



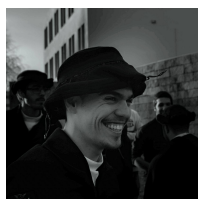
**Universidade do Minho**

# Computação Gráfica

MIEI - 3º ANO - 2º SEMESTRE

UNIVERSIDADE DO MINHO

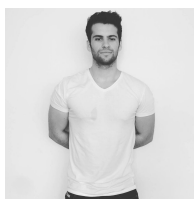
## TRABALHO PRÁTICO



João Leal  
A75569



Rui Vieira  
A74658



Carlos Faria  
A67638



Ricardo Leal  
A75411

21 de Maio de 2018

# *Conteúdo*

<b>1</b>	<b>Introdução</b>	<b>2</b>
1.1	Contextualização . . . . .	2
1.2	Resumo . . . . .	2
<b>2</b>	<b>Arquitetura de Código</b>	<b>3</b>
2.1	Aplicações . . . . .	3
2.1.1	Generator . . . . .	3
2.1.2	Engine . . . . .	3
2.2	Classes . . . . .	4
2.2.1	Light . . . . .	4
2.2.2	Group . . . . .	5
<b>3</b>	<b>Generator</b>	<b>6</b>
3.1	Aplicação de normais e pontos de textura . . . . .	6
3.1.1	Plano . . . . .	7
3.1.2	Paralelepípedo . . . . .	8
3.1.3	Esfera . . . . .	9
3.1.4	Torus . . . . .	10
3.1.5	Bezier Patches . . . . .	12
<b>4</b>	<b>Engine</b>	<b>14</b>
4.1	Descrição . . . . .	14
4.2	VBOs e texturas . . . . .	14
4.2.1	Iluminação . . . . .	14
<b>5</b>	<b>Resultados obtidos</b>	<b>16</b>
<b>6</b>	<b>Conclusão</b>	<b>18</b>

# 1. *Introdução*

## 1.1 Contextualização

No âmbito da Unidade Curricular de Computação Gráfica, foi nos posposto a realização de um mecanismo de 3D onde foram sendo aplicados múltiplas ferramentas usadas ao longo do semestre.

Este projeto foi seccionado, em 4 partes, sendo esta a quarta e ultima fase que tem como objetivo a inclusão de texturas e iluminação no trabalho previamente realizado, com a finalidade de criar um modelo dinâmico do Sistema Solar.

## 1.2 Resumo

Tratando-se da ultima parte de um projeto prático, é natural que mantenham funcionalidades desenvolvidas nas fases anteriores, e que outras sejam alteradas de forma a cumprir os requisitos necessários.

Esta fase traz consigo várias novidades que irão levar a várias alterações, tanto ao nível do engine como do generator.

Começando pelo generator, nesta fase, este passará a conseguir ser capaz de obter as normais e pontos de textura para os vários vértices das primitivas geométricas anteriormente criadas.

Passando para o engine, este sofrerá algumas modificações e, ao mesmo tempo, receberá também novas funcionalidades. Os ficheiros XML passarão a conter as informações relativas à iluminação do cenário. Assim, deste modo, terá que ser alterado não só o parser responsável por ler esses mesmos ficheiros, assim como, o modo como é processada toda a informação recebida com o intuito de gerar todo o cenário pretendido.

A última modificação, e não menos importante, que o engine sofrerá, está relacionada com a preparação dos VBOs e das texturas durante o processo de leitura de informação proveniente do ficheiro de configuração.

Todas as alterações tem como finalidade gerar de forma eficaz um modelo do Sistema Solar ainda mais realista, passando a ter a texturas e a iluminação necessárias.

## 2. *Arquitetura de Código*

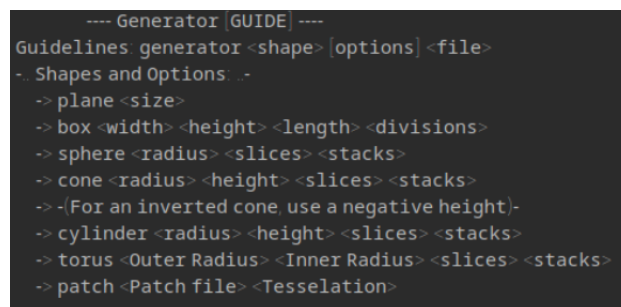
Tratando-se de uma continuação do trabalho desenvolvido nas fases anteriores, mantendo inalterado grande parte do código, contudo de forma a cumprir os requisitos necessários propostos para esta fase parte do código foi alterado e criado outro.

### 2.1 Aplicações

Nesta secção são apresentadas as aplicações fundamentais que permitem gerar e exibir os diferentes cenários pretendidos. Na realização desta fase houve alterações significativas em ambas as aplicações do projeto, de modo a que os requisitos propostos pela mesma fossem devidamente cumpridos.

#### 2.1.1 Generator

**generator.cpp** -Tal como explicado em fases anteriores, trata-se da aplicação onde estão definidas as estruturas das diferentes primitivas geométricas a desenvolver de forma a gerar os respetivos vértices. A aplicação cria uma pasta "files" e guarda os ficheiros gerados dentro da mesma. Para além das primitivas gráficas previamente desenvolvidas, na realização desta fase é pretendido que o gerador seja capaz de gerar também as normais e coordenadas de textura para cada vértice.



```
---- Generator [GUIDE] ----
Guidelines generator <shape> [options] <file>
- Shapes and Options: -
-> plane <size>
-> box <width> <height> <length> <divisions>
-> sphere <radius> <slices> <stacks>
-> cone <radius> <height> <slices> <stacks>
-> -(For an inverted cone, use a negative height)-
-> cylinder <radius> <height> <slices> <stacks>
-> torus <Outer Radius> <Inner Radius> <slices> <stacks>
-> patch <Patch file> <Tesselation>
```

Figura 2.1: Apresentação do ficheiro generator.h

#### 2.1.2 Engine

**engine.cpp** - Aplicação que permite a apresentação de uma janela exibindo os modelos pretendidos, lidos a partir de um ficheiro XML (criado pelo utilizador), e ainda a interação com estes mesmos modelos, através de comandos. Durante a realização desta fase do projeto inevitavelmente surgiram alterações relativamente à fase anterior implementando funcionalidades de iluminação e representação de texturas.

```

# ENGINE [GUIDE] #

Usage: ./engine {XML FILE}
       [-h]

FILE
Specify a path to an XML file in which the information about
the models you wish to create are specified

MOVE
w. Move Forward

s. Move Back

a. Strafe Left

d. Strafe Right

q. Turn Left (Third Person Only)

e. Turn Right (Third Person Only)

+. Increase Camera Speed

-. Decrease Camera Speed

FORMAT:
p. Change the figure format into points

l. Change the figure format into lines

m. Fill up the figure

WINDOW:
f. Change to Fullscreen Mode

ESC. Close Window
#

```

Figura 2.2: Apresentação do ficheiro engine.h

## 2.2 Classes

### 2.2.1 Light

**light.h** - Classe que representa cada uma das origens de iluminação que são pontos emitindo luz em todas as direções.

```

#ifndef _LIGHTS_H_
#define _LIGHTS_H_

#include <GL/glut.h>

class Light {
    float pos[4];

public:
    Light(float *p);
    void renderLight();
};

#endif

```

Figura 2.3: Apresentação do ficheiro light.h

## 2.2.2 Group

**group.cpp**- Classe responsável pelo armazenamento de toda a informação necessária à representação de um dado modelo, possuindo para isso três VBOs(pontos de construção de triângulos, normais e coordenadas de textura) e respectivos tamanhos. De maneira a possibilitar associar texturas aos modelos, necessita de possuir informação sobre textura, mais especificamente o seu ID, e por fim uma componente encarregue pelas cores produzidas através da iluminação.

```
#ifndef _GROUP_H_
#define _GROUP_H_

#include <GL/glut.h>
#include <vector>
#include "transforms.h"
#include "../src/shape.h"

class Model{

    float buff_size[1];
    GLuint buffers[1];

public:
    void setUp(std::vector<Vertex*> vert);
    void renderModel();
    ~Model(void);
};

class Group{

    std::vector<Transform*> transforms;
    std::vector<Model*> models;
    std::vector<Group*> children;

public:
    Group();
    std::vector<Transform*> getTransforms();
    std::vector<Model*> getModels();
    std::vector<Group*> getChilds();
    void pushTransform(Transform* t);
    void pushModel(Model* model);
    void pushChild(Group* c);
    ~Group(void);
};

#endif
```

Figura 2.4: Apresentação do ficheiro group.h

## 3. *Generator*

### 3.1 Aplicação de normais e pontos de textura

De forma a obter as normais e pontos de textura para as figuras geométricas desenvolvidas implicou o estudo das faces e vértices constituintes das mesmas.

O estudo das normais consiste em obter um vetor normal para cada vértice, que constitui a figura geométrica.

Quanto aos pontos de textura, implicou o estudo de um plano 2D para a figuras geométricas 3D, usando como revestimento das mesmas. O mapeamento é feito da seguinte forma:

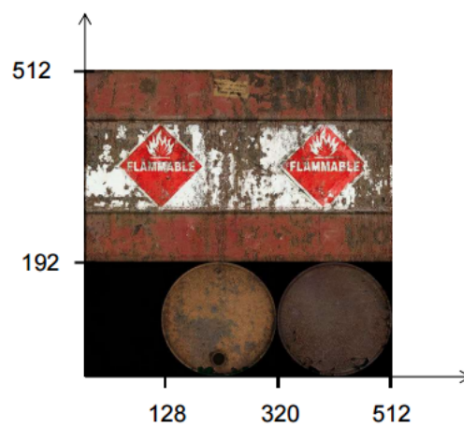


Figura 3.1: Espaço da imagem real.

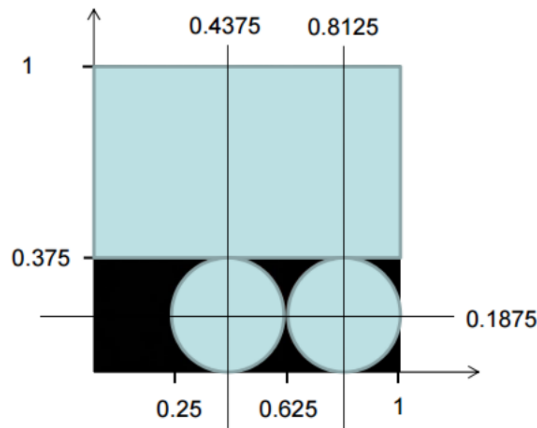


Figura 3.2: Espaço da textura.

Tal como podemos verificar na ilustração anterior, estabelecemos que o eixo cartesiano da textura tem como limite máximo igual e um limite mínimo de zero. E de notar que estes limites correspondem aos limites da imagem real, no entanto, mapeados para valores entre 0 e 1 usando o quanto inferior esquerdo de todas as imagens.

### 3.1.1 Plano

De forma a o conjunto dos vetores normais, de cada vértice, é apenas necessário verificar o plano cartesiano em que o plano está desenhado. Como este plano se encontra em  $xOz$ , todos os vértices partilham a mesma normal, vetor  $(0,0,1)$ .

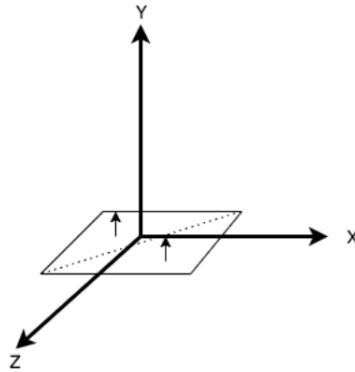


Figura 3.3: Referencial cartesiano, com os vetores normais do plano.

O processo de obtenção dos pontos de textura é simples no caso desta figura geométrica. O formato é mesmo que o da imagem 2D sendo apenas necessário fazer a correspondência direta de cada vértice do plano com os vértices da imagem 2D.



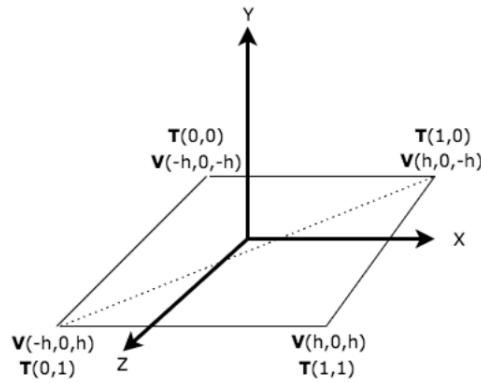


Figura 3.4: Referencial cartesiano, com o mapeamento dos vértices com a imagem 2D

### 3.1.2 Paralelepípedo

Tal como no plano, de forma a obter o conjunto dos vetores normais para cada vértice, necessitamos verificar o plano cartesiano de cada face do paralelepípedo. Facilmente reconhecemos os correspondentes vetores normais a cada uma das faces do mesmo:

- **Face Frontal** - vetor  $(0,0,1)$
- **Face Traseira** - vetor  $(0,0,-1)$
- **Face Direita** - vetor  $(1,0,0)$
- **Face Esquerda** - vetor  $(-1,0,0)$
- **Topo** - vetor  $(0,0,1)$
- **Base** - vetor  $(0,0,-1)$

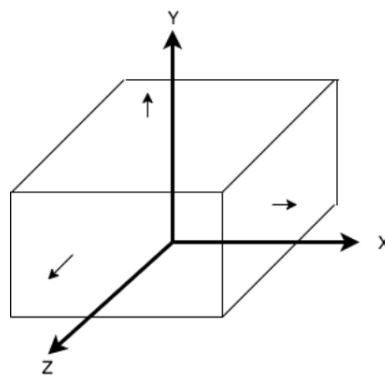


Figura 3.5: Referencial cartesiano, com os vetores normais de 3 faces

Para obtenção dos pontos de textura do paralelepípedo, estabelecemos uma imagem 2D como regra possuindo sempre o seguinte formato:

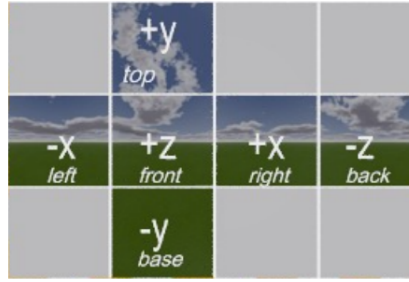


Figura 3.6: Exemplo de imagem 2D para textura de uma paralelepípedo.

Para cálculo dos pontos de textura obtemos a posição da imagem a que corresponde a face em questão e iterar no mesmo sentido que o paralelepípedo é desenhado, atribuindo textura a cada vértice o ponto de textura correspondente. Tal como é ilustrado na figura seguinte:

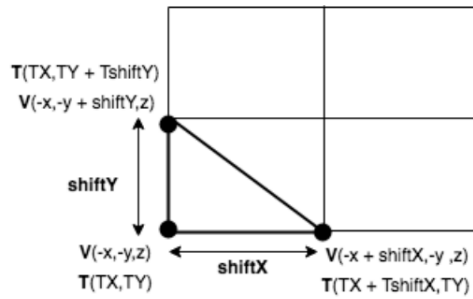


Figura 3.7: Processo de cálculo dos pontos de textura para cada face frontal.

De notar que o ponto  $(TX, TY)$  corresponde ao canto inferior esquerdo da face frontal, na imagem 2D, já os valores  $TX$  e  $TY$  correspondem à posição onde se encontra a face frontal.

### 3.1.3 Esfera

De forma a obter o conjunto dos vetores normais da esfera bastava apenas a orientação da origem do referencial até ao ponto em questão, algo que já fazemos no processo de desenho da esfera. De notar que interessa-nos apenas saber a direção a partir da origem até ao ponto e não a distância até o mesmo. Obtendo o resultado ilustrado na imagem seguinte.

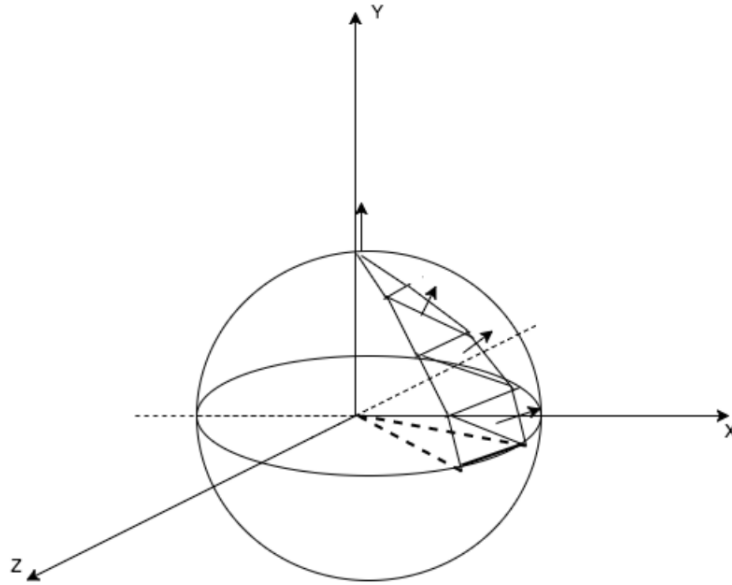


Figura 3.8: Referencial cartesiano, com os vetores normais de uma fatia da esfera.

O vetor normal de um dado vértice  $V(x,y,z)$  é dado por:

- $N(\sin(PH), y/\text{raio}, \cos(PH))$  - PH corresponde ao passo (desvio) horizontal, que se faz para iterar a circunferência que compõe o centro da esfera.

Para obtenção do ponto de textura  $T(t1,t2)$  de um dado vértice  $V(x,y,z)$  recorreremos à seguinte solução:

- $t1 = (-\text{atan2}(-x,z) + \pi) / 2\pi$  ;
- $t2 = 1 - ((-y/\text{raio}) + 1) / 2$  ;

Conseguindo desta forma mapear os pontos de textura de uma imagem 2D normal, para os vértices de uma esfera, revestindo a mesma com a imagem.

### 3.1.4 Torus

Tal como na esfera no Torus basta a orientação dos vetores normais de cada vértice no momento do desenho que coincide com orientação da origem até ao vértice que já calculamos.

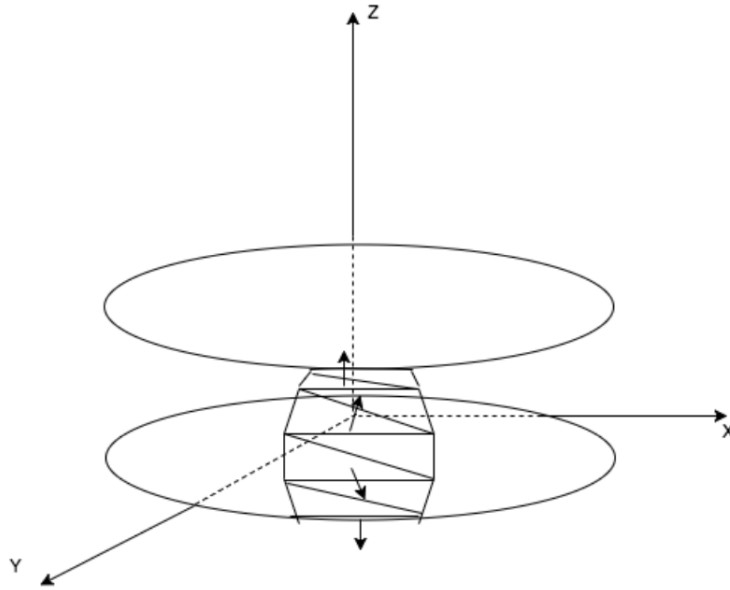


Figura 3.9: Referencial cartesiano, com os vetores normais de uma fatia da torus.

Obtendo os vetores normais apartir da seguinte expressão:

- $N (\cos(DA)) \cdot \cos(DL), \sin(DA) \cdot \cos(DL), \sin(DL))$  - DA corresponde ao desvio do anel, e o DL ao desvio de cada lado formando um anel. Desta forma, num processo iterativo conseguimos obter todos os vetores normais.

Quanto ao pontos de textura, o mapeamento ocorre de forma simples. A imagem padrão para textura do torus é um retângulo muito largo. Com isto, para revestir o torus, vamos atribuir uma tira da imagem 2D e envolver esta num anel do torus. Aplicando este processo iterativamente, conseguimos revestir o torus completamente. O processo iterativo no lado da imagem 2D corresponde apenas a percorrer a tira correspondente ao anel. As seguintes imagens ilustram melhor o processo:

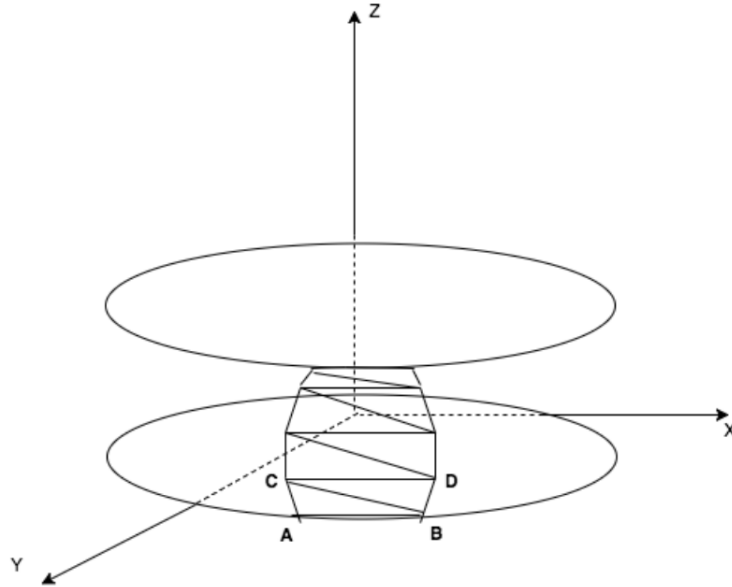


Figura 3.10: Referencial cartesiano, com os vértices de uma fatia da torus.

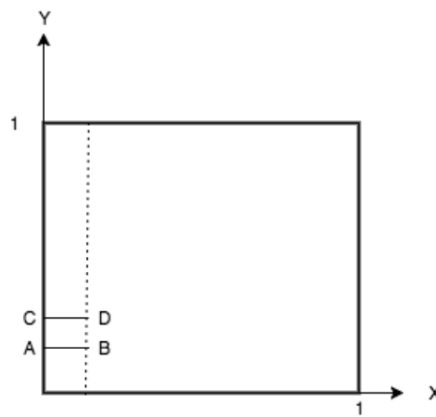


Figura 3.11: Referencial cartesiano, o mapeamento de textura e os vértices do torus.

Utilizando para cálculo dos pontos de textura a seguinte expressão.

- $T(\text{NA/slice}, \text{NL/sides})$  - NA correspondendo ao número total de anéis e NL ao número total de lados.

### 3.1.5 Bezier Patches

De forma a obter o conjunto de vetores normais dos vértices gerados pelos ficheiros de configuração optamos por um processo através de 4 pontos, da mesma forma, obtendo as tangentes usando as seguintes fórmulas das derivadas:

$$\text{Let } U = [u^3 \quad u^2 \quad u \quad 1] \text{ and } M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Figura 3.12: Matriz U e M.

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

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

Figura 3.13: Derivadas parciais de u e v.

Para um dado u e v, o processo de obtenção das normais consiste apenas em calcular as derivadas parciais de B(u,v) para u e v. O resultado das derivadas parciais, é a tangente e com esta chegamos as normais de cada vértice. A normal, é o produto escalar entre as tangentes normalizadas obtidas das derivadas parciais de u e v. Com isto, o resultado depois de normalizado, corresponde ao vetor normal do ponto.

## 4. *Engine*

### 4.1 Descrição

O *Engine* é responsável pelo armazenamento da informação dos modelos a representar, assim como as respetivas transformações geométricas de cada um desses modelos, que no seu conjunto, irão formar o sistema solar que queremos representar. Esta também resultou na implementação de novas características no motor do modelo. Findando o processo de leitura do ficheiro, passamos para a fase seguinte, a qual é responsável pela renderização da informação, previamente obtida.

### 4.2 VBOs e texturas

Em paralelo com a fase anterior continuamos a fazer uso de VBOs (Vertex Buffer Object) para passar diretamente para a placa de vídeo um array com os vértices dos objetos a serem gerados. Com a implementação de textura e luz tornou-se necessário criar mais 2 VBOs, um com as normais e outra com a textura dos pontos.

```
void Model::setUpVBO(vector<Vertex*> vert, vector<Vertex*> norm, vector<Vertex*> tex){
    buff_size[0] = vert.size();
    buff_size[1] = norm.size();
    buff_size[2] = tex.size();

    float* vertex_array = (float*) malloc(sizeof(float) * (vert.size()) * 3);
    float* normal_array = (float*) malloc(sizeof(float) * (norm.size()) * 3);
    float* texture_array = (float*) malloc(sizeof(float) * (tex.size()) * 2);
```

Figura 4.1: Construção de VBOs

#### 4.2.1 Iluminação

A iluminação dos modelos gerados no projecto é corretamente obtida através do cálculo das várias normais do mesmo, sendo que é através destas que se pode obter a intensidade da luz.

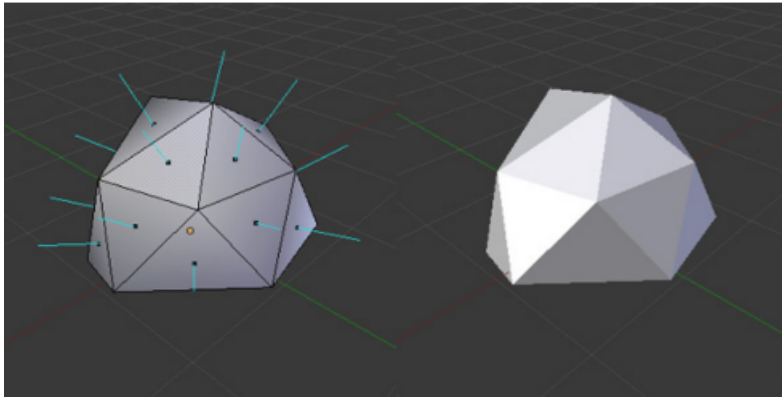
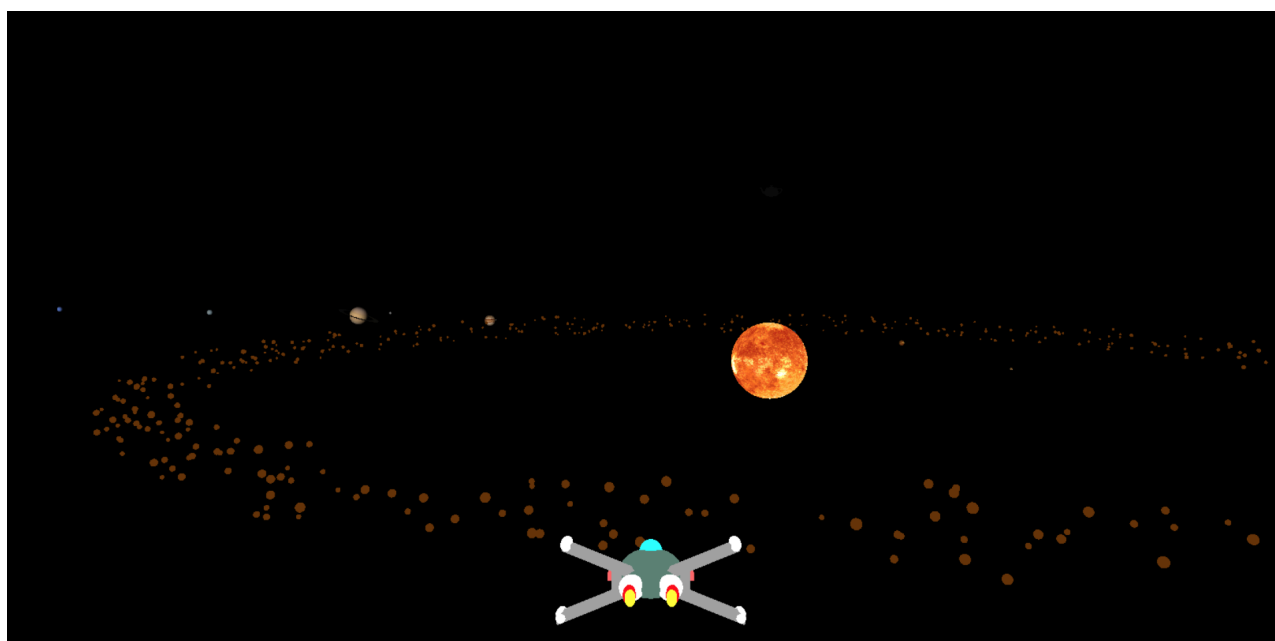
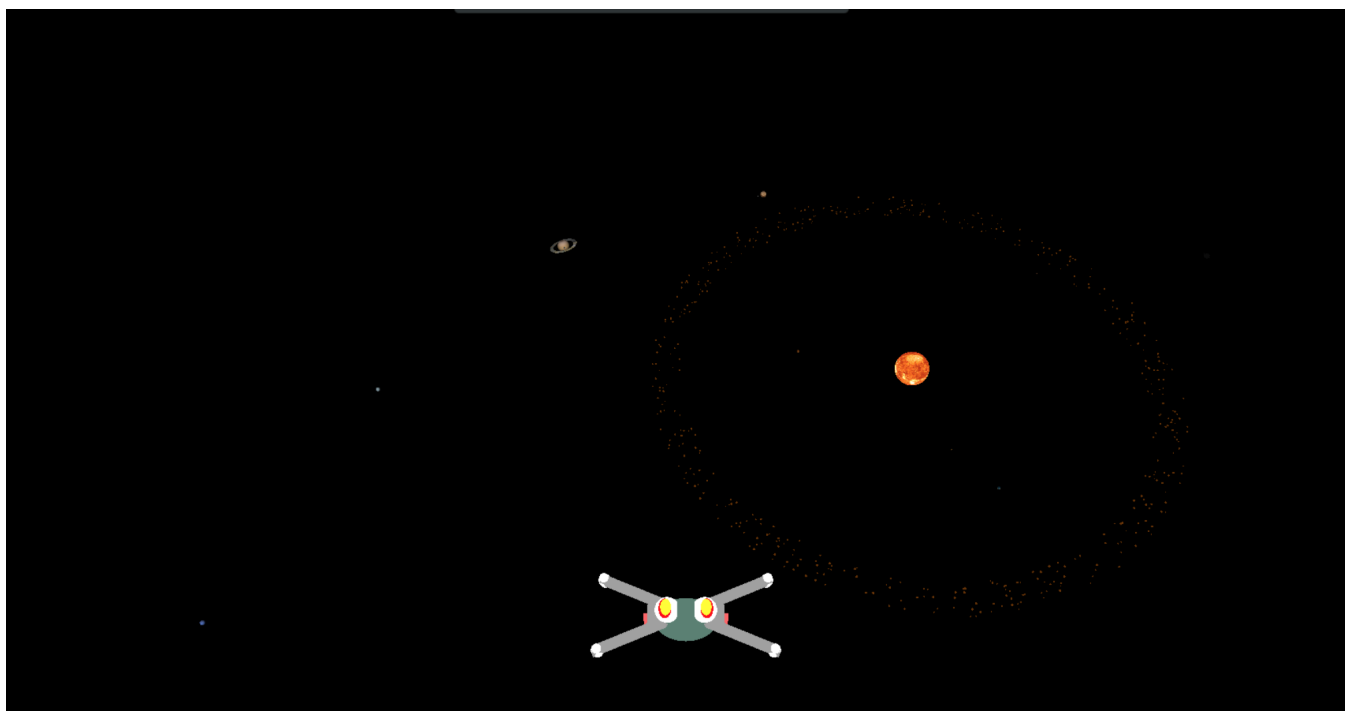
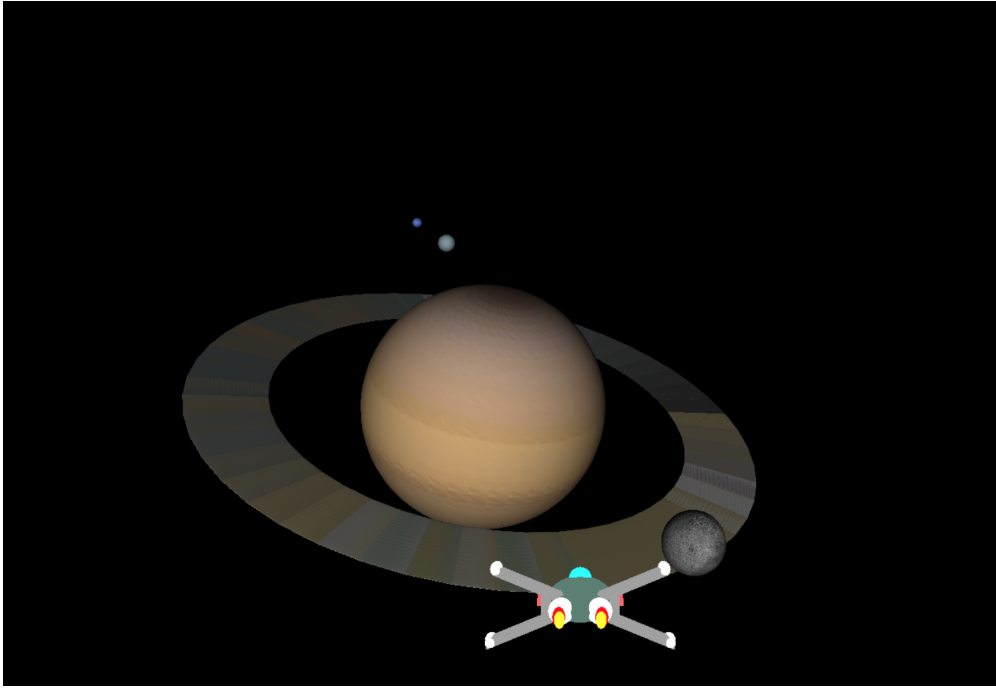


Figura 4.2: Funcionamento das normais e iluminação



## 5. *Resultados obtenidos*





## 6. *Conclusão*

A elaboração desta última fase do trabalho consideravelmente menos trabalhosa em relação às anteriores. Devendo-se não só ao facto de os requisitos necessários serem relativamente menores, mas também à experiência que já adquirimos durante as fase anteriores.

Em suma e fazendo uma retrospectiva sobre todo o desenvolvimento do projeto sentimo-nos orgulhosos do trabalho realizado cumprindo todos os requisitos e ainda algumas funcionalidades recreativas extra. Funcionalidades essas que passam por renderizar uma nave que dispara esferas capazes de destruir meteoritos. Sentimo-nos aptos e com boas bases para desenvolver eventuais futuros projetos na área da Computação Gráfica.