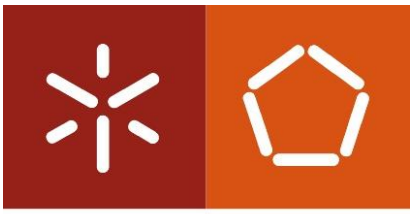


Universidade do Minho

Licenciatura em Engenharia Informática


# Computação Gráfica

Fase 4 - Normals and Texture Coordinates

Grupo 21


Duarte Parente (A95844) Gonçalo Pereira (A96849)

José Moreira (A95522) Santiago Domingues (A96886)


Ano Letivo 2022/2023

# Índice

# Capítulo 1

# Introdução

O presente relatório visa apresentar a quarta e última fase do projeto a desenvolver no âmbito da Unidade Curricular de Computação Gráfica. Ao longo do documento irá ser explicada a metodologia adotada para a resolução dos problemas, assim como as diversas opções tomadas ao longo desta fase. O principal objetivo desta fase passa pela implementação de iluminação e texturas, de forma a, tal como na fase anterior, conferir à representação do sistema solar um elevado grau de realidade.

Em semelhança à fase 3, foi necessário alterar nesta fase ambas as aplicações, `Generator` e `Engine`. No que toca ao `Generator`, foi necessário criar os algoritmos de obtenção das normas de cada vértice e das coordenadas das texturas, em cada figura, para de seguida colocar a informação nos ficheiros dos modelos. Relativamente ao `Engine`, as alterações passaram pela atualização do parser para suportar as novas atualizações do ficheiro de configuração XML.

Finalmente, e ainda com vista em tornar a representação do sistema solar o mais fiel possível à realidade, foi adicionada também a cintura de asteroides, localizada entre Marte e Júpiter. O processo de como foi criada está também explicado de forma mais detalhada numa fase posterior deste documento.

# Capítulo 2

# Generator

## 2.1 Nova Primitiva Gráfica - *Plane Torus*

Nesta fase, o grupo decidiu também adicionar uma nova primitiva gráfica, o *plane torus*. Tal como o nome indica, esta é semelhante ao torus original, porém, sem altura, ficando com a forma de um torus "achatado".

## 2.2 Geração das normais

### 2.2.1 Plane

Uma vez que o plano não possui altura, o cálculo das suas normais foi uma tarefa muito fácil. Com isto, atribuiu-se os pontos de normal (0,1,0) a todos os vértices de triângulos pertencentes à face superior do plano e (0,-1,0) a todos os pontos pertencentes à face inferior do mesmo.

### 2.2.2 Plane torus

À semelhança do plane e, esta primitiva gráfica apresenta a sua componente y sempre igual a zero. Com isto, o cálculo das normais efetuou-se exatamente da mesma forma, ou seja, os pontos (0,1,0) a todos os vértices voltados para a parte positiva do do eixo dos y e (0,-1,0) aos que se encontram voltados para a parte negativa do mesmo.

### 2.2.3 Sphere

O cálculo das normais da esfera mostrou-se, também, uma tarefa simples. Sendo a normal em cada vértice o vetor direcionado entre o centro da esfera, ou seja, o centro do referencial, e o vértice, apenas se necessitou de normalizar todos os vértices presentes na mesma.

### 2.2.4 Box

Uma vez que esta primitiva é composta por três primitivas do tipo *plane*, apenas surgiu a necessidade de complementar o código corresponde a esta última. Tudo isto uma vez que a primitiva *plane* é capaz de gerar planos com distanciamento entre a sua face posterior e a sua face inferior, orientados para qualquer um dos três diferentes eixos. Desta forma, as normais surgiram naturalmente com valores (0,1,0) e (0,-1,0) para os planos superior e inferior, respetivamente, e (0,0,1), (0,0,-1), (1,0,0), (-1,0,0) para os planos laterais.

### 2.2.5 Cone

Procedeu-se ao cálculo das normais na primitiva do tipo *cone*. Este mesmo cálculo dividiu-se em duas partes distintas: cálculo das normais na base do cone e cálculo das normais no corpo do cone. No primeiro caso, atribuiu-se o valor (0,-1,0) a todos os vértices. O segundo caso exigiu uma capacidade de raciocínio maior, onde se conseguiu chegar à fórmula (sin(ang*j), sin(atan(radius / height), cos(angle*j), onde, posteriormente, se dividiu todos os valores por 2 e efetuou-se a respetiva normalização.

### 2.2.6 Bezier

Em relação à primitiva de *bezier*, surgiu a necessidade de calcular as normais para cada triângulo. Isto efetuou-se calculando dois vetores comuns a um mesmo triângulo da seguinte forma: v1 = (p3x-p1x, p3y-p1y, p3z-p1z) e v2 = (p2x-p1x, p2y-p1y, p2z-p1z). De seguida, procedeu-se ao cálculo do produto destes dois vetores e consequente normalização. Após esta fase, procedeu-se ao cálculo médio das normais dos vértices comuns, isto é, calculou-se a normal média para aqueles pontos que pertencem a mais do que um triângulo, atribuindo, assim, ao trabalho, uma abordagem teórica de Phong.

## 2.3 Geração de Texturas

### 2.3.1 Plane

O cálculo das coordenadas de textura no plano foi relativamente simples pois este já se encontra na forma de uma imagem de textura, visto não possuir qualquer valor de altura. Sendo assim, os pontos foram mapeados diretamente, para cada vértice associou-se uma coordenada de textura.

### 2.3.2 Box

Uma box é gerada através de pares de planos, logo as coordenadas de textura para a caixa ficam definidas a partir dos planos gerados.

### 2.3.3 Sphere

O mapeamento dos pontos da esfera é realizado de forma análoga à sua construção,ou seja, para cada ponto da esfera as coordenadas de textura serão calculadas através da *slice* e da *stack* do ponto em questão.

### 2.3.4 Cone

A abordagem para as coordenadas de textura do cone é semelhante à da esfera, na medida em que após se identificar as coordenadas de textura iniciais (ponto (0,0)) os restantes pontos de textura são calculados através da *stack* e da *slice* em que o ponto de encontra.

### 2.3.5 Bezier

Para esta primitiva o mapeamento das coordenadas de textura foi feito de acordo com que cada ponto de bezier contenha uma coordenada de textura, de forma a invés de se associar a cada patch uma textura uniforme, associar a cada ponto uma porção da textura. Desta forma a quantidade de vezes que a textura se repete é menor, obtendo-se o mapeamento pretendido.

# Capítulo 3

# Engine

## 3.1 Alterações no Ficheiro de configuração

Como referido anteriormente, esta nova fase trouxe novas extensões ao ficheiro de configuração. Neste momento, este ficheiro pode conter a definição do modelo com texturas da seguinte forma:

```
<models>
    <model file="sphere.3d" >
        <!-- if texture is not specified then the
              model is rendered with color only-->
        <texture file="earth.jpg" />
        <!-- if color is not specified, use the values
              below as default -->
        <color>
            <diffuse R="200" G="200" B="200" />
            <ambient R="50" G="50" B="50" />
            <specular R="0" G="0" B="0" />
            <emissive R="0" G="0" B="0" />
            <shininess value="0" />
        </color>
    </model>
</models>
```

Como se observa acima, o ficheiro XML pode agora conter informação das texturas, e quanto às cores, é possível que sejam colocadas as suas características(*diffuse, ambient, specular, emissive, shininess*).

Para além disto, é possível também ter os tipos de fonte de iluminação que serão aplicados. Existem 3 tipos (*point, directional* e *spotlight*), e são aplicados no XML da seguinte forma:

```
<lights>
    <light type="point" posX="0" posY="10" posZ="0" />
```

```
    <light type="directional" dirX="1" dirY="1" dirZ="1"/>
    <light type="spotlight" posX="0" posY="10" posZ="0"
                            dirX="1" dirY="1" dirZ="1"
                            cutoff="45" />
</lights>
```

## 3.2 Iluminação

Para a aplicação da iluminação, o grupo criou três novas classes, uma para cada tipo de fonte de luz. Foram criadas então as classes `light_point`, `light_directional` e `light_spotlight`. Tal como indicado no enunciado do projeto, a primeira recebe as coordenadas da sua posição, a segunda recebe uma direção (e não posição), e a terceira recebe uma posição, uma direção e ainda o valor do `cutoff`.

Tal como para a estruturação das transformações optou-se pelo uso de uma super classe *Light*, onde as classes definidas abaixo se apresentam como extensões da mesma.

```
class Light{
public:
    virtual void apply_light(){};
};

class Light_point : public Light{
public:
    Point pos;
    Light_point(float _x, float _y, float _z);
    void apply_light();
};

class Light_directional : public Light{
public:
    Point dir;
    Light_directional(float _x, float _y, float _z);
    void apply_light();
};

class Light_spotlight : public Light{
public:
    Point pos;
    Point dir;
    float cutoff;
    Light_spotlight(Point s, Point dir, float cutoff);
    void apply_light();
};
```

Em relação ao aparecimento diferentes fontes de luz no ficheiro de configurações, foi dada a possibilidade de o número máximo ser 8, tal como o OpenGL permite. Dessa forma cada luz aparece associada a um identificador único de forma a poder ser corretamente inicializada e usada.

## 3.3 Alterações no Parser

Nesta secção serão abordadas as alterações feitas no parser. À função `parseModel` foram adicionadas as variáveis para capturar cada um dos campos `rgb` dos primeiros quatro componentes da luz (*diffuse, ambient, specular, emissive, shininess*), e também para capturar o valor do componente que resta, o *shininess*. De seguida a função chama o método `add_color`, passando como argumento os valores capturados anteriormente. É também feito o parse do campo da textura.

Outra das alterações realizadas no parser foi a adição do método `parseLight()`, de forma a chamar os construtores do tipo de fonte de luz com os devidos valores, uma vez que nesta função são obtidos os valores dos campos de cada tipo de fonte de luz.

Por último, na função `parse`, foi adicionada a condição para identificar quando é chamada a função `parseLight` referida anteriormente.

## 3.4 Ficheiros .3d

Relativamente à estrutura dos ficheiros 3d dos modelos, esta foi alterada de modo a ficar mais apropriada às novas atualizações no projeto. Como referido nos relatórios de fases anteriores, a estrura destes ficheiros até ao inicio desta fase era a seguinte (por linha):

```
n (nº de pontos)
coordenadas_ponto1
coordenadas_ponto2
(...)
coordenadas_pontoN
```

Uma vez que nesta fase os ficheiros têm também que conter as coordenadas das normais e dos pontos de textura, a estrutura destes ficheiros passou a ser a:

```
n (nº de pontos)
coordenadas_ponto1 coordenadas_normal_ponto1 coordenadas_textura
coordenadas_ponto2 coordenadas_normal_ponto2 coordenadas_textura
(...)
coordenadas_pontoN coordenadas_normal_pontoN coordenadas_textura
```

Como está claro, esta alteração é imprescindível uma vez que assim se obtém toda a informação necessária relativa aos pontos, para depois executar as operações (de desenho, das normais e das texturas) necessárias para a representação.

# Capítulo 4

# Análise de Resultados

Neste capítulo serão apresentados os resultados do programa quando confrontado com os testes e respetivos ficheiros de configuração disponibilizados pela equipa docente. Para esta última fase, foram disponibilizados pelos docentes 6 testes.

Seguem-se representados os resultados obtidos com o programa desenvolvido (metade esquerda da figura), e respetiva comparação com o resultado esperado (metade direita da figura).

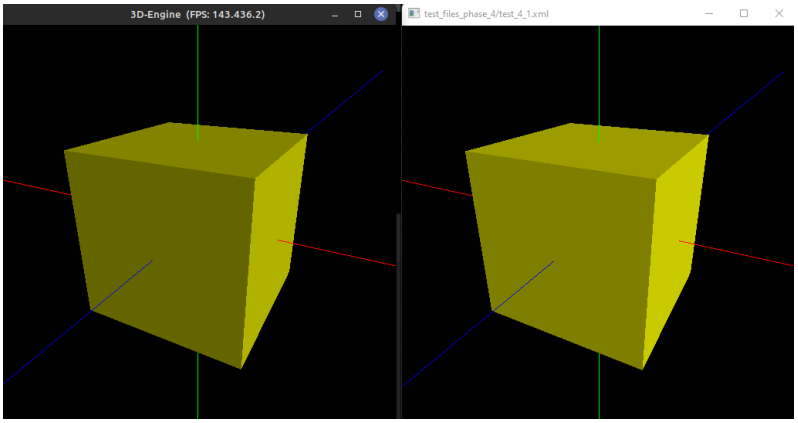


Figura 1: Teste 1

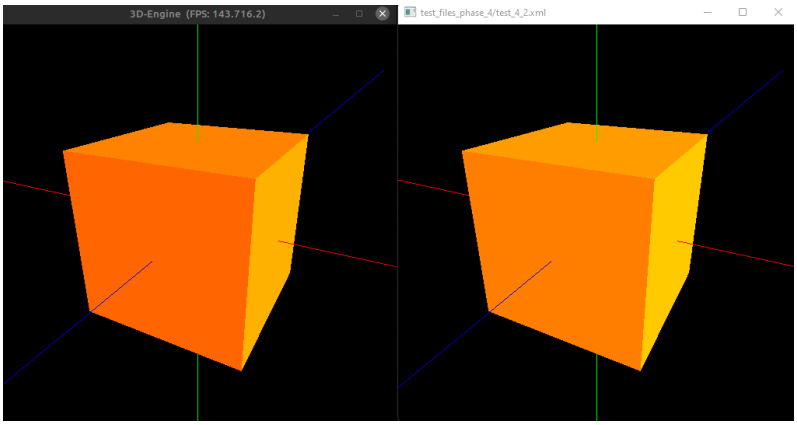


Figura 2: Teste 2

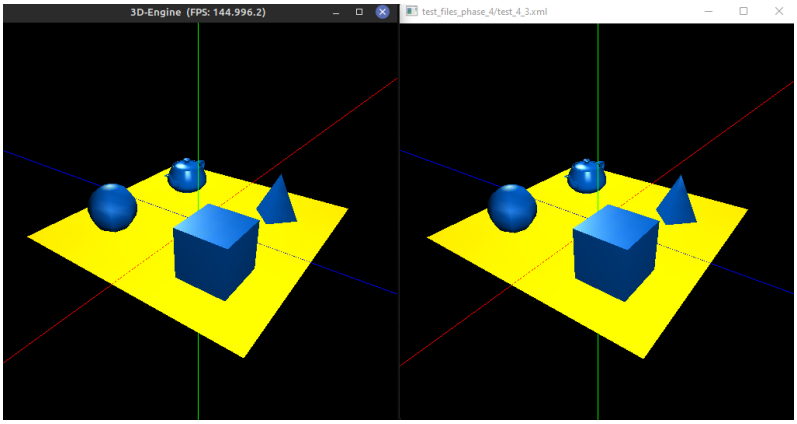


Figura 3: Teste 3

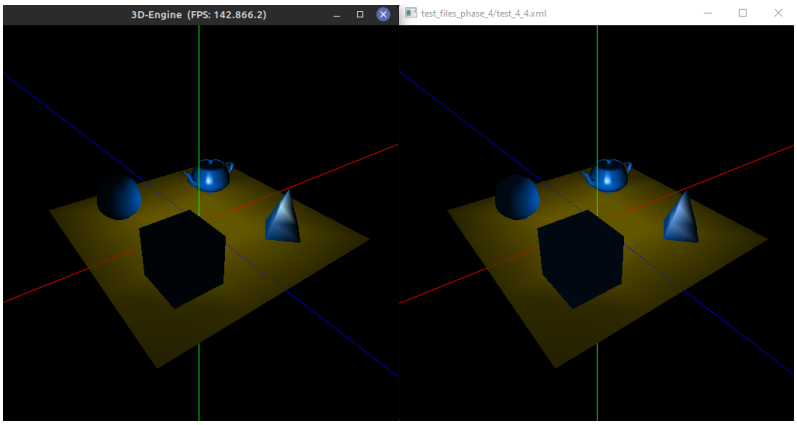


Figura 4: Teste 4

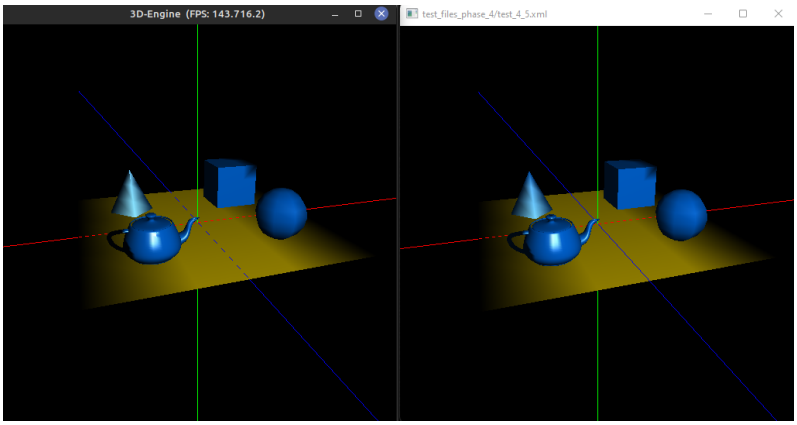


Figura 5: Teste 5

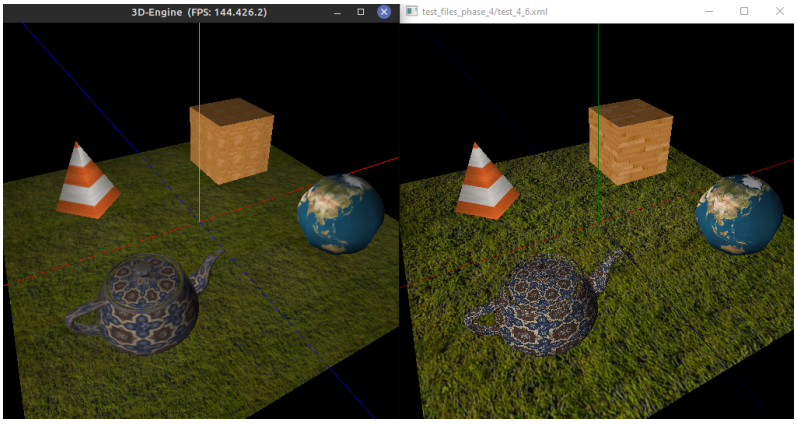


Figura 6: Teste 6

# Capítulo 5

# Sistema Solar

## 5.1 Implementação da Cintura de Asteroides

A cintura de asteroides trata-se de uma região circular, localizada aproximadamente entre as órbitas de Marte e Júpiter, formada por uma grande quantidade de objetos irregulares com variados tamanhos denominados de asteroides. A implementação deste elemento foi um dos objetivos definidos pelo próprio grupo aquando da realização da fase 3.

A estratégia escolhida pelo grupo para a sua representação, foi a geração de várias esferas (no exemplo da figura foram 1000) com diferentes tamanhos. Em relação ao tamanho, os valores do *scale* destas esferas são gerados aleatoriamente de entre quatro possíveis(0.01, 0.012, 0.015 e 0.017). No que toca à distância ao centro do referencial, as coordenadas variam também de forma aleatória num intervalo de 39 a 42.5. Por último, o grupo decidiu também variar no seu valor no eixo dos yy, para se aproximar da realidade, valor esse que varia de -0.5 a 0.5.

## 5.2 Representação Final do Sistema Solar

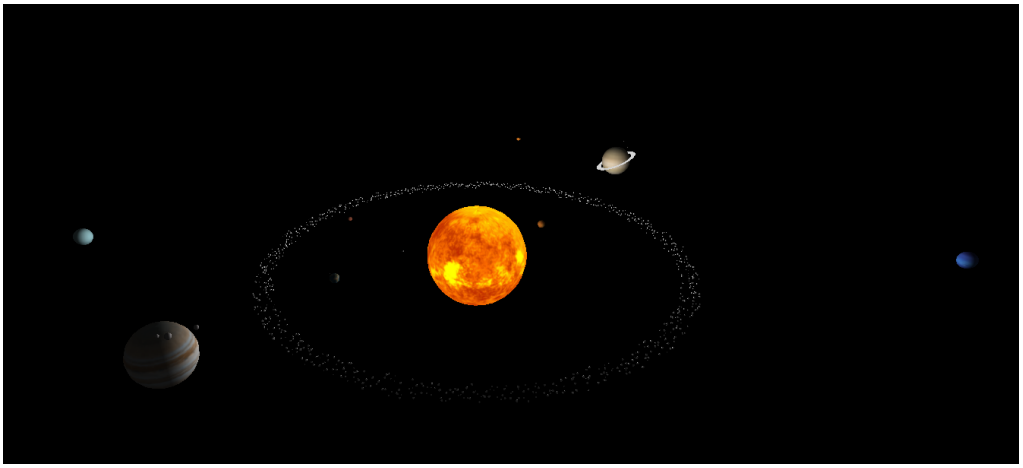

Figura 7: Representação do Sistema Solar - vista frontal

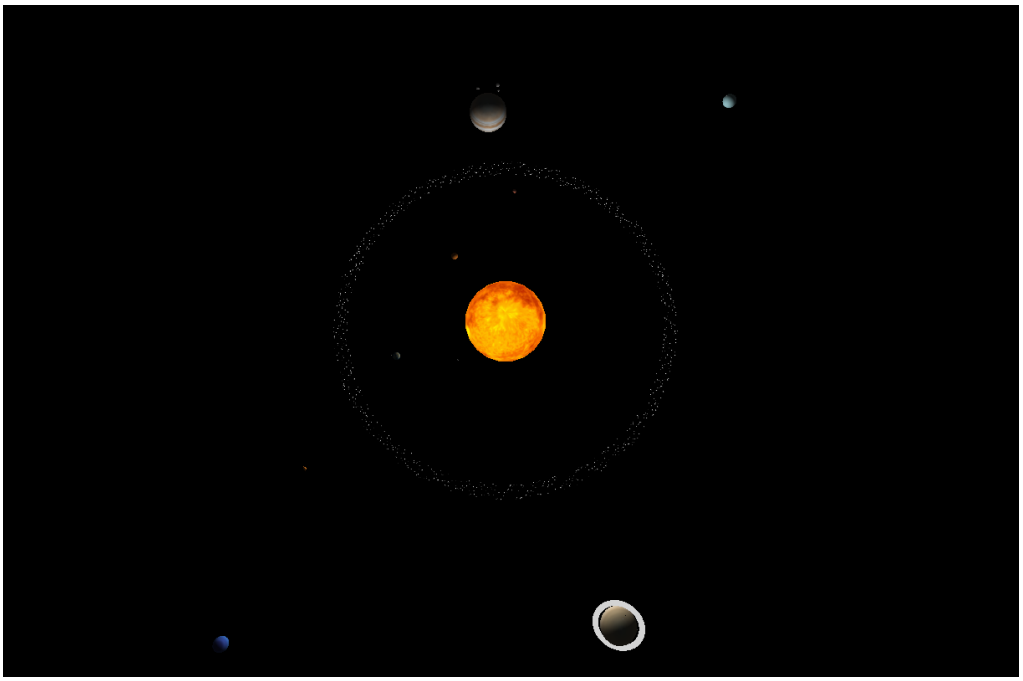

Figura 8: Representação do Sistema Solar - vista de cima

É de notar que na figura 7 (vista frontal do sistema solar), à primeira vista, parece que apenas existe luz num dos lados da cintura de asteroides, na metade mais afastada do ponto de vista. Isto dá-se ao facto de ter sido implementada uma iluminação realista. Qualquer que seja o ponto de vista, vai sempre parecer que os objetos entre "nós"e o sol estão escuros, uma vez que a luz aponta na nossa direção, o que faz com que a face dos objetos que está iluminada seja sempre a que está virada para o sol.

# Capítulo 6

# Conclusão

Dá-se assim por terminado o trabalho prático em grupo no âmbito da unidade curricular de Computação Gráfica. Após análise dos resultados obtidos, tanto em relação aos testes disponibilizados pela equipa docente, como na representação do sistema solar, conclui-se que foram respondidos com sucesso os principais objetivos do trabalho.

Embora se tenham verificado algumas dificuldade nas etapas referentes à implementação da iluminação, e também no cálculo das normais dos pontos das figuras, visto que envolve cálculos distintos para cada primitiva, é possível concluir que estas foram ultrapassadas.

No que toca ao trabalho futuro, o grupo tem como objetivo a colocação das estrelas no sistema solar a implementação da câmara no modo de explorador, de forma a ter uma navegação mais completa pelo mesmo.