Departamento de Informática
Mestrado Integrado em Engenharia Informática

Computação Gráfica
Phase 4 – Normals and Texture Coordinates

Grupo 38

Diogo Tavares – A61044
Pedro Lima – A61061
Gil Gonçalves – A67738
Judson Paiva – E6846

Índice

Introdução

Nesta quarta e última fase do projeto proposto no âmbito da unidade curricular de Computação Gráfica damos seguimento ao trabalho desenvolvido nas fases anteriores. Nesta fase pretende-se estender a aplicação de modo a que o gerador de primitivas gere, para além das coordenadas de cada vértice, as normais e coordenadas de textura. Relativamente ao motor 3D, este vai ter de ser capaz de ativar as funcionalidades de iluminação e texturização, assim como possibilitar a pormenorização de cores e texturas no ficheiro XML.
O sistema solar dinâmico, o sol, os planetas e as respetivas luas vão possuir já uma textura. Sendo a única fonte de luz no sistema, o sol terá também associado um ponto de luminosidade.

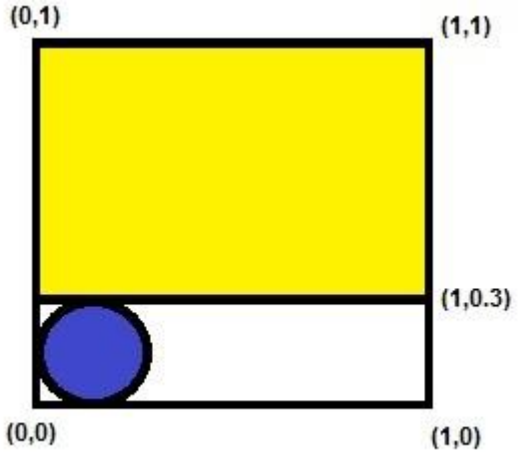
Atualização do Motor 3D e do Gerador

Gerador

No gerador começou logo por sofrer alterações no sentido de gerar não só as normais aos pontos assim como as coordenadas de textura.

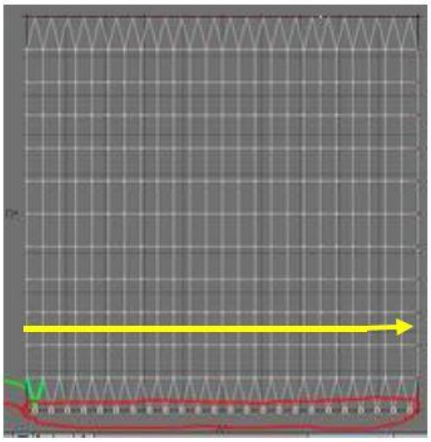
Aplicação de Texturas

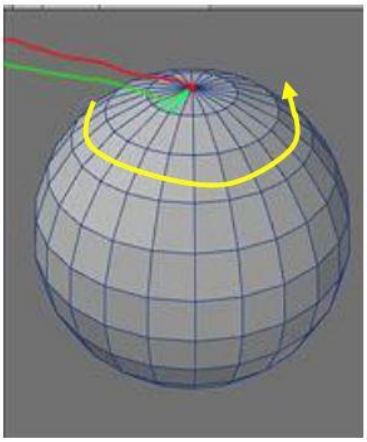
Cone



Cubo

Esfera





Ficheiros .3D

```
2.5,0,2.5
2.5,0,-2.5
-2.5,0,-2.5                    Vértices
-2.5,0,2.5
2.5,0,2.5
-2.5,0,-2.5
@
0,1,0
0,1,0
0,1,0                          Normais
0,1,0
0,1,0
0,1,0
@
0,0
0,1
1,0
0,1                            Texturas
1,1
1,0
```

Engine

A nível de alterações, o Engine foi mais modificado as novas funcionalidades do que o gerador. A começar pelo *parsing* que engloba mais dois termos, pois é necessário ler as normas e texturadas do .3d.

Ficheiro XML de input

De seguida, expomos um breve excerto de um ficheiro XML responsável pela estruturação da cena de desenho. Trata-se do já conhecido Sistema solar dinâmico, agora com luzes, sombras e texturas.

```
<scene name="Sistema Solar" >
    <luz>
    <light tipo="POINT" posX="0.0" posY="0.0" posZ="0.0"/>
</luz>
        <group name="Sun" >
            <scale X="1" Y="1" Z="1" />

            <models>
                <model file="sphere.3d" texture="sol.jpg"/>
            </models>
```

Figura 1 Excerto do xml

O ficheiro XML evoluiu bastante ao longo das fases, passando de um simples ficheiro com primitivas geométricas, a um ficheiro complexo onde existem transformações geométricas, curvas, texturas e luzes.

Classe Luz

Com a introdução da luz foi necessário criar uma nova classe para lidar com a mesma. A classe luz tem como objetivo guardar, neste caso, o único ponto de luz existente: o sol.

```
class Luz {
    string tipo;
    float posX;
    float posY;
    float posZ;
public:
    Luz();
    Luz(float x, float y, float z, string t);
    float getPosX() { return posX; }
    float getPosY() { return posY; }
    float getPosZ() { return posZ; }
    string getTipo() { return tipo; }
    void setPosX(float x) { posX = x; }
    void setPosY(float y) { posY = y; }
    void setPosZ(float z) { posZ = z; }
    void setTipo(string t) { tipo = t; }
    virtual ~Luz(void);
};
```

Figura 2 Classe luz .h

Note que esta classe também admite os outros tipos de luzes dependendo do valor do campo “tipo”. As coordenadas posX, posY, posZ correspondem à localização da fonte de luz na cena.

No nosso caso especifico será o ponto (0,0,0). Acontece que como o sol emite luz em todas as direções, o seu tipo será de “ponto de luz”.

```
XMLElement* luzes =root->FirstChildElement("luz");
if (luzes != NULL) {
    XMLElement* l = luzes->FirstChildElement("light");
    string tipo = "POINT";
    float x = 0, y = 0, z = 0;
    if (l->Attribute("tipo")) { tipo = l->Attribute("tipo"); }
    if (l->Attribute("posX")) { x = stof(l->Attribute("posX")); }
    if (l->Attribute("posY")) { y = stof(l->Attribute("posY")); }
    if (l->Attribute("posZ")) { z = stof(l->Attribute("posZ")); }

    luz = Luz::Luz(x, y, z, tipo);
}
else {
    luz = Luz::Luz(0, 0, 0, "POINT");
}
```

Figura 3 parsing da Luz

Classe Model

A classe que mais evolui foi a Model, de maneira a suportar todos os atributos que lhe podem ser atribuídos.

```
class Model {
    string nome;
    Transformation transf;
    GLuint buffer[3];
    int nv;
    vector<Model> descendentes;
    string imagem;
    unsigned int imag;
    unsigned int textID;
    unsigned char *imageData;
    float diffR, diffG, diffB;
public:
    Model(string);
    vector<Point> pontos;
    vector<Point> normas;
    vector<Point> texturas;
    string getNomeModelo() { return nome; }
    vector<Point> getPontos() { return pontos; }
    vector<Point> getNormas() { return normas; }
    vector<Point> getTexturas() { return texturas; }
    Transformation getTransformacao() { return transf; }
    unsigned int getTextID() { return textID; }
    float getdiffR() { return diffR; }
    float getdiffG() { return diffG; }
    float getdiffB() { return diffB; }
    void setTransformacao(Transformation t) { transf = t; }
    void Model::setDiffs(float dr, float dg, float db);
```

Figura 4 excerto do Model.h

A classe modelo passa agora a contemplar:

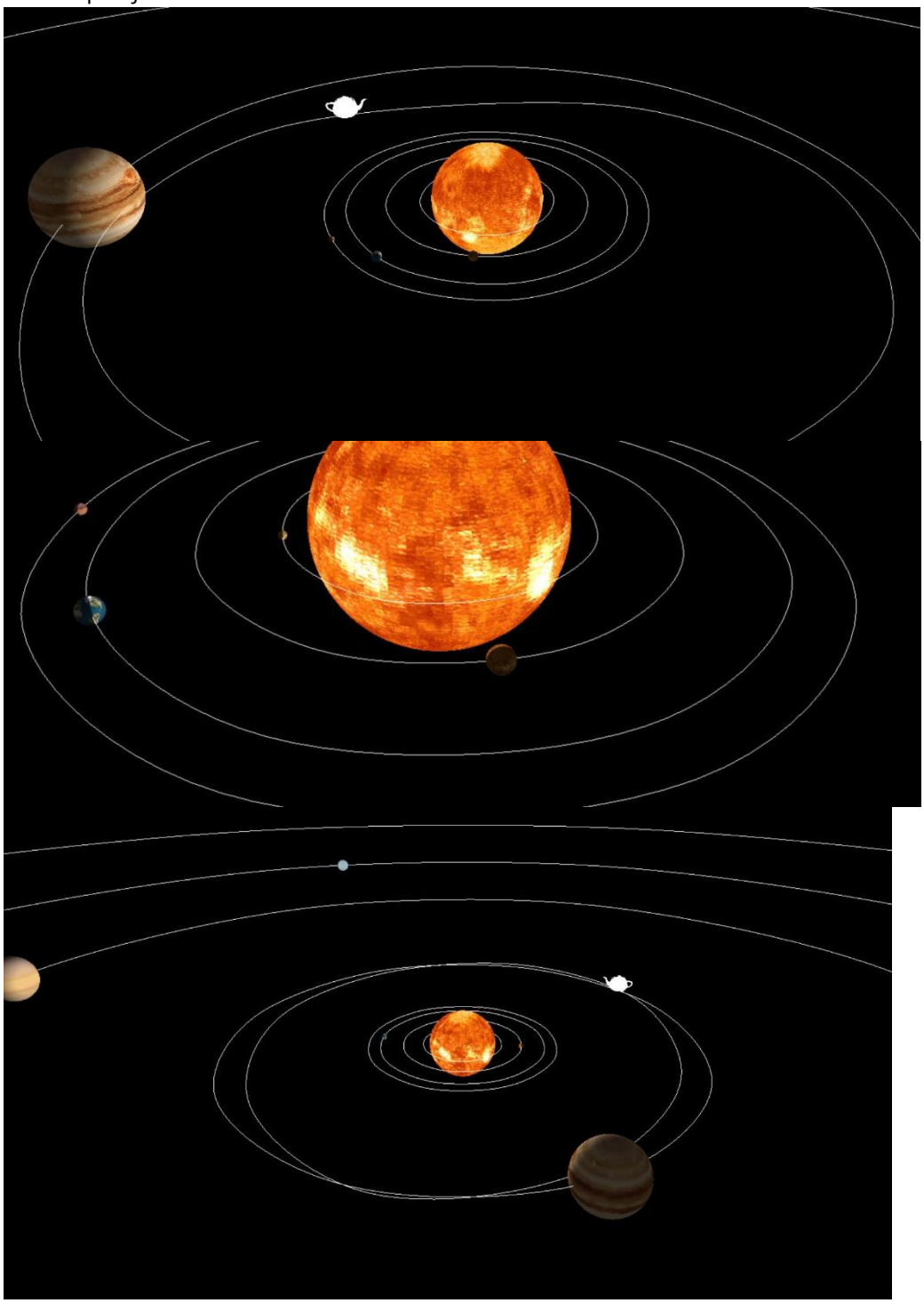
1. Três *buffers* para as VBOs (triângulos, normais e texturas).
2. O número de vértices. Na leitura das VBOs é fundamental saber o número de vértices de cada modelo. Ora, como é gerada normal para cada vértice, a leitura das VBOs de vértices e das normais é feita de três em três coordenadas. Contudo as texturas são lidas de dois em dois, devido às coordenadas cartesianas a duas dimensões.
3. O *nome* da imagem de textura. 4.
4. O código da difusão de cor.
5. Variáveis da textura e da própria textura já carregada
6. Os vetores que contêm os pontos dos vértices, normais e texturas.

```
vector<Point> pontos;
vector<Point> normas;
vector<Point> texturas;
```

Figura 5 novas estruturas

Resultados obtidos

Nesta secção exibimos os resultados obtidos na conclusão da última fase deste projeto.



No entanto ocorreram erros inesperados de ultima hora, o que causou algumas falhas nomeadamente nas luas.

Conclusão

No final desta quarta e última fase do trabalho prático consideramos os objetivos fundamentais foram abrangidos com sucesso. O sistema solar dinâmico encontra-se, finalmente, completo.
A parte referente ao gerador exigiu algum raciocino, nomeadamente na parte do cálculo das normais dos vértices e no mapeamento de texturas, que obrigou a realizar uma conversão de coordenadas tridimensionais para coordenadas bidimensionais.
Contudo, as maiores dificuldades nesta fase consistiram na adaptação da classe modelo às novas funcionalidades do motor 3D. A classe modelo sofreu bastantes alterações de modo a sustentar estas novas funcionalidades de luminosidade e de texturas. Para além disso, o *parsing* passou a ler mais informação dos ficheiros XML porque estes passaram a ter novos campos como as texturas e as luzes. Não querendo deixar de referir, o motor 3D não admite que um filho possa ter filhos. Como se pode verificar nos ficheiros de *input* não existe nenhum caso onde haja mais do que filhos de modelos que já são filhos.
Em suma, estamos bastante satisfeitos com o trabalho desenvolvido uma vez que já é possível observar algo que já era aguardado há algum tempo.