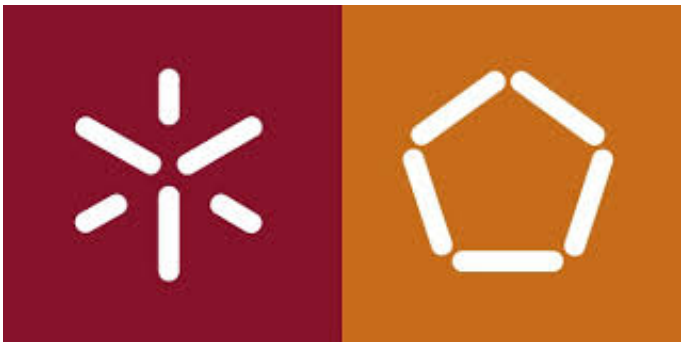


Universidade do Minho - Escola de Engenharia

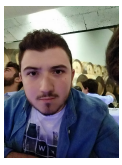
Mestrado Integrado em Engenharia Informática

Computação Gráfica

# Relatório do Projeto Prático - Parte 4

*Autores :*
Diana Costa (A78985)

José Oliveira (A78806)



Marcos Pereira(A79116)

Vitor Castro(A77870)

21 de Maio de 2018

# Conteúdo

# 1 Introdução

Este projeto surge no âmbito da unidade curricular de Computação Gráfica, do Mestrado Integrado em Engenharia Informática da Universidade do Minho. Era requerido, nesta ultima fase, que fossem adaptadas as duas aplicações, o *Generator* e o *Engine*, com o objetivo de processar dados relativos a cores, luzes e texturas para os vários elementos do nosso sistema solar.

O Generator nesta fase deve gerar coordenadas de texturas bem como as normais para cada vértice, na secção 2 é descrita a metodologia utilizada especialmente a base matemática para obter o resultado pretendido.

O Engine deve interpretar as funcionalidades criadas pelo Generator, como a leitura das normas e coordenadas de textura, de maneira a serem apresentadas figuras iluminadas com texturas e cores. Na secção 3 são explicadas as estruturas de dados utilizadas para o problema em questão, bem como as alterações efetuadas ao ficheiro XML e no sistema de leitura do mesmo. A secção termina com uma explicação do método de renderização e do algoritmo utilizado.

# 2 Generator

## 2.1 Normais

A seguir são apresentados os métodos matemáticos utilizados na aplicação Generator para calcular os vetores normais das diferentes figuras presentes no nosso sistema solar.

### 2.1.1 Plane

De acordo com a nossa implementação, a normal para qualquer ponto do plano é (0,1,0).

### 2.1.2 Sphere

No caso da esfera as normais foram obtidas à medida que eram encontrados os vértices que constituem a mesma. Para o ponto mais baixo da esfera, a normal é (0,-1,0) e para o ponto mais alto a normal é (0,1,0). Relativamente aos restantes vértices da esfera, as normais correspondem à normalização da coordenada do respectivo pontos. Como por exemplo, dado um v=(x,y,z) e um raio da esfera r, a normal corresponde a n=(x/r,y/r,z/r).

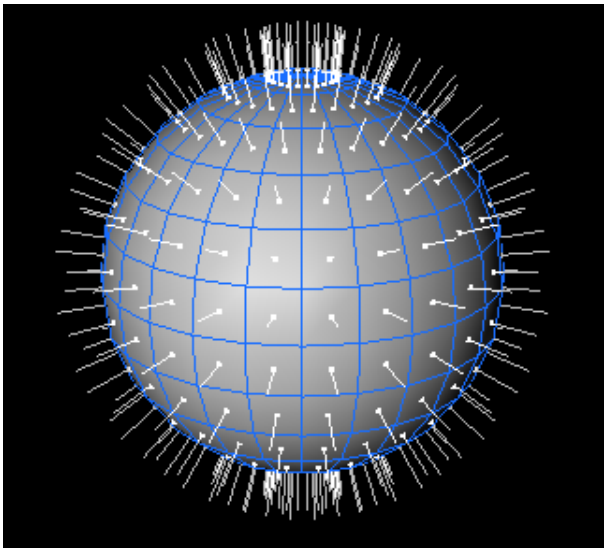


Figura 1: Exemplo das normais ao longo da superfície de uma esfera

### 2.1.3 Teapot

O método usado para calcular as normais dos pontos do teapot foi o seguinte:

- Calcular dois vetores tangentes ao ponto em questão, para isso procedemos ao cálculo das derivadas em ordem a $u$ e a $v$ da função que permite calcular os pontos num patch de Bezier.

  Para isso, foram utilizadas as fórmulas dos polinómios de Bernstein:

$$k_1^{'}(t) = -3(1-t)^2 \tag{1}$$

$$k_2^{'}(t) = 3(1-t)^2 - 6t(1-t) \tag{2}$$

$$k_3^{'}(t) = 6t(1-t) - 3t^2 \tag{3}$$

$$k_4^{'}(t) = 3t^2 \tag{4}$$

- Depois foram utilizadas as seguintes formulas para calcular o vetor tangente a u e a v.

$$\sum_j k_j^{'}(u) \cdot \sum_i p_{ji}.k_i(v) \tag{5}$$

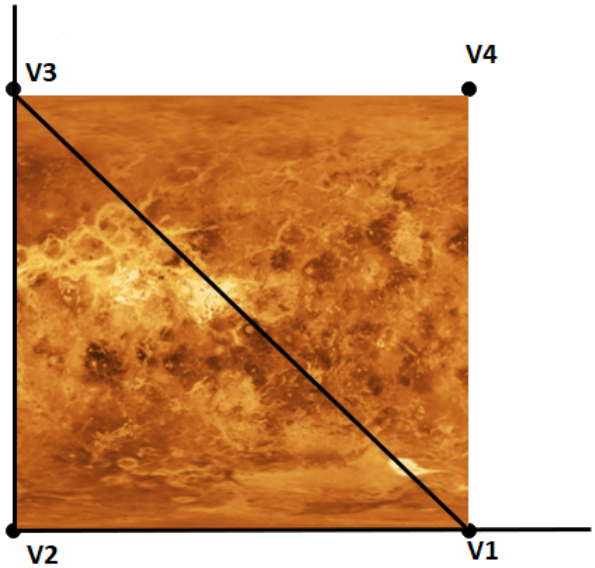
$$\sum_j k_j^{'}(v) \cdot \sum_i p_{ij}.k_i(u) \tag{6}$$

- Depois aplicamos o produto externo e obtemos a normal pertendida.

## 2.2 Coordenadas de textura

### 2.2.1 Plane

O plano tem apenas quatro coordenadas de textura, representadas na seguinte imagem:



### 2.2.2 Sphere

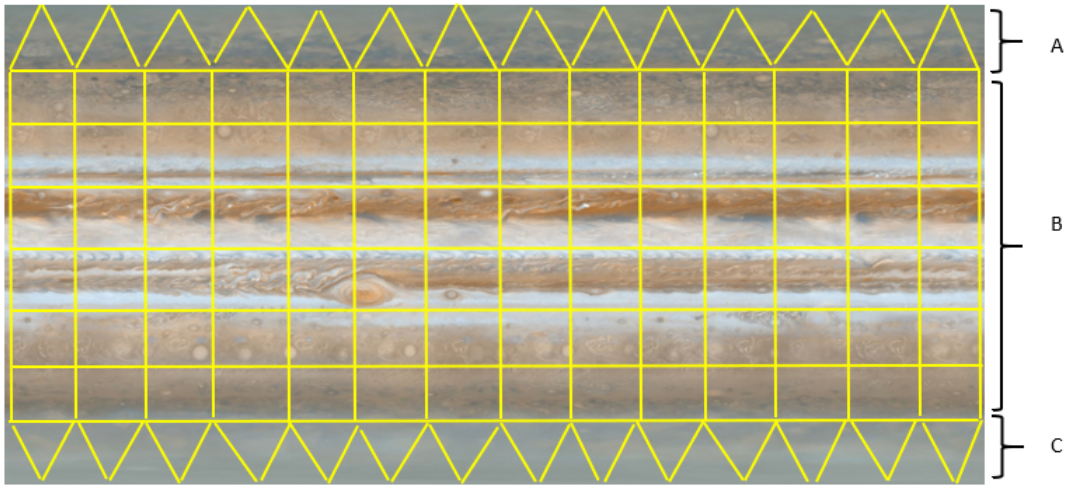
De maneira a encontrar as coordenadas de textura para a esfera desenvolvemos um esboço de uma textura, a qual dividimos no número de slices e stacks pretendidas, optando por dividir em triângulos as zonas próximas dos dois polos da esfera para garantir o menor erro possível.

É fácil verificar que a superfície intermédia é dividida em retângulos, e ,portanto, basta percorrer uma stack e por cada slice registar a coordenada de textura.

A superfície próxima dos polos como é dividida em triângulos requer outra abordagem. Considerando estas porções da superfície, a primeira stack e última de cada slice, calcula-se o ponto médio a fim de obter um triângulo.

De seguida foi calculada a sequência correta das coordenadas de textura para calcular os diversos triângulos.

Abaixo está ilustrada uma imagem de uma textura com o método explicado anteriormente.




A – Superfície superior

B – Superfície intermédia

C – Superfície inferior

### 2.2.3 Teapot

Para obter as coordenadas de textura para um determinado ponto do teapot, usam-se os valores u e v desse ponto. Estes valores são as coordenadas na imagem da textura (x,y) que serão utilizadas para o ponto referido.

O seguinte código representa a geração das coordenadas de textura e das normais, bem como os pontos do teapot e é referente à função *makeBezier()* presente na aplicaçâo Generator.

Após ser feita a geraçâo dos pontos, são escritos para o ficheiro do modelo.

```
  for (int patch_number = 0; patch_number < 32; patch_number++) {
  for (int tesselation_v = 0; tesselation_v < tesselationLevel; tesselation_v++)
      {
    float v = (float)tesselation_v / tesselationLevel;
    for (int tesselation_u = 0; tesselation_u < tesselationLevel; tesselation_u
        ++) {
      float u = (float)tesselation_u / tesselationLevel;
      // first triangle
      points.push_back(bezierPoint(patch_number, u, v));
      points.push_back(bezierPoint(patch_number, (u + (1.0f / tesselationLevel)),
          (v + (1.0f / tesselationLevel))));
      points.push_back(bezierPoint(patch_number, (u + (1.0f / tesselationLevel)),
          v));

      // second triangle
      points.push_back(bezierPoint(patch_number, (u + (1.0f / tesselationLevel)),
          (v + (1.0f / tesselationLevel))));
      points.push_back(bezierPoint(patch_number, u, v));
      points.push_back(bezierPoint(patch_number, u, (v + (1.0f / tesselationLevel
          ))));

      // first normal point
      normal.push_back(bezierPointNormal(patch_number, u, v));
      normal.push_back(bezierPointNormal(patch_number, (u + (1.0f /
          tesselationLevel)), (v + (1.0f / tesselationLevel))));
      normal.push_back(bezierPointNormal(patch_number, (u + (1.0f /
          tesselationLevel)), v));

      // second normal point
      normal.push_back(bezierPointNormal(patch_number, (u + (1.0f /
          tesselationLevel)), (v + (1.0f / tesselationLevel))));
      normal.push_back(bezierPointNormal(patch_number, u, v));
      normal.push_back(bezierPointNormal(patch_number, u, (v + (1.0f /
          tesselationLevel))));

      // first texture Points
      texture_points.push_back(TexturePoint(u, v));
      texture_points.push_back(TexturePoint(u + (1.0f / tesselationLevel), (v +
          (1.0f / tesselationLevel))));
      texture_points.push_back(TexturePoint(u + (1.0f / tesselationLevel), v));

      // second texture point
      texture_points.push_back(TexturePoint(u + (1.0f / tesselationLevel), (v +
          (1.0f / tesselationLevel))));
      texture_points.push_back(TexturePoint(u, v));
      texture_points.push_back(TexturePoint(u, (v + (1.0f / tesselationLevel))));
    }
  }
}
```

# 3 Engine

## 3.1 Estruturas de dados

Nesta última fase do trabalho prático a aplicação *Engine* deve ser capaz de processar informações relativas a *luzes*, *texturas* e *cores*. Para tal, foi necessário criar algumas estruturas de dados, bem como adaptar as já existentes.

Relativamente à class *Light*: a string *type* indica o tipo de luz (pointlight, directional light, spotlight), no array *pos* temos a posição no referencial, *color* representa a cor da luz. Os floats *cons*, *quad*, *linear* caracterizam a atenuação de uma luz pointlight, e *spotCutOff*, **spotDirection*, *spotExponent* caracterizam uma luz spotlight. Por fim, o int *id* indica o número da luz.

```
struct Light {
    string type;
    float pos[4];
    Color color;
    float cons = NULL,
          quad = NULL,
          linear = NULL,
          spotCutOff = NULL,
          *spotDirection = NULL,
          spotExponent = NULL;
    unsigned int id;
};
```

Na class *Color*: Os floats **diffuseColor*, **specularColor*, **emissiveColor*, **ambientColor* representam os quatro componentes que definem uma cor.

```
struct Color {
    bool vazio = true;
    float *diffuseColor = NULL,
        *specularColor = NULL,
        *emissiveColor = NULL,
        *ambientColor = NULL;
};
```

Relativamente à class *Figure* para além das variáveis que já existiam das outras fases, foram adicionadas *textureIndexVBO*, *normalIndex* *textureIndex* que representam o índice VBO, e o índice da normal e da textura.

```
struct Figure {
    string name;
    int vboIndex = -1, normalIndex, textureIndex, textureIndexVBO = -1, triangles = 0;
    Color color;
    Scale scale;
    Rotation rotate;
    Translate translate;
};
```

## 3.2 Ficheiro XML

De acordo com o que foi pedido para a esta última fase do projeto, foi necessário alterar a estrutura do ficheiro XML. Deste modo, relativamente ao que tinhamos feito da outra fase, foi adicionada uma tag *<lights>* que tem como filho *<light>* onde estão presentes um conjuto de atributos que caracterizam uma luz e que serão abordados em mais detalhe de seguida. Para além desta tag, em <model> é agora possível acrescentar atributos que permitem caracterizar a textura ou a cor de uma dada figura.

```
<scene>
  <lights>
    <light type="POINT" posX="0" posY="0" posZ="0" diffR="10" diffG="10" diffB="10"
    ambR="0.2" ambG="0.2" ambB="0.2"/>
  </lights>
  <group a="Sol">
    <group>
      <rotate time="1" axisX="0" axisY="1" axisZ="0"/>
      <scale X="1" Y="1" Z="1"/>
      <scale X="1" Y="1" Z="1"/>
      <models>
        <model file="sun.3d" texture="sun.jpg"/>
      </models>
      .....
```

## 3.3 Leitura do ficheiro XML

O algoritmo de leitura do ficheiro XML é idêntico ao utilizado nas fases anteriores tendo por base a iteração recursiva dos *groups* e das restantes tags subjacentes. De acordo com as alterações no ficheiro XML referidas acima, passa-se a explicar o funcionamento do sistema de leitura:

- Caso seja encontrada uma tag *<lights>*, ativam-se as luzes através do glEnable(GL_LIGHTING) e itera-se sobre as *<light>* existentes:
  - É ativada a luz a que vai corresponder, de acordo com a valor da variável global usada para registar o número total de luzes;
  - É criada uma instância do objeto *Light*;
  - Atualiza o campo *type* com o tipo da luz (POINT, DIRECTIONAL ou SPOT);
  - Atualiza o campo *pos* com a posição;
  - Caso tenha cor, atualizam-se os diferentes componentes da cor;
  - Caso tenha caracteristicas de uma luz, atualizam-se os respetivos campos;
  - Por fim, adiciona-se a luz processada à arvore final;
- Caso seja encontrada uma tag *<models>*, para além de guardar o nome do .3d relativo ao atributo *file* para posteriormente fazer load da figura correspondente (tal como nas fases anteriores), verificamos se:
  - Existe um atributo *texture*:
    1. Verifica-se se a textura em causa se encontrada carregada, para tal procuramos no map texturesIndex se contém o nome da textura. Caso não se verifique é chamada a função *getTexture()*. Aqui é aberto o ficheiro passado como argumento e é carregada a textura, sendo retornado um id para atualização do map anteriormente mencionado.

2. É feita a leitura do ficheiro .3d de modo a obter as coordenadas da textura bem como as normais. Primeiro verificamos se existe um VBO para a respetiva textura ou normal e caso isso se verifique adicionamos o seu valor ao nodo da arvore final. Caso isso não se verifique é usada a função *loadFigura()* que recebe como argumentos os arrays a preencher, sendo depois gerados os índices VBOs através de glGenBuffer e glBindBuffer. Por fim, são atualizados os maps com o nome da figura e os valores anteriormente obtidos.

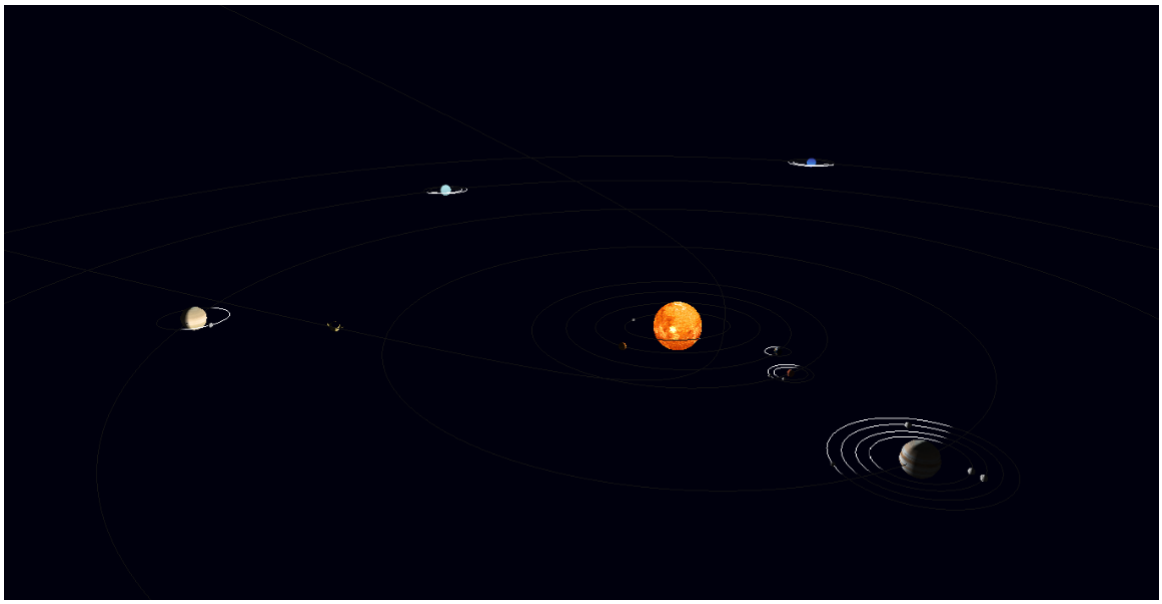
– Ainda dentro da tag <models>, é verificado se existe informação relativa a cores atribuidas a uma determinada figura, para tal procura-se atributos que são referências aos componentes de uma cor. Caso se verifique é atualizada o respetivo campo da estrutura da figura, se não é especificada nenhuma cor são introduzidos valores por defeito ficando os componentes *specular* e *emissive* a branco e *diffuse* e *ambient* a preto.

## 3.4 Renderização

De maneira a fazer o *rendering* do sistema solar, foi usada uma pesquisa em profundidade da árvore criada à semelhança do que foi feito em fases anteriores. O processo inicia-se quando a função *renderScene* é chamada na main, que por sua vez vai chamar a função do *drawScene(Tree tree)* presente no engine. O algoritmo produzido na fase anterior foi alterado de maneira a processar os novos parâmetros das estruturas, deste modo o que concerne a luzes, cores e texturas é descrito de seguida:

- Verifica-se se a figura atual tem cor, em caso positivo usa-se glMaterialfv() para a representar;
- Verifica-se se o nodo atual tem luzes, em caso positivo itera-se sobre o vetor que contém as luzes e através de glLightfv() aplica-se todas componentes necessárias;
- Por fim, todas as figuras, texturas e normais são representadas.

# 4 Scene



# 5 Debug

De maneira a verificar os resultados obtidos, foram adicionadas algumas funcionalidades à interface do programa. A tecla F1 foi definida para renderizar apenas as arestas das figuras, enquanto que a F2 define o modo de renderização que inclui as faces dos modelos. As teclas W, A, S e D rodam os modelos, enquanto que as setas movem-nos no espaço.

Com a ajuda destas funcionalidades foram descobertos alguns problemas ao longo do desenvolvimento do programa, que por terem sido identificados a tempo foram rapidamente resolvidos.

# 6 Conclusão

Esta útilma fase do projeto prático foi bastante importante e enriquecedora, pois permitiu aos membros do grupo perceber e interiorizar melhor os conceitos abordados ao longo das últimas aulas práticas da Unidade Curricular de Computação Gráfica. Deste modo, foram postos em prática conhecimentos relativos à utilização de texturas e cores aplicadas a diversas figuras geométricas, bem como a produção da iluminação em objetos.

No que diz respeito ao desenvolvimento da aplicação Generator, a principal dificuldade encontrada foi na geração das normais e das coordenadas de textura visto que foi necessário implementar conceitos matemáticos e geométricos para obter o output pretendido. Ao nível da aplicação Engine o grupo achou que foram acessiveis as alterações efetuadas de maneira a processar as novas informações apresentadas como input, à semelhança do que foi feito em outras fases.

Em suma, é feita uma apreciação positiva relativamente ao trabalho realizado, visto que a implementação de todas as funcionalidades propostas foram conseguidas com sucesso. O grupo conseguiu tirar partido dos conhecimentos adquiridos neste projeto, sentido-se capaz de, num contexto futuro, aplicar os conceitos subjacentes de forma eficaz.