Universidade do Minho
Mestrado Integrado em Engenharia Informática

Computação Gráfica

Practical Assignment CG - 2019/20
Phase 4 : Normals and Texture Coordinates

Maio 2020


Angélica Freitas
(A83761)


Joana Afonso Gomes
(A84912)


Rodrigo Pimentel
(A83765)

Conteúdo

1 Introdução

Nesta última fase do projeto de Computação Gráfica foram acrescentadas ao modelo que temos vindo a desenvolver detalhes relativos a **texturas** e **iluminação**, aperfeiçoando o modelo final do Sistema Solar e obtendo uma versão mais realística. Para tal, foi preciso atualizarmos na nossa aplicação o `Generator` – que gera agora as normais e os pontos para as texturas – e o `Engine` – que implementa novas funcionalidades no que toca à iluminação (a partir das normais referidas) e à aplicação de texturas. Em paralelo com este último foi também alterado o ficheiro XML do nosso Sistema para se adaptar à agora existência de luzes e texturas.

No presente relatório iremos apresentar de forma detalhada as mudanças realizadas e a situação final do nosso projeto de desenvolvimento de um cenário com gráficos 3D do Sistema Solar.

2 Generator

2.1 Normais

De modo a facilitar a normalização de vetores, recorremos à seguinte função

```
void normalize(float* a) {

    float l = sqrt(a[0] * a[0] + a[1] * a[1] + a[2] * a[2]);
    a[0] = a[0] / l;
    a[1] = a[1] / l;
    a[2] = a[2] / l;
}
```

2.1.1 Testes

Foi adicionada uma pasta com modelos XML de teste com as primitivas não utilizadas no Sistema Solar. Nestes testes, é possível alterar texturas, luminosidade, material, rodar, fazer tanslações... Os modelos tem nomes como "Caixa", "Cone"... para que o teste das primitivas seja intuitivo.

2.1.2 Plano

As normais de um plano contido no plano XZ são iguais para todos os pontos de um lado, em que as normais do lado oposto é o simétrico do outro. Sendo assim as normais serão apenas `( 0, 1, 0 )` e `( 0, -1, 0 )`.

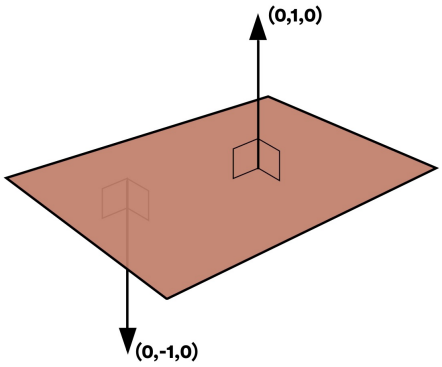

Figura 1: Normais do Plano

2.1.3 Caixa

Para a caixa, sabemos que para cada face, as normais de cada ponto vão ser iguais. Também sabemos que as direções das mesmas vão ser paralelos e colineares ao eixos de coordenadas. A fim de percetir melhor as normais apresentamos a seguinte imagem.

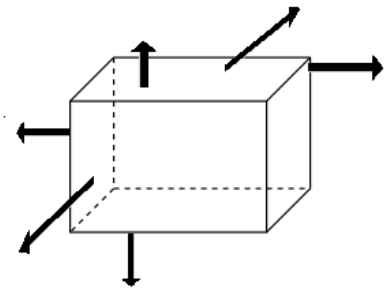

Figura 2: Normais da Caixa

Seja a caixa representada da seguinte forma.

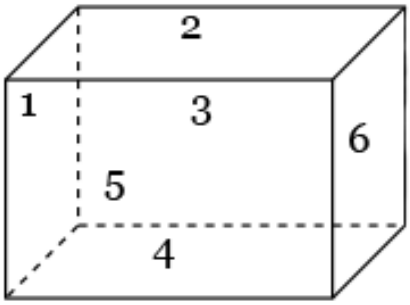

Figura 3: Normais da Caixa

Em que:

- 1 representa a face lateral esquerda;
- 2 representa a face superior;
- 3 representa a face traseira;
- 4 representa a face inferior;
- 5 representa a face frontal;
- 6 representa a face lateral direita.

É de fácil compreensão que as normais são as seguintes para as devidas faces:

$$Face\ 1\ (-1, 0, 0)$$

$$Face\ 2\ (0, 1, 0)$$

$$Face\ 3\ (0, 0, -1)$$

$$Face\ 4\ (0, -1, 0)$$

$$Face\ 5\ (0, 0, 1)$$

$$Face\ 6\ (1, 0, 0)$$

2.1.4 Esfera

As normais da esfera também são muito simples, apenas normalizar os valores de cada coordenada em cada ponto que já foi calculado na fase 1, em que a ordem das normais corresponde também à ordem dos pontos anteriormente calculados.

Consideremos o ponto p1, um dos pontos calculados em cada iteração. Para obtermos a sua normal, apenas teremos de chamar a função normalize listada acima.

2.1.5 Cone

As normais em cada ponto na base vão ser também iguais, tal que, no nosso caso, como a ponta do cone está para cima, terão o seguinte valor.

$$(0, -1, 0)$$

Analogamente à esfera, as normais no corpo do cone serão também a normalização dos pontos calculados na fase 1.

2.1.6 Teapot (*Bezier* Patches)

Sejam os parâmetros passados para a função bezier u e v. Para obter a normal num qualquer ponto da superfície, é necessário calcular os vetores tangentes u e v da seguinte forma:

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

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

Em que cada elemento elevado a T corresponde à transposta da mesma e:

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

O vetor num ponto na superfície é definido pela normalização do resultado do produto interno dos vetores tangentes, obtendo assim o código presente no generator.

2.2 Texturas

A aplicação de texturas em Computação Gráfica passa pela aplicação de uma imagem 2D na superfície de um objeto 3D. Conseguimos obter isso no nosso projeto relacionando os vértices da textura com os do objeto a que queremos aplicá-la.

2.2.1 Plano

Para as coordenadas do plano, apenas temos de mapear valores entre 0 ou 1 visto que não há slices e stacks. Para tal, dependendo de cada vamos ter pontos de textura diferentes.

Por exemplo, para o ponto superior esquerdo do plano, as suas coordenadas de textura serão (0,0), tal como para o ponto inferior direito as coordenadas serão (1,1) e assim sucessivamente.

2.2.2 Esfera

As coordenadas para o mapeamento das texturas seguiram a seguinte imagem e cálculos:

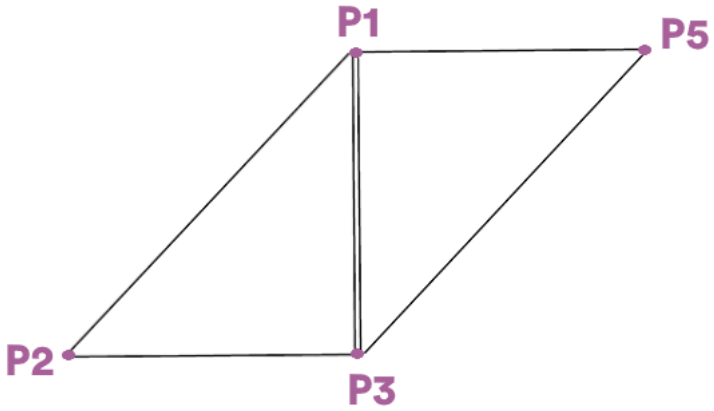

Figura 4: Coordenadas para o mapeamento das texturas.

```
Point textP2(((j- 1) / slices), 1.0 - ((i)/ stacks), 0);
Point textP1(((j)/ slices), 1.0 - ((i - 1) / stacks), 0);
Point textP3( ((j) / slices), 1.0 - ((i) / stacks), 0);
Point textP5( ((j + 1) / slices), 1.0 - ((i - 1) / stacks), 0);
```

Sendo estes pontos usados para mapear a textura inseridos e calculados pela mesma ordem dos pontos da esfera:

```
Triangle textureT1(textP1, textP2, textP3);
Triangle textureT2(textP3, textP5, textP1);
textures.push_back(textureT1);
textures.push_back(textureT2);

Triangle t(p1, p2, p3);
triangles.push_back(t);
Triangle t1(p3, p5, p1);
triangles.push_back(t1);
```

Como desenhamos a esfera de cima para baixo, então para obtermos as coordenadas de texturas corretas tivemos de subtrair 1 na segunda componente.

2.2.3 Caixa

Tal como a esfera, para faces semelhantes (faces cuja diferença entre as mesmas refere numa só coordenada), teremos de subtrarir 1 a uma das componentes visto que se fossem realizadas da mesma maneira, a imagem seria um espelho.

De resto, as coordenadas são parecidos aos pontos calculados já na fase 1, tendo em conta que não pode execeder o tamanho igual a 1. Para tal, teremos de dividir a componente pela qual estamos a iterar. Isto é, se a componente usa o índice i teremos de dividir essa componente por x.

Por exemplo, para a face 1 tinhamos calculado os seguintes pontos para o 1º triângulo.

```
Point p1(0, j + yn, k + zn);
Point p2(0, j + yn, k);
Point p3(0, j, k);
```

Obtemos então os pontos de textura para esse triângulo

```
Point textP1((k + zn)/z,(j + yn)/y, 0);
Point textP2((k) / z, (j + yn) / y, 0);
Point textP3((k) / z, (j) / y, 0);
```

2.2.4 Teapot (*Bezier* Patches)

Como referido na fase anterior temos os seguintes pontos de iteração:

Ora, como as coordenadas dos mesmos são calculados a partir do step, isto é, da divisão da superfície conforme o tecellattion, que, por sua vez, leva às componentes passadas à função do ponto de controlo de bezier (u,u1,v e v1 com step*j, step*(j+1), step*k e step*(k+1), respetivamente) é fácil entender que os pontos necessários para implementar a textura são exatamente esses parametros por cada ponto.

Portanto, se para obter o ponto A precisavamos dos parâmetros u e v, então o ponto de textura para esse mesmo ponto será u e v.

Sendo que

```
auto vectorsPA = getdVectorsUandV(u, v, allPoints, index[i]);
auto vectorsPB = getdVectorsUandV(u, v1, allPoints, index[i]);
auto vectorsPC = getdVectorsUandV(u1, v, allPoints, index[i]);
auto vectorsPD = getdVectorsUandV(u1, v1, allPoints, index[i]);
```

então os seus respetivos pontos de textura serão

```
Point textPA(u, v, 0);
Point textPB(u, v1, 0);
Point textPC(u1, v, 0);
Point textPD(u1, v1, 0);
```

3 Engine

O *Engine*, no que diz respeito à última fase, permite agora a iluminação e a aplicação de texturas ao modelo e/ou mudança do material através do comando *glMaterialfv*, recorrendo às coordenadas que geramos no *Generator*.

3.1 Iluminação

O *Engine* sabe qual o tipo de luz a ser implementada a partir da tag `lights`. As *childs* dessa tag têm uma tag *light* que indica o *type*, que pode ser `POINT`, `DIRETIONAL` ou `SPOT,` e as coordenadas.

- **POINT** : os raios de luz são emitidos em todas as direções, através de um único ponto.
- **DIRETIONAL** : os raios de luz são todos paralelos entre si.
- **SPOT** : age de forma semelhante

No caso do Sistema Solar queremos que a luz seja proveniente do sol, ou seja, o ponto (0,0,0). Por esse motivo, esta é a posição dada no início de ficheiro XML:

```
<lights>
        <light type="POINT" posX="0.0" posY="0.0" posZ="0.0" att="0"/>
</lights>
```

Definimos também no *Engine* uma luz difusa e uma luz ambiente que é aplicada a todos os elementos do modelo.

```
GLfloat amb[3] = { 0.001, 0.001, 0.001 };
GLfloat diff[4] = { 1, 1, 1, 1.0 };
```

Para além disso, temos que definir os parâmetros RGB das cores das componentes ambiente (`ambR`, `ambG`, `ambB`), difusa (`diffR`, `diffG`, `diffB`), especular (`speR`, `speG`, `speB`) e luz emissiva (`emR`, `emG`, `emB`). No exemplo a seguir temos um exemplo de uma `model` em que são atribuídas as componentes da luz emissiva.

```
<model file="sol.3d" texture="sun.jpg" emR="1" emG="1" emB="1"/>
```

3.2 Texturas

Como já referido, utilizamos as coordenadas de textura que geramos no Generator para aplicar as texturas aos vários elementos do nosso modelo. No ficheiro XML, que é carregado pela *Engine* para gerar o Sistema Solar, existe para o efeito agora na *tag* model o elemento `textura`, como no exemplo seguinte.

```
<model file="sol.3d" \textbf{texture}="mercury.jpg" />
```

Além das diversas texturas para os planetas, planetas satélites, cometa (*teapot*) e Sol, adicionamos também uma textura *default* ao *background* do nosso Sistema.

Todo o nosso *Engine* foi cuidadosamente feito para que todos estes parâmetros possam ser dados e deste modo ser executada com os diversos *inputs* de texturas. Há também a possibilidade de a textura sem carregada ou de não ser.

3.3 Ficheiro XML

De seguida é apresentada uma secçao significativa do modelo XML como entregue na última fase do nosso projeto:

```
    <scene>
    <lights>
        <light type="POINT" posX="0.0" posY="0.0" posZ="0.0" />
    </lights>
    <group>
        <!--Sol-->
        <group>
```

```
            <colour R="255" G="226" B="77" />
            <translate time="10">
                <point X="0" Y="0" Z="0" />
            </translate>
            <scale X="10" Y="10" Z="10" />
            <models>
                <model file="sol.3d" texture="sun.jpg"/>
            </models>
        </group>
    (...)
```

4 Modelo final do Sistema Solar

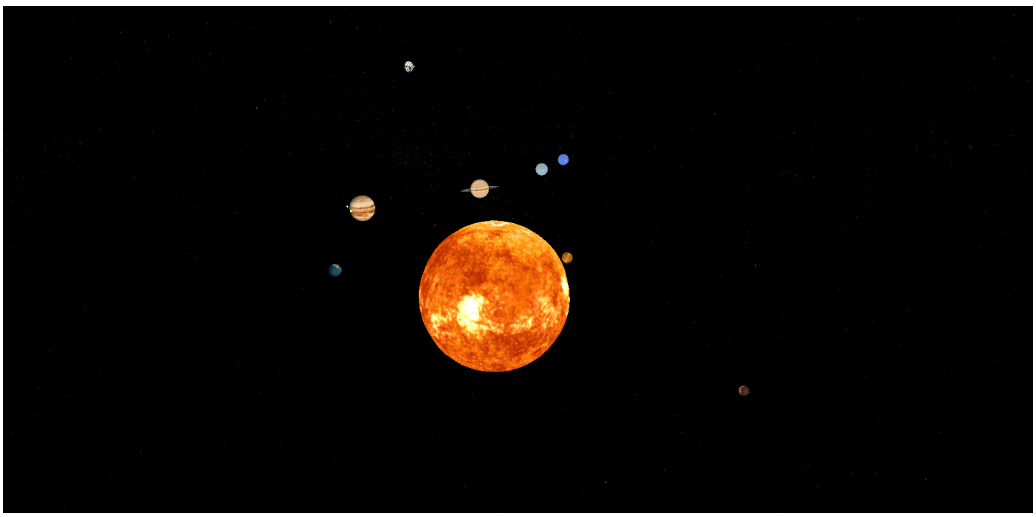

Figura 5: Vista geral do Modelo do Sistema Solar

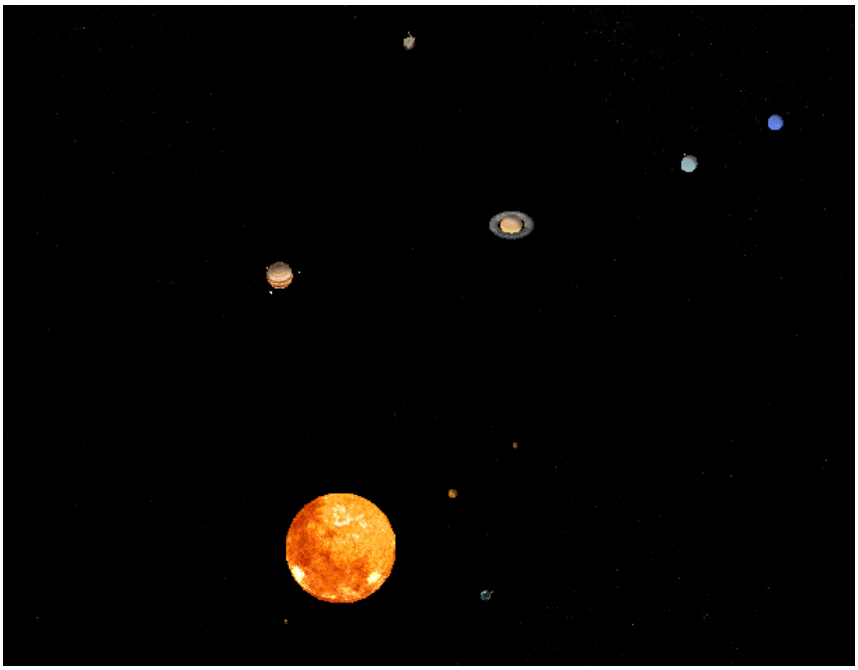

Figura 6: Vista Superior do Modelo do Sistema Solar

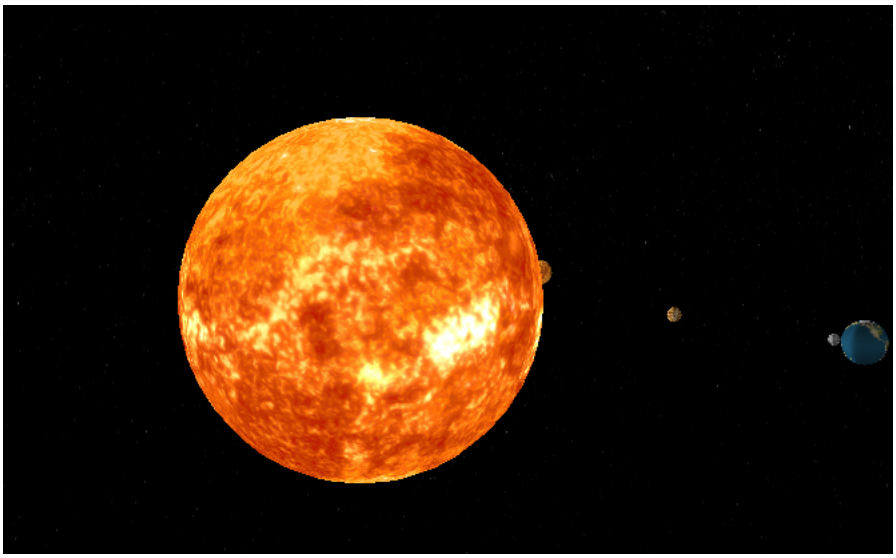

Figura 7: Evidência dos efeitos da iluminação

5 Conclusão

Nesta quarta e última fase do projeto de Computação Gráfica sentimos que conseguimos implementar com sucesso o cenário com gráficos 3D do Sistema Solar que nos foi sugerido. Temos vindo, em cada fase, a aperfeiçoar as funcionalidades do mesmo, pondo em prática os vários conteúdos lecionados na UC, simultaneamente ultrapassando as diversas dificuldades com que nos defrontamos nas diversas fases, obtendo progressivamente agilidade na implementação do que é pedido, podendo em conclusão serem implementadas na nossa aplicação todas funcionalidades requeridas.