Normals and Texture Coordinates

Relatório

Trabalho elaborado por:
Carlos Pedrosa - a77320
David Sousa - a78938
Manuel Sousa - a78869

Universidade do Minho

Mestrado integrado em Engenharia Informática

Unidade Curricular de Computação Gráfica

Índice

Índice de ilustrações

Introdução

No âmbito da unidade curricular de Computação Gráfica foi-nos proposto o desenvolvimento de um mini cenário baseado em gráficos 3D. Assim, o projeto encontra-se dividido em 4 fases. Neste relatório pretendemos demonstrar todos os passos que foram efetuados no sentido a cumprir a quarta tarefa proposta.

Nesse sentido, a quarta fase encontra-se dividida em duas partes. A primeira prende-se com a inserção de texturas no modelo até então desenvolvido. A segunda, por sua vez, pressupõe a introdução de luz no sistema. Assim, nesta fase o foco passa por alterar o *parser* e consequentemente o ficheiro *XML*, de modo a, tornar ainda mais realista o sistema solar desenvolvido.

Fase 4

Desenho do sistema solar com luz e texturas

1. Alteração do parser

Nesta última fase do projeto, era necessário que a partir de uma configuração XML fosse possível extrair um ficheiro que diria respeito a uma textura aplicada, mais tarde, a um determinado objeto. Para além disso, era também pedido para extrair componentes relacionados com a iluminação e coloração dos diversos objetos (light e color). Nesse sentido, resolvemos incluir esses atributos na nossa configuração XML.
Para ser possível ler uma determinada textura, acrescentamos um atributo no campo "model" nomeado de "texture" sendo que este é seguido pelo nome do ficheiro onde se encontra a textura.

```
<models>
    <model file="sphere.3d" texture="texture_sun.jpg" /> <!-- Sun -->
</models>
```

Figura 1 - Exemplificação da textura no parser

Ainda no campo "model", e relacionado com a coloração dos objetos, os atributos seguintes poderiam variar conforme a coloração pretendida. As componentes de cor fornecidas pelo openGL são: "DIFFUSE", "SPECULAR", "EMISSIVE" e "AMBIENT". Desta forma os atributos seguintes tomariam o nome das iniciais de cada componente seguido de um caracter - "R", "G" ou "B".

```
<models>
    <model file="sphere.3d" texture="texture_sun.jpg" emisR="1" emisG="1" emisB="1" /> <!-- Sun -->
</models>
```

Figura 2 - Exemplificação da emissividade no parser

Para ser possível ler componentes relacionados com a luz, resolvemos criar um novo campo denominado de "lights" que conteria dentro de si informações sobre como a luz iria ser processada no sistema. Dentro deste campo é possível ter diversos tipos de luz. Os atributos usados são: "type" - que corresponde, então, ao tipo desta. Este atributo poderá tomar o valor de "POINT", "DIRECTIONAL" e "SPOTLIGHT", que são tipos de luz fornecidos pelo openGL. Para além disso, deverá ser ainda especificado o ponto onde a luz será aplicada, usando-se para tal os atributos, “posX”, “posY” e “posZ”.

```
<lights>
    <light type="POINT" posX="0.0" posY="0.0" posZ="30.0"/>
</lights>
```

Figura 3 - Exemplificação da luz no parser

2. Aplicação de textura

Os cálculos das coordenadas das texturas (aplicadas à esfera), foram efetuados no *generator*, pelo que este foi um pouco alterado. O processo baseou-se em calcular de quanto em quanto é que uma textura teria de ser aplicada. O espaço das texturas varia entre 0 e 1 em cada eixo. Neste caso, os eixos que irão sofrer alterações serão o eixo do X e do Y. Desta forma, obtemos o intervalo pretendido dividindo 1 pelo número de *slices* da esfera (X), e 1 pelo número de *stacks* da esfera (Y), permitindo assim o obter quais as coordenadas que deveriam ser aplicadas.

```
// Textures
double s;
double deltaS = 1.0f / slices;
double t;
double deltaT = 1.0f / stacks;
```

Figura 4 - Obtenção das coordenadas das texturas

Com o auxílio dos índices do ciclo que formam as coordenadas da esfera, as coordenadas das texturas foram usando duas variáveis:

- ➔ s = j * deltaS;
- ➔ t = (stacks - i) * deltaT;

Por fim, as coordenadas calculadas iam sendo escritas para o ficheiro .3d (neste caso sphere.3d). De notar que o ficheiro foi alterado, visto que era necessário a inclusão destas novas coordenadas. Na primeira posição era escrito um vértice da esfera, na segunda um vértice que representa as normais (usado para a iluminação - referida mais adiante), e na terceira posição um dos vértices da textura a ser aplicada.

```
file << p1x_S << " " << p1y_S << " " << p1z_S << "\n";
file << sin(beta) * sin(alpha) << " " << cos(beta) << " " << sin(beta) * cos(alpha) << "\n";
file << s << " " << t << " " << -1 << "\n";

file << p2x_S << " " << p2y_S << " " << p2z_S << "\n";
file << sin(beta + deltaBeta) * sin(alpha) << " " << cos(beta + deltaBeta) << " " << sin(beta + deltaBeta) * cos(alpha) << "\n";
file << s << " " << (t - deltaT) << " " << -1 << "\n";
```

Figura 5 - Escrita das coordenadas para os ficheiros

A aplicação das texturas de cada objeto (depois de devidamente extraídas do ficheiro XML), foi feita com o auxilio da livraria DevIL. O processo passava por carregar uma certa textura, gerando assim um "id" o qual identificaria mais tarde a respetiva textura. Posto isto, através da função fornecida pelo openGL (*glBindTexture*), aplicamos a textura ao desenhar o VBO do objeto correspondente.

```
glBindTexture(GL_TEXTURE_2D, texIds[i]);

glDrawArrays(GL_TRIANGLES, (i * fo->totalVertexes), fo->totalVertexes);

glBindTexture(GL_TEXTURE_2D, 0);
```

Figura 6 - Funções openGL para a escrita das texturas

3. Aplicação de luz

Para a aplicação de luz no sistema, foi necessário o cálculo dos vetores das normais. Uma normal é um vetor unitário cuja direção é perpendicular a uma superfície num determinado ponto.
Desta forma, foi necessário determinar o ângulo entre a normal num determinado ponto da superfície (neste caso, apenas aplicamos as normais à esfera). Com o cálculo dos diversos vértices da esfera, o cálculo das normais foi bastante intuitivo, bastando apenas extrair o ângulo usado para o cálculo de cada vértice do objeto.

```
p1x_S = radius * sin(beta) * sin(alpha);
p1y_S = radius * cos(beta);
p1z_S = radius * sin(beta) * cos(alpha);
```

Ângulos usados nas normais

Figura 7 - Coordenadas dos vértices da esfera

De forma a guardar a informação que dizia respeito a luz presente na configuração XML, foi criada uma estrutura de dados auxiliar, "Light". Esta estrutura guarda qual a fonte da luz, bem como os campos "X", "Y" e "Z" da mesma.

```
typedef struct light {
    char* source;
    double x, y, z;
} Light;
```

Figura 8 - Estrutura light

Deste modo, a luz é processada no início da função ***renderScene()***, logo após o posicionamento da câmara. Este processo é feito através da função ***processLight()***, a qual avalia o tipo de luz a ser processado, e aplicando as diversas operações openGL necessárias.

```
void processLight()
{
    if (strcmp(lt->source, "POINT") == 0) {
        pos[0] = static_cast<GLfloat>(lt->x);
        pos[1] = static_cast<GLfloat>(lt->y);
        pos[2]= static_cast<GLfloat>(lt->z);
        pos[3] = 1;

        glEnable(GL_LIGHT0);
        glLightfv(GL_LIGHT0, GL_POSITION, pos);
        glLightfv(GL_LIGHT0, GL_POSITION, amb);
        glLightfv(GL_LIGHT0, GL_POSITION, diff);
    } else if (strcmp(lt->source, "DIRECTIONAL") == 0) {
        pos[0] = static_cast<GLfloat>(lt->x);
        pos[1] = static_cast<GLfloat>(lt->y);
        pos[2]= static_cast<GLfloat>(lt->z);
        pos[3] = 0;

        glLightfv(GL_LIGHT0, GL_POSITION, pos);
        glLightfv(GL_LIGHT0, GL_POSITION, amb);
        glLightfv(GL_LIGHT0, GL_POSITION, diff);
    } else if (strcmp(lt->source, "SPOTLIGHT") == 0) {
        spotDir[0] = static_cast<GLfloat>(lt->x);
        spotDir[1] = static_cast<GLfloat>(lt->y);
        spotDir[2] = static_cast<GLfloat>(lt->z);

        pos[0] = static_cast<GLfloat>(lt->x);
        pos[1] = static_cast<GLfloat>(lt->y);
        pos[2]= static_cast<GLfloat>(lt->z);
        pos[3] = 1;

        glLightfv(GL_LIGHT0, GL_POSITION, pos);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, diff);
        glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spotDir);
        glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
        glLightf(GL_LIGHT0,GL_SPOT_EXPONENT, 0.0);
    }
```

Figura 9 - Função que processa a luz

Depois de processada a luz, o vetor das normais vai ser desenhado, por forma a aplicar a luz nos diversos pontos de todos os objetos.

4. Aplicação da cor

Para a coloração, foi também usada uma estrutura de dados auxiliar a qual continha a coloração a ser aplicada a um determinado ficheiro ".3d". Esta estrutura foi denominada de “Color", e é formada pelo nome da componente da cor, bem como os campos "R", "G" e "B" da mesma.

```
typedef struct colors {
    char* component;
    double r, g, b;
} Colors;
```

Figura 10 - Estrutura colors

A aplicação desta coloração era feita logo após o processamento das diversas transformações referentes a um objeto, através da função ***processColors(Colors* col)***. Esta função identifica qual o tipo de componente pelo que trata de aplicar a operação referente a essa mesma componente.

```
void processColors(Colors* col)
{
    if (strcmp(col->component, "diffuse") == 0) {
        GLfloat diff[4] = { static_cast<GLfloat>(col->r),
                            static_cast<GLfloat>(col->g),
                            static_cast<GLfloat>(col->b),
                            1 };

        glMaterialfv(GL_FRONT, GL_DIFFUSE, diff);
    }

    if (strcmp(col->component, "specular") == 0) {
        GLfloat spec[4] = { static_cast<GLfloat>(col->r),
                            static_cast<GLfloat>(col->g),
                            static_cast<GLfloat>(col->b),
                            1 };

        glMaterialfv(GL_FRONT, GL_SPECULAR, spec);
    }

    if (strcmp(col->component, "emissive") == 0) {
        GLfloat emis[4] = { static_cast<GLfloat>(col->r),
                            static_cast<GLfloat>(col->g),
                            static_cast<GLfloat>(col->b),
                            1 };

        glMaterialfv(GL_FRONT, GL_EMISSION, emis);
    }

    if (strcmp(col->component, "ambient") == 0) {
        GLfloat ambi[4] = { static_cast<GLfloat>(col->r),
                            static_cast<GLfloat>(col->g),
                            static_cast<GLfloat>(col->b),
                            1 };

        glMaterialfv(GL_FRONT, GL_AMBIENT, ambi);
    }
}
```

Figura 11 - Função que processa a cor

5. Desenho do sistema solar

Apenas alteramos a configuração *XML* para que as texturas e a luz sejam aplicadas ao sistema solar. Deste modo, obtemos a seguinte configuração:

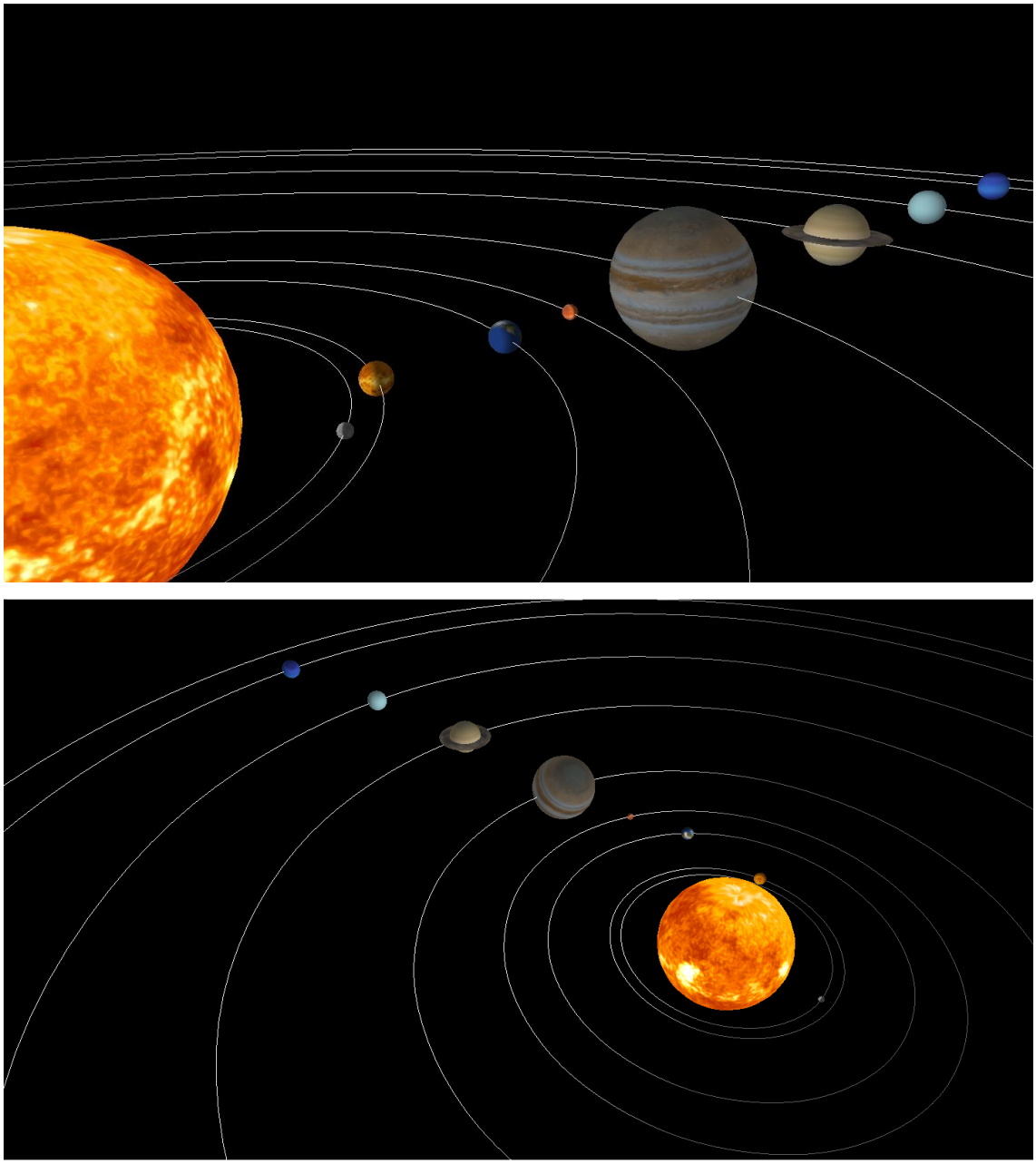

Figura 12 - Sistema solar obtido

Conclusão

De acordo com o principal objetivo definido na quarta fase consideramos que este foi alcançado. De facto, o desenho e a representação, de um sistema solar foi conseguido, apesar de algumas dificuldades em relação a algumas componentes, nomeadamente a inclusão de luz no sistema. Apesar da luz ser visível no sistema, a aplicação desta não foi totalmente percetível, o que tornou o processo bastante complicado. Contudo, a aplicação das texturas (o que tornará o sistema mais agradável) foram feitas com sucesso.

Assim, a presente fase do trabalho permitiu obter um conhecimento mais prático relativamente ao modo como as texturas e a luz eram aplicadas. O principal desafio com que nos deparamos (para além da aplicação da luz) foi numa fase inicial na perceção do enunciado relacionado com a forma de como poderia ser escrito o XML, como também a forma de como os novos atributos se podiam relacionar entre si. No entanto, depois de ultrapassados todo o projeto surgiu naturalmente.

Em suma, o presente trabalho tornou-nos mais objetivos e racionais sendo que foi uma mais valia para todos os elementos do grupo.

Bibliografia

- Notes for an Undergraduate Course in Computer Graphics, university of minho, António Ramires.