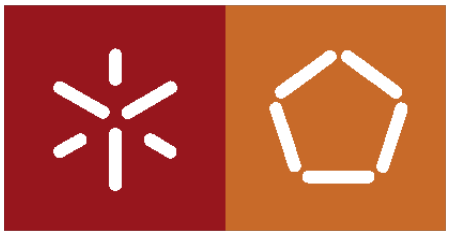

Universidade do Minho

MIEI - 3º ano - 2º semestre
Universidade do Minho

Trabalho Prático - Fase 4

Grupo 11


João Vieira
A76516


Manuel Monteiro
A74036


Bernardo Viseu
A74618


Fernando Pereira
A75496

June 4, 2019

Conteúdo

1 Introdução

Este é o último relatório do trabalho prático proposto na cadeira de Computação Gráfica. Apesar do projeto ter sido dividido em 4 fases, foi sempre em torno do mesmo programa e cada fase é a continuação da anterior. Quer isto dizer que esta fase não é exceção em relação às anteriores e o código da terceira fase foi alterado de modo a cumprir com os novos objetivos propostos.

É pedido que o gerador seja capaz de gerar coordenadas de textura e normais para os vários vértices das primitivas geométricas criadas anteriormente.

Também o motor foi modificado, passando a ter mais funcionalidades. Para esta última parte vai passar a conseguir gerar um cenário com texturas e iluminações de acordo com o código `.xml` recebido, sendo que como é óbvio também o parser teve de ser refeito pois vai agora receber também as informações relativas às texturas e iluminações.

Com estas novas funcionalidades esperamos que o nosso Sistema Solar se torne ainda mais realista de maneira a obtermos um cenário gráfico completo.

2 Arquitetura

Sendo esta a continuação do trabalho já desenvolvido, é normal que a estrutura seja mantida, necessitando apenas de algumas classes novas. No entanto foram feitas várias alterações à estrutura da aplicação, de modo a incluir as novas funcionalidades.

2.1 Aplicações

Aqui faremos menção às aplicações do nosso programa, que são a parte mais importante pois são elas que permitem gerar os cenários desejados.

2.1.1 Gerador

O gerador, como o próprio nome indica, sempre teve, desde a primeira fase, o propósito de gerar os vértices das figuras geométricas. A partir de agora o gerador também é capaz de gerar as normais e as coordenadas de textura para os respetivos vértices.

```
#################################################################
#                       Generator MENU                          #
#     Usage:                                                    #
#     ./generate <shape> [options] <file>                       #
#                                                               #
#     Shapes & Options:                                         #
#        -> plane <size>                                        #
#        -> box <width> <height> <length> <divisions>           #
#        -> sphere <radius> <slices> <stacks>                   #
#        -> cone <radius> <height> <slices> <stacks>            #
#        -> cylinder <radius> <height> <slices>                 #
#        -> torus <innerRadius> <outerRadius> <slices> <rings>  #
#        -> patch <path_to_path_file> <tesselation> <shapename> #
#################################################################
```

Figura 1: Menu de utilização do Gerador.

2.1.2 Motor

Com as alterações feitas ao Gerador, também o motor teve de ser alterado, de modo a ser possível aplicar as novas funcionalidades de iluminação e representação de texturas do nosso Sistema Solar.

```
##############################
#         Engine MENU        #
#    Usage:                  #
#    ./engine path_to_XML    #
#                            #
#    KeyBinds:               #
#       w - Rotate up        #
#       s - Rotate down      #
#       a - Rotate left      #
#       d - Rotate right     #
#                            #
#       j - Fill Mode        #
#       k - Line Mode        #
#       l - Point Mode       #
#                            #
#      '-' - Move Cam Back   #
#      '+' - Move Cam In     #
#                            #
#       m - Make axis longer #
#       n - Make axis smaller#
#                            #
#       c - Reset colors     #
##############################
```

Figura 2: Menu de utilização do motor.

2.2 Classes

2.2.1 Light

Classe que representa cada uma das posições de origem da iluminação e realiza a renderização da mesma.

2.2.2 Material

Esta classe é encarregue de guardar as diferentes componentes das cores produzidas, através da iluminação (difusa, especular, ambiente e emissiva).

3 Gerador

3.1 Normais e Pontos de Textura

Para obtermos as normais e os pontos de textura das figuras geométricas desenvolvidas já em fases anteriores tivemos que analisar bem e estudar as faces e vértices que as representam.

O objetivo é então obter um vetor normal para cada vértice das figuras geométricas, para obter o ângulo de incidência da iluminação do sistema.

Já para as texturas foi necessário fazer um mapeamento.

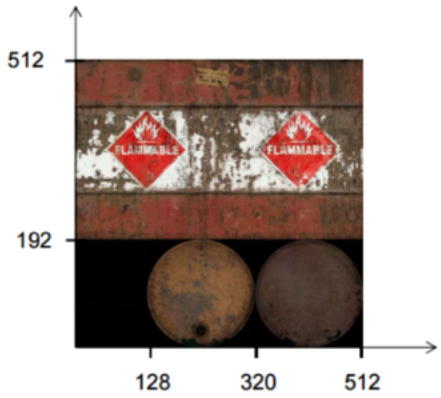

Figura 3: Imagem real.

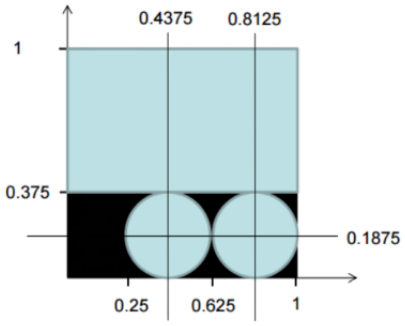

Figura 4: Espaço da textura.

Como é de notar na imagem, os pontos da textura serão calculados, para cada face do sólido geométrico, desdobrado em apenas duas dimensões. Foi então necessário estabelecer a origem do eixo cartesiano da textura, que é limitada entre X=0 e X=1 e entre Y=0 até Y=1. A origem encontra-se no canto inferior, como é costume.

3.1.1 Esfera

Como já anteriormente referimos nas últimas fases, conseguimos obter os vetores normais da esfera na altura do desenho. Apenas interessa saber a direção a partir da origem até ao ponto e não a distância até ao mesmo. Obtemos assim o resultado como é apresentado na imagem a seguir:

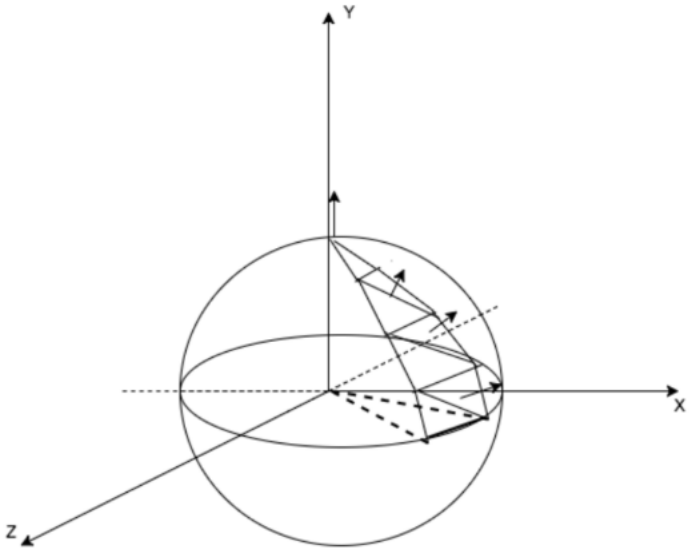

Figura 5: Vetores normais de uma parte da esfera.

O cálculo do vetor normal de cada vértice é dado pela função *normalcalc*, a qual é falada mais à frente neste relatório.

Para se obter os pontos de textura basta efetuar a divisão de 1 pelo número de *Stacks/Slices* e assim temos os valores necessários para definir os pontos da mesma. Depois houve o cuidado na definição dos pontos para que estes fossem gerados da mesma ordem que os pontos dos vértices, correspondendo o nosso *mapping* com os vertíces da esfera. Obtemos então os seguintes pontos:

$v1(x1, y1)$
$x1 = TexU$
$y1 = TexV$

$v2(x2, y2)$
$x2 = TexU$
$y2 = 2TexV$

$v3(x3, y3)$
$x3 = 2TexU$
$y3 = TexV$

$v4(x4, y4)$
$x4 = 2TexU$
$y4 = 2TexV$

$$TexU = 1/slices$$
$$TexV = 1/stacks$$

3.1.2 Torus

Semelhante ao que acontece na esfera, também no Torus é possível obter os vetores normais de cada vértice no momento do desenho.

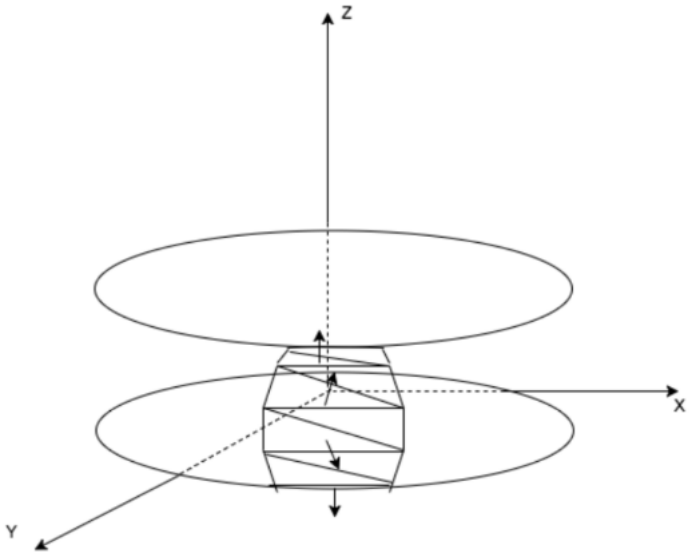

Figura 6: Vetores normais de uma parte do torus.

Os pontos de textura também foram obtidos enquanto se desenha o torus. O processo é bem simples pois como a imagem padrão para a textura do torus é um retângulo, apenas atribuímos uma tira da imagem 2D e aplicámo-la num anel do torus. Fazendo isto para cada anel o torus acaba todo revestido. Também na imagem 2D se realiza um processo iterativo de modo a percorrer a tira correspondente ao anel.

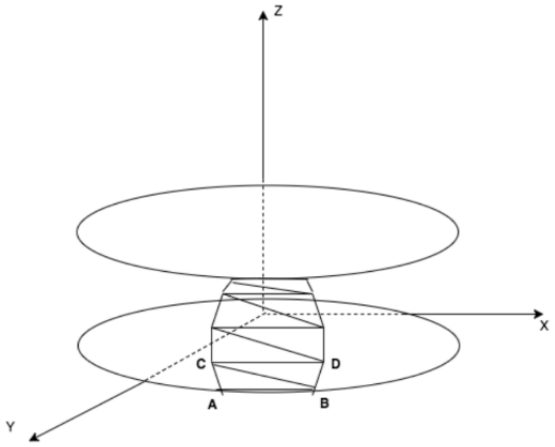

Figura 7: Vértices de uma fatia do torus.

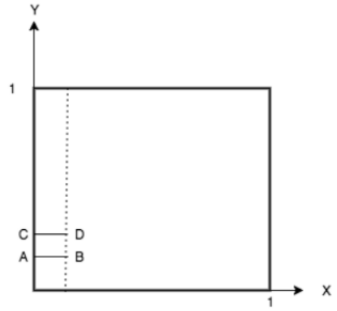

Figura 8: Mapeamento de textura.

Semelhante à esfera, chegámos aos seguintes cálculos, para os pontos de textura:

$$\begin{aligned} &A(x1, y1) \\ &x1 = TexU \\ &y1 = TexV \end{aligned} \qquad \begin{aligned} &B(x2, y2) \\ &x2 = TexU \\ &y2 = 2TexV \end{aligned}$$

$$\begin{aligned} &C(x3, y3) \\ &x3 = 2TexU \\ &y3 = TexV \end{aligned} \qquad \begin{aligned} &D(x4, y4) \\ &x4 = 2TexU \\ &y4 = 2TexV \end{aligned}$$

$$\begin{aligned} TexU &= 1/slices \\ TexV &= 1/rings \end{aligned}$$

4 Engine

4.1 VBOs e texturas

Relativamente ao que foi realizado na fase anterior, continuamos a dar uso aos VBOs (*Vertex Buffer Object*), de modo a permitir passar para a placa gráfica um *array* com os vértices dos objetos a serem gerados. Nesta fase, de maneira a implementar as texturas e a iluminação, foi necessário criar mais 2 VBOs, um com as normais e outro com as texturas dos pontos:

```
void Shape::setUp(){
    float *vertex_array = (float*) malloc(sizeof(float) * vertexes.size() * 3);
    float *normal_array = (float*) malloc(sizeof(float) * normal.size() * 3);
    float *texture_array = (float*) malloc(sizeof(float) * texture.size() * 2);
(...)
```

4.1.1 Iluminação

A iluminação dos modelos gerados é obtida calculando as várias normais do objeto, sendo que através destas obtém-se a intensidade da luz. O cálculo das normais é feito através da seguinte função:

```
Vertex* normalcalc(float x,float y, float z){
    float l;
    l = sqrt(x*x + y*y + z*z);
    return new Vertex(x/l,y/l,z/l);
}
```

Esta função calcula o módulo da distância entre os pontos e retorna a respetiva normal dividindo cada ponto do vértice pelo módulo.

5 Resultados finais

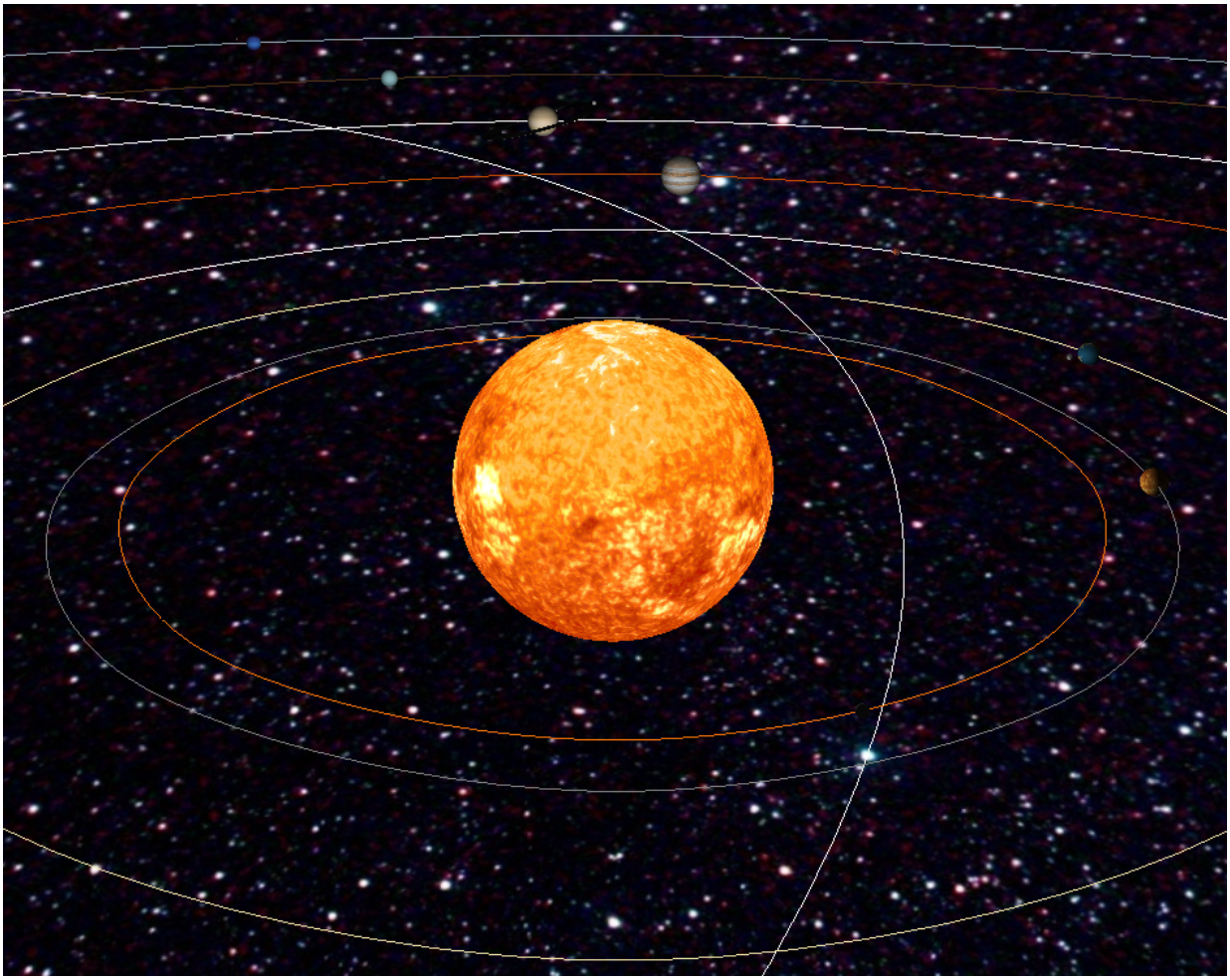

Figura 9: Resultado final do sistema solar.

6 Conclusão

Esta última fase do trabalho foi nos mais acessível no sentido que aplicamos o que aprendemos nas aulas práticas, mas esta implementação não foi direta, tendo sido preciso um estudo intensivo de como deveríamos fazer o *Lightning* de todo o sistema.

Decidimos também implementar uma *Skybox* que consiste numa esfera invertida com uma textura de várias estrelas, para nos dar um *background* espacial ao sistema solar.

Em conclusão, ficamos satisfeitos com o resultado final, e este projeto suscitou-nos muito interesse nesta área gráfica da computação.