguinte fórmula elaborada:  $\sin(\tan(\text{radius} \div \text{height}))$ .

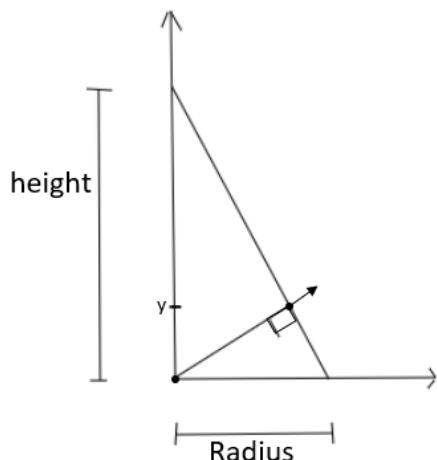


Figura 9:

Este valor indica o y que precisamos para obter o respetivo vetor normal e o ângulo define a inclinação das normais laterais e após calcular este ângulo basta descobrir as componentes 'x' e 'z' do vetor e normalizar.

#### Coordenadas de Textura

Para aplicar a textura a um cone foi definido o formato da figura abaixo. O círculo da esquerda é a base do cone e o da direita é a textura do lado do cone. A textura da parte lateral é dividida pelos *slices* e *stacks* que o cone tem e faz-se o mapeamento, idêntico à esfera.

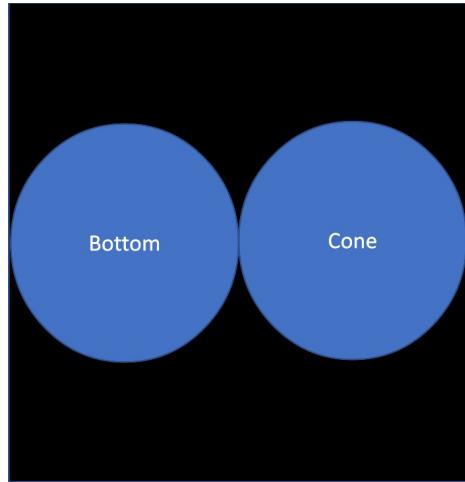


Figura 10: Esquema utilizado na textura de um cone

#### 2.1.5 Cilindro



Figura 11: Cilindro com luz e textura

## Normais

Como costume, para calcular as normais dos cilindros têm de se **calcular a das bases e para a lateral**. A normal das bases, tal como no cone, são **vetores unitários perpendiculares ao plano** onde estas se encontram, normalmente sendo estas a do topo e base respetivamente  $(0, 1, 0)$  e  $(0, -1, 0)$ . Para a parte lateral, é necessário calcular, da mesma forma que o cone já explicado anteriormente, o **ângulo entre o ponto lateral e a base do mesmo**. É ainda **normalizado** o vetor das coordenadas dos pontos.

## Coordenadas de Textura

Para as coordenadas de textura é utilizado o esquema da figura abaixo. Para os lados divide-se o retângulo pelo número de *slices* e sempre que se avança de *slice* soma-se essa quantidade ao *x* da textura. O *y* só toma dois valores 0 e 0.625. Por fim, o topo e a base são obtidas através de coordenadas polares, somando ao centro de cada uma  $\sin(\alpha) \times 0.1875$  para o *x* e  $\cos(\alpha) \times 0.1875$  para o *y*. O primeiro círculo corresponde ao topo e o segundo corresponde à base.

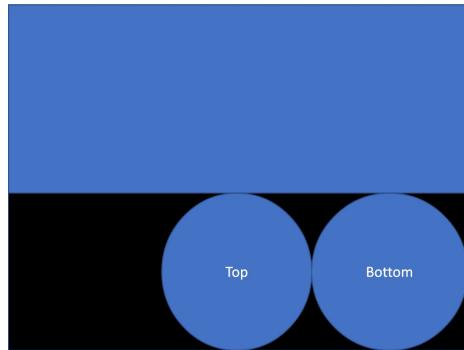


Figura 12: Esquema utilizado na textura de um cilindro

### 2.1.6 Torus

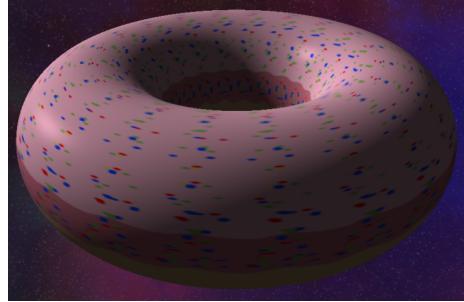


Figura 13: Tourus com luz e textura

#### Normais

Para as normais do torus é usado o mesmo raciocínio da esfera, usando como ponto de referência os **centros de cada anel**, não esquecendo de **normalizar os vetores**.

#### Coordenadas de Textura

Para as coordenadas de textura, o mapeamento também é similar ao da esfera. Tal como representado de seguida, é necessário ter em conta os *slices*, que permite percorrer cada anel do torus ao longo do seu perímetro e o *cyl\_slices* que permite desenhar os diferentes anéis do torus.

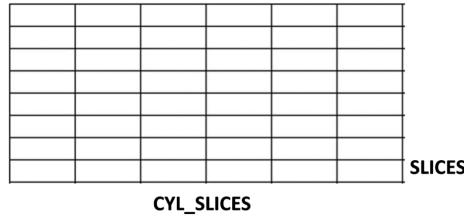


Figura 14: Esquema utilizado na textura de um torus

### 2.1.7 Bezier Patches

#### Normais

Para o cálculo do vetor normal de cada vértice é utilizada as fórmulas das derivadas parciais em U e V e calcula-se o vetor normal a partir do produto interno da derivada em U e V para um certo iter\_u e iter\_v:

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

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

Figura 15: Formulas de cálculo dos vetores tangentes

De seguida normaliza-se o resultado do produto interno destes dois para se tornar um vetor unitário normalize(  $V \times U$ ).

#### Coordenadas de Textura

Para as coordenadas de textura, o pensamento é semelhante às primitivas, principalmente o plano e o cubo. Considerando que cada bezier patch tem 16 pontos, é mapeada a textura de para cada patch diretamente. No fim, o modelo vai conter várias réplicas da aplicação da textura.

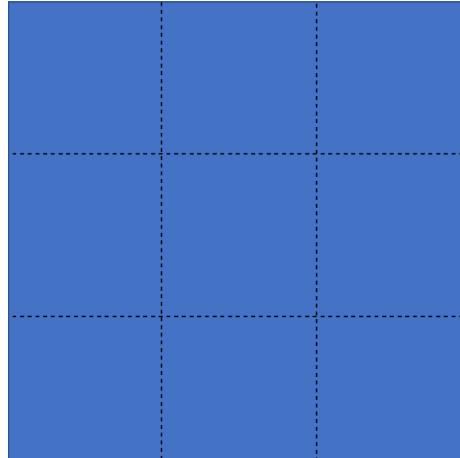


Figura 16: Esquema utilizado na textura de bezier patches

## 3 Engine

### 3.1 Material

O material descreve as propriedades de um objeto e com a utilização de iluminação este sobrepõem-se à utilização de cores com `glColor`.

Um material descreve-se da seguinte forma:

- difusa - adiciona diferentes tonalidade consoante a luz que incide no objeto
- ambiente - adiciona uma cor uniforme a todo o objeto
- especular - brilho intenso quando a luz incide diretamente num objeto
- emissiva - luz emitida por um objeto
- brilho - valor de refleção da luz especular associado.

De seguida, apresenta-se uma imagem ilustrativa dos diferentes componentes da cor para facilitar a sua compreensão:

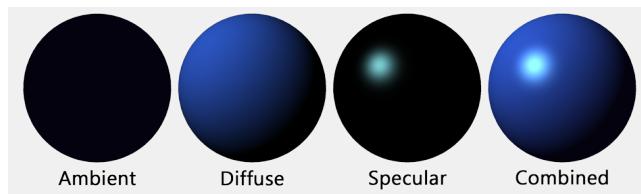


Figura 17: Ilustração das componentes materiais

A componente emissiva (não aparece na imagem acima) corresponde à cor que o objeto emite independente da luz (esta componente não emite luz).

### 3.2 XML Parser

De forma a aplicar as texturas e a escolher as componentes de material do objeto estabeleceu-se a seguinte sintaxe para o formato xml:

```
<model command="plane 4" texture="wood.jpg" diffuse="_amb" ambient="_amb"
specular="_spec" emission="_emi" shininess="128"/>
```

Estes novos atributos dedicados são:

**texture** - que especifica o caminho para a imagem da textura que queremos aplicar ao objeto.  
**diffuse** - para especificar a componente difusa do material do objeto.  
**ambient** - para especificar a componente ambiente do material do objeto.  
**specular** - para especificar a componente especular do material do objeto.

`emission` - para especificar a componente emissiva do material do objeto.  
`shininess` - para especificar o valor de brilho do material

Segundo o formato especificado de fases anteriores tínhamos que, para especificar uma cor de um modelo declara-la previamente num nodo de `colors` identificadas por um `name` e quando queríamos colocar a cor no objeto fazíamos `color="nome_da_cor"`.

Como foi acrescentada iluminação à *engine* deixou-se de invocar `color` e passou-se a especificar as componentes do material .

A fim de aproveitar essa declaração prévia das cores os atributos descritos acima respetivos ao material do modelo têm como valor o `name` da cor correspondente que queremos atribuir

Assim sendo temos de seguida um **exemplo**:

```
<colors>
    <color name="null" r="0.0" g="0.0" b="0.0" a="1.0"/>
    <color name="white" r="1.0" g="1.0" b="1.0" a="1.0"/>
</colors>

. . .

<model command="patch patches/teapot.patch 5" texture="assets/_wood.jpg"
diffuse="null" ambient="null" specular="null"
emission="white" shininess="128"/>
```

### 3.3 Skybox

Uma das adições feitas nesta fase foi a possibilidade de acrescentar uma *skybox* para dar ambiente ao cenário.

No fundo esta skybox é só uma box com 0 divisões, *width* fixa para os 3 eixos e com os triângulos invertidos



Figura 18: Skybox do cenário

**Exemplo de XML:**

```
<skybox width="500" texture="assets/skybox_nebula.png"/>
```

### 3.4 Movimentação da câmera

Para movimentar a câmera esféricamente já tínhamos as 'Arrow Keys' que alteravam a posição esférica. Agora com as teclas 'WASD' é possível movimentar posicionalmente o ponto de foco sobre o eixo xOz.

Para conseguir inserir esta funcionalidade aderimos ao cálculo do vetor unitário da direção frontal da câmera e depois obteve-se o vetor lateral através do produto interno desse vetor e da posição da câmera relativa ao foco ( $p - f$ ).

Para saber a posição do foco da câmera o modo de debug indica esta com um cubo branco.

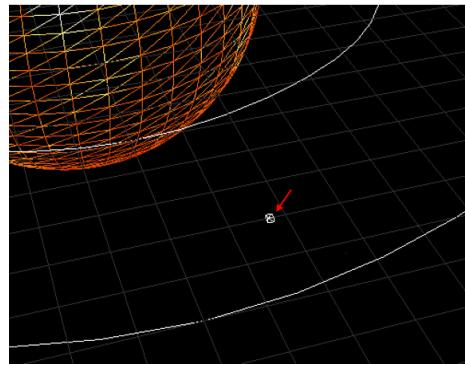


Figura 19: Cubo que indica a posição de foco da camera

### 3.5 MipMapping

Para colocar as texturas mais *smooth* ativamos Mipmapping com minification filtering a **GL\_LINEAR\_MIPMAP\_LINEAR**.

## 4 Lighting

As luzes no nosso sistema são organizadas todas num array de 8 luzes que podem conter qualquer um dos 3 tipos de luzes descritas abaixo.

Uma luz pode especificar componentes **difusa**, **ambiente** e **especular**.

**Ativação das luzes:** Quando as luzes são criadas , apenas são ativadas as que estão a ser utilizadas

**Exemplo de XML:**

```
<lights>

    <!-- LIGHT 0 -->
    <light type="POINT" posX="-5" posY="5" posZ="0" attenuation="0"/>

    <!-- LIGHT 1 -->
    <light type="DIRECTIONAL" dirX="5" dirY="5" dirZ="0" attenuation="0"/>

    <!-- LIGHT 2 -->
    <light type="SPOT" posX="-3.25" posY="0" posZ="0" dirX="-1" dirY="0"
          dirZ="0" cutoff="10" attenuation="0"/>

</lights>
```

**Nota:** O parser do ficheiro XML dá um erro caso se tente meter mais do que 8 luzes.

### 4.1 Light Types

Existem diferentes tipos de luz, que têm em conta a posição, a direção e o ângulo de emissão do objeto emissor desta, que são respetivamente descritas de seguida. Estas componentes são consideradas características extrínsecas.

#### 4.1.1 Point Light

Quando a luz é apenas um ponto (*point light*) significa que existe um ponto que envia luz em todas as direções. O objeto vai receber dependendo da posição quantidades de luz diferentes.

```
GLfloat pos[4] = {0.0, 0.0, 10.0, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION, pos);
```

#### 4.1.2 *Directional Light*

Quando a luz é direcional (*directional light*) significa que existe uma luz direcional que envia raios de luz paralelos entre si numa determinada direção. Independentemente da posição a face do objeto virada para a luz receberá sempre a mesma quantidade de luz.

```
GLfloat spot[3] = {0.0, 0.0, -1.0};  
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spotDir);
```

#### 4.1.3 *Spotlight*

Quando a luz têm um ângulo de emissão (*spot light*), significa que está iluminar o objeto naquela determinada posição com um certo ângulo de abertura. Este ângulo varia entre 0 e 90 ou 180.

```
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
```

De seguida, apresenta-se uma imagem ilustrativa para facilitar a distinção das diferentes características extrínsecas da luz.

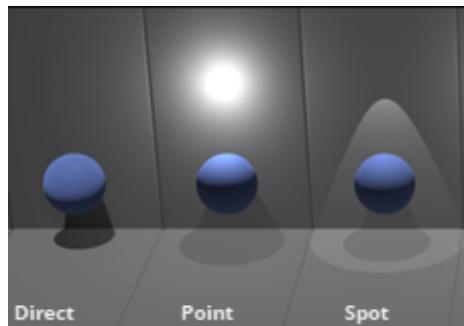


Figura 20: Características extrínsecas da luz

## 5 Optimizações

### 5.1 *View Frustum Culling*

Com o intuito de otimizar a performance de renderização implementamos View Frustum culling para cada objeto a partir de uma bounding box.

1. Cada objeto, quando é criado, gera para si automaticamente uma bounding box (BBox) que será usada para verificar se está dentro do frustum ou não;
2. De cada vez que queremos desenhar o objeto tiramos a matriz A (Projection x ModelView) e calculamos os vetores normais dos planos do *frustum*. Com isto, obtemos os planos do frustum no *clip space*;

3. Finalmente, para verificar se a BBox está dentro do frustum verifica-se se algum vértice está no seu interior, considerando que nalguns sitios a distância do vértice ao plano é 0.

Se o resultado da verificação for verdadeiro então desenha-se o modelo e caso contrário este não é desenhado.

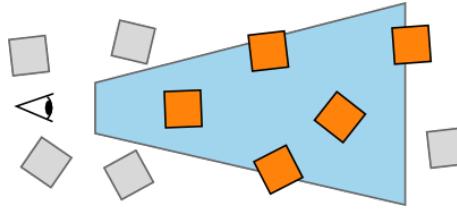


Figura 21: Imagem ilustrativa do processo de culling

## 5.2 Remodelação RGB Color

Devido à mudança de atribuição de cores para materiais, tivemos que refazer a *keyword* especial `rgb`.

Antes este modo `rgb` era só aplicado à cor do objeto , mas agora este é aplicado a qualquer uma das componentes de cor do material (*diffuse*, *ambient*, *specular* e *emissive*).

Um dos problemas da implementação anterior é que dependia de variáveis globais para *tracking* do estado anterior da cor para o novo estado. Isto também criava problemas quando se adicionava vários modelos com `rgb`, pois o estado global era partilhado e quantos mais modelos havia com `rgb` a mudança de cores ficava cada vez mais rápida.

De forma a contrariar este problema pesquisamos sobre o assunto e implementamos este modo `rgb` de uma forma mais correta utilizando conversão de componentes HSV de uma cor saturada para `rgb[1]` a partir de um dado ângulo. Esse ângulo é dado a partir do tempo tirado da máquina (com isto não há dependência de nenhum estado).



Figura 22:

### 5.3 Debug Info

Um pormenor que adicionamos ao modo de debug para informar o utilizador sobre o estado da *engine*.

- FPS - Frames Per Second
- Time Multiplier - A constante de tempo com o qual as animações se baseiam



Figura 23: Debug Info

## 6 Cenários



Figura 24: tardis.xml



Figura 25: house.xml



Figura 26: house.xml



Figura 27: house.xml



Figura 28: house.xml



Figura 29: flashlight.xml



Figura 30: flashlight.xml



Figura 31: flashlight.xml

## 6.1 Sistema Solar

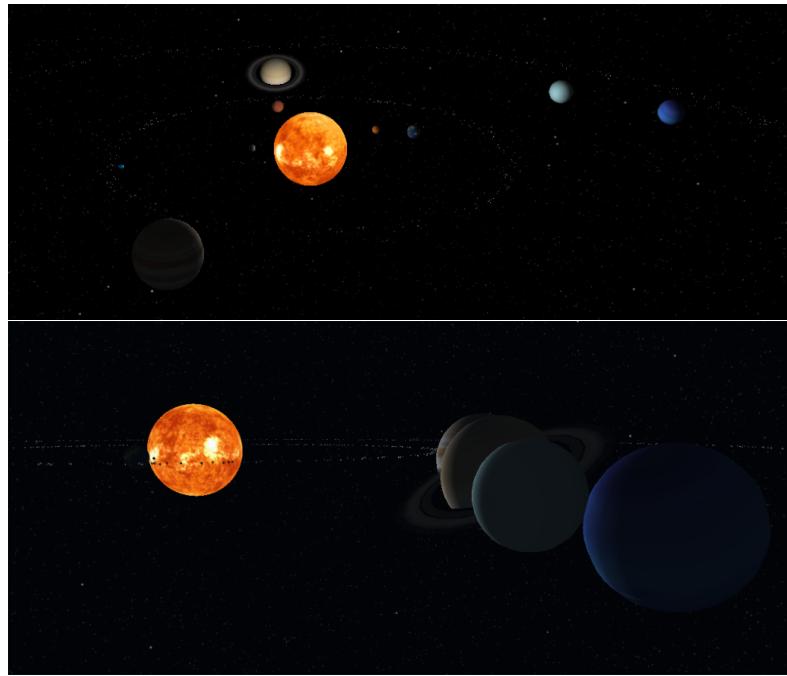


Figura 32: Sistema solar

## 7 Conclusão

Este trabalho permitiu-nos não só consolidar a matéria dada na unidade curricular de computação gráfica como também foi desafiantes e nos permitiu descobrir novas áreas da ciência de computação. Para além disso possibilitou-nos alargar os nossos horizontes permitindo-nos fazer vários cenários ao nosso gosto e feitio, cativando-nos ainda mais para a realização do projeto.

## Referências

- [1] **Sobre HSV e algoritmo de conversão de HSV para RGB:**  
[https://en.wikipedia.org/wiki/HSL\\_and\\_HSV#HSV\\_to\\_RGB](https://en.wikipedia.org/wiki/HSL_and_HSV#HSV_to_RGB)