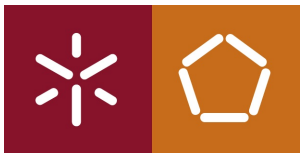

Universidade do Minho

Computação Gráfia

2018/2019

Phase 4 – Normals and Texture Coordinates

Trabalho realizado por:	*Número de Aluno:*
Nelson Gonçalves	A78713
João Aloísio	A77953
César Henriques	A64321
Ricardo Ponte	A79097

15 de Maio de 2019

Conteúdo

1 Introdução **2**

2 Descrição do Problema **3**

3 Resolução do Problema **4**
3.1 Gerador 4
3.1.1 Plane 4
3.1.2 Box 5
3.1.3 Sphere 7
3.1.4 *Bezier Patch* 8
3.2 Motor 10
3.2.1 *Parsing* do ficheiro XML 10
3.2.2 Aplicação de Iluminação 11
3.2.3 Aplicação de Texturas 11
3.2.4 Algoritmo 12
3.3 Modelo do Sistema Solar 14

4 Conclusão **16**

1 Introdução

A vertente prática da unidade curricular de Computação Gráfica tem como base a utilização do OpenGL, recorrendo à biblioteca GLUT, para a construção de modelos 3D.

A produção dos modelos referidos envolve diversos temas, entre os quais, transformações geométricas, curvas e superfícies, iluminação e texturas. Com o objetivo de aplicar e demonstrar a aprendizagem destes tópicos, foi proposta a realização de um projeto prático, que se encontra dividido em quatro fases.

A existência destas quatro fases tem como objetivo a evolução gradual do desenvolvimento do projeto, sendo que cada uma das fases deverá respeitar os requisitos inicialmente estipulados no enunciado do trabalho.

Este relatório diz respeito à realização da quarta fase do projeto mencionado, que tem como principais objectivos o desenvolvimento do *Engine* para ser capaz de aplicar texturas e iluminação a um modelo e contruir uma cena que utilize que utilize todas as funcionalidades disponíveis através do Sistema Solar, presente na terceira fase do projecto, agora texturado e iluminado.

2 Descrição do Problema

A quarta fase deste projeto tem como objetivo o desenvolvimento de novas funcionalidades aplicadas ao Gerador e ao Motor desenvolvidos na fase anterior. Agora, o Gerador deverá ser capaz de calcular normais e coordenadas de textura das diversas primitivas, e o Motor deverá fazer uso desses parâmetros para aplicar iluminação e texturas. Assim os requesitos estipulados são:

- Gerador (recebe como parâmetros o tipo da primitiva gráfica, parâmetros relativos ao modelo e o nome do ficheiro onde vão ser guardados os vértices):
  - calcular os vectores **normais** e as **coordenadas de textura** de cada primitiva.
- Motor (lê oficheiro XML que contém os modelos das primitivas):
  - ler as características de ilminação do **ficheiro XML**.
  - ler as características de textura do **ficheiro XML**.
  - aplicar **texturas e iluminação** ao modelo.

3 Resolução do Problema

Nesta fase, serão realizadas alterações tanto ao Motor como ao Gerador desenvolvidos anteriormente, com o objetivo de cumprir com os requisitos propostos em 2.

3.1 Gerador

Tendo como objectivo implementar a iluminação, bem como a aplicação de texturas nas primitivas desejadas, será necessário adicionar mais funcionalidades ao gerador desenvolvido até esta fase. Para atingir este fim foi necessário calcular as normais e as coordenadas de textura de cada ponto para cada primitiva, e escrevê-las nos seus respetivos ficheiros. Abaixo estão explicados o métodos de cálculo das normais e das texturas para as primitivas apresentadas.

3.1.1 Plane

Normais

Em cada vértice do plano será aplicada uma normal , sendo esta igual entre todos os vértices. Sendo o plano desenhado no eixo dos x e y, a normal desse plano é um vetor unitário que aponta para o sentido positivo do eixo dos Z. Isto tem como consequência que para todos os vértices que sejam pontos do plano gerarão , cada um, uma normal que aponta no eixo positivo do eixo dos z (0,0,1), ou seja, na direção do utilizador.

Texturas

Tendo em conta que o plano não apresenta divisões o método de cálculo dos pontos de textura para a primitiva do plano consistirá em aplicar as extremidades da textura às extremidades do plano. Isto significa que , por exemplo , as coordenadas do ponto superior esquerdo do plano corresponderão às coordenadas do ponto superior esquerdo da textura e assim por diante.

É importante ainda ter em conta que o referencial da textura utilizado tem uma escala de 0 a 1 cujo plano está centrado na origem. Sendo assim, uma coordenada de textura 0 corresponderá à coordenada do plano mais à direita/baixo enquanto que a coordenada de textura 1 corresponderá à coordenada do plano mais à esquerda/baixo. Por exemplo, o ponto do plano dado pelas coordenadas (-1,1,0) ao ser-lhe aplicada a escala descrita, a coordenada de textura será o ponto (0,1).

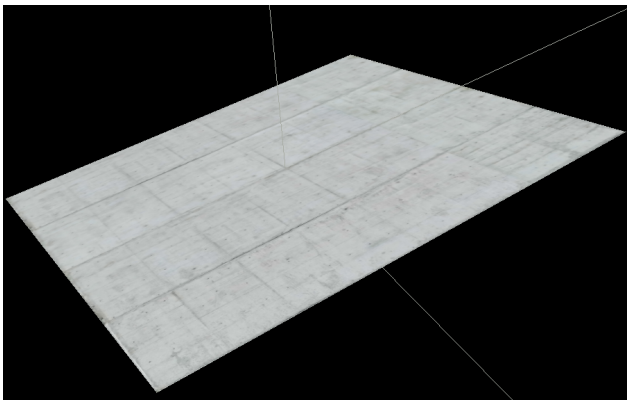

Figura 1: Plano gerado com textura

3.1.2 Box

Normais

Visto que a caixa corresponde a um cubo com 6 faces, para cada uma destas, teremos um vetor normal diferente para cada uma das faces, vetor este que será igual para todos os pontos da face em que se encontra. Tendo isto em conta, estes são os vetores normais para cada uma das faces do cubo:

- Face Direita - Aponta no eixo positivo do eixo dos X , tem coordenadas (1,0,0);
- Face Esquerda - Aponta no eixo negativo do eixo dos X , tem coordenadas (-1,0,0);
- Face Cima - Aponta no eixo positivo do eixo dos Y , tem coordenadas (0,1,0);
- Face Baixo - Aponta no eixo negativo do eixo dos Y , tem coordenadas (0,-1,0);
- Face Frente - Aponta no eixo positivo do eixo dos Z , tem coordenadas (0,0,1);
- Face Trás - Aponta no eixo negativo do eixo dos Z , tem coordenadas (0,0,-1).

Texturas

De forma a desenhar a caixa, separaram-se os pontos de textura a desenhar nas 6 faces do cubo. A divisão foi feita da seguinte forma:

- Face Direita : $x \in [0, \frac{1}{3}]$, $y \in [0, \frac{1}{2}]$
- Face Esquerda : $x \in [0, \frac{1}{3}]$, $y \in [\frac{1}{2}, 1]$
- Face de Cima : $x \in [\frac{1}{3}, \frac{2}{3}]$, $y \in [0, \frac{1}{2}]$

- Face de Baixo : $x \in [\frac{1}{3}, \frac{2}{3}]$, $y \in [\frac{1}{2}, 1]$
- Face de Frente : $x \in [\frac{2}{3}, 1]$, $y \in [0, \frac{1}{2}]$
- Face de Trás : $x \in [\frac{2}{3}, 1]$, $y \in [\frac{1}{2}, 1]$

Feita a divisão dos pontos o grupo desenvolveu posteriormente um algoritmo que recriasse os pontos de textura na mesma ordem da geração de pontos, produzindo a figura abaixo.

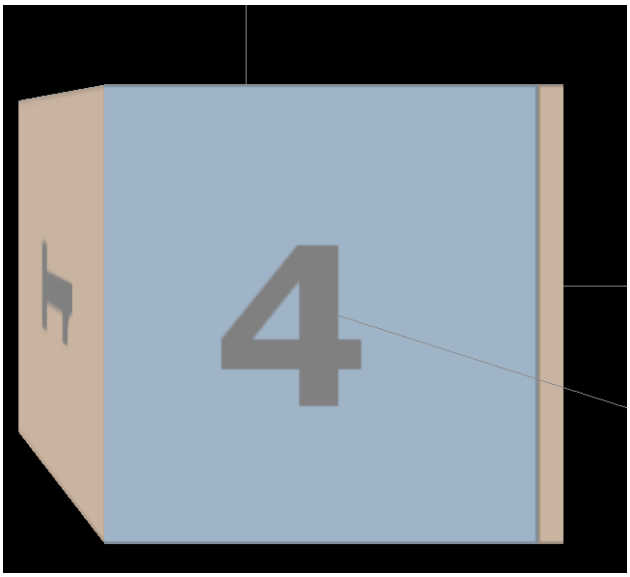

Figura 2: Box gerada com textura

Infelizmente houve um erro de associação de algumas faces às coordenadas de textura pretendidas, produzindo faces mistas por exemplo, o que leva a concluir que o algoritmo poderia ser melhorado de forma a reproduzir uma textura mais adequada para a primitiva em questão.

3.1.3 Sphere

Normais

Para efetuar o cálculo das normais, teremos de percorrer todos os pontos da esfera e para cada um destes pontos temos de definir o vetor normal (perpendicular à superfície). Sabe-se de antemão que o vetor normal tem a mesma direção do ponto. A diferença entre o método de cálculo dos pontos de superfície da esfera para o método de cálculo dos pontos das normais é que no último não multiplicamos o valor do raio (vetor unitário). Sendo assim as fórmulas para as coordenadas x,y,z (e respetivas normais) de cada ponto da superfície são dadas por:

$$\mathbf{x} = \text{raio} \times cos(\beta) \times sin(\alpha)$$
$$\mathbf{y} = \text{raio} \times sin(\beta)$$
$$\mathbf{z} = \text{raio} \times sin(\beta) \times cos(\beta)$$
$$\mathbf{normal}_x = \mathbf{cos(}\beta) \times sin(\alpha)$$
$$\mathbf{normal}_y = \mathbf{sin(}\beta)$$
$$\mathbf{normal}_z = \mathbf{sin(}\beta) \times cos(\beta)$$

Texturas

De forma a associar corretamente uma coordenada de textura a um ponto primeiro tem de se saber que coordenada de textura imprimir em cada triângulo, para este efeito foi criado um algoritmo para desenhar corretamente as coordenadas de textura.

Começa-se por desenhar a parte superior da esfera, cujos valores variam de 0,5 a 1, sendo que esses pontos dependem sempre do número de slices e stacks, ou seja, percorrem-se os dois eixos de coordenadas, em cada iteração, de forma a desenhar todos os pontos superiores.

Após desenhar-se a parte superior da esfera, a parte inferior será desenhada. Desta vez os valores variam entre 0 a 0,5 , sendo o restante algoritmo semelhante ao de desenho superior, apenas aplicado ao desenho dos pontos inferiores.

3.1.4 *Bezier Patch*

Normais

Fez-se o cálculo das normais do patch de bezier recorrendo ao cálculo do produto externo das derivadas (normalizadas) dos vetores u e v.

- $$u = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$
- $$u' = \begin{bmatrix} 3 \times u^2 & 2 \times u & 1 & 0 \end{bmatrix}$$
- $$v = \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$
- $$v' = \begin{bmatrix} 3 \times v^2 \\ 2 \times v \\ 1 \\ 0 \end{bmatrix}$$
- $$M = M^T = \begin{bmatrix} 1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ 3 & -3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$
- P representa a matriz dos pontos de controlo lidos
- $Derivada\_U = u' * M * P * M^T * v^T$
- $Derivada\_V = u * M * P * M^T * v'$

Texturas

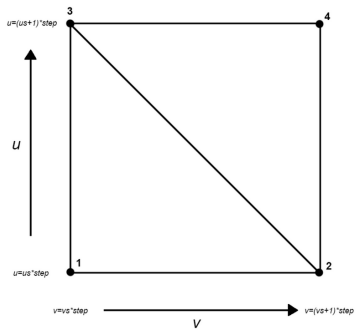

Figura 3: Representação do método de cáculo das coordenadas de textura do *bezier patch*.

Visto que para cada ponto calculado temos dois valores u e v que cujo conjunto forma uma espécie de grelha , de acordo com a figura acima, a coordenada de textura x será dada por v enquanto que y será dada por u.Desta forma, a imagem vai ser dividida numa grelha tendo em conta os valores de v e u.

Figura 4: Teapot gerado com textura

3.2 Motor

3.2.1 *Parsing* do ficheiro XML

Para aplicar as texturas o grupo modificou as estruturas, sendo estas mudanças realizadas com o objetivo de guardar as informações necessárias para gerar a textura e guardar a referência da mesma para cada modelo.

- *PathsInfo*
  - *has_texture*: Variável que indica se o modelo tem ou não uma textura associada
  - *text*: Variável para guardar o *path* para o ficheiro de texturas
- *ModelInfo*
  - *has_texture*: Variável que indica se o modelo tem ou não uma textura
  - *text* : Variável para guardar o *path* para o ficheiro de texturas
  - *tex* : Inteiro que guarda qual a posição da textura no array de texturas
- *textures*
  - Array que guarda todas as texturas geradas de todos os modelos para que sejam acedidas pelos mesmos

Para que seja possível acomodar as informações de iluminação, teve que existir necessariamente um acrescento/modificação de variáveis/estruturas. Neste sentido foram criadas as seguintes estruturas:

- *Col*, que contém três floats(r, g e b) correspondentes aos valores de cor vermelha, verde e azul da luz;
- *Lights*, que armazena 4 variáveis do tipo *Col* sendo que estas variáveis são representativas das diferentes componentes de luz (ambiente,emissiva,difusa, ou especular) e 4 booleanos que indicam se essa componente existe ou não
- *LightType* , que define uma luz, contendo o seu tipo, posição/direcção, *cut_off* e *exponent* se necessário, uma variável de tipo *Lights* no caso de a luz conter definições de componentes e um booleano indicando se deve ser posicionada antes ou depois da câmara;

```
    typedef struct rgb {
    float r;
    float g;
    float b;
}Col;

typedef struct light {
    bool amb= false;
```

```
    bool differ = false;
    bool spect = false;
    bool emiss = false;
    Col ambient;
    Col diffuse;
    Col specular;
    Col emissive;
}Lights;

typedef struct type {
    char* light_type;
    bool joint_camera;
    float posx;
    float posy;
    float posz;
    Vertex spotDir;
    Lights lights;
    float cutoff;
    float exponent;
    float directional;
} LightType;

vector<LightType> light_type;
```

3.2.2 Aplicação de Iluminação

De forma a poder iluminar a cena, é necessário que o Engine consiga ler toda a informação necessária do ficheiro XML. Assim sendo, deve ser possível de ler a seguinte informação:

- O tipo de luz a aplicar (*POINT*, *SPOT* ou *DIRECTIONAL*);
- A posição/direcção da luz a aplicar, dependendo do tipo de luz;
- No caso de o tipo da luz ser *SPOT*, devemos ter que ler o *cut_off* e o *exponent* do cone de luz;
- Caso exista, a componente ambiente, emissiva, difusa ou especular;
- Caso exista, deve ser lido também a variável *joint_camera*, que indica se esta luz deve estar posicionada com a câmara (e mover-se com ela) ou não.

3.2.3 Aplicação de Texturas

Após implementar as estruturas de dados, utilizamos um array que guardará todas as texturas (o array textures) e na inicialização da engine o programa irá gerar as texturas com a função *initGL()* e irá armazenar nas estruturas o local do array que a textura do modelo está associado.

3.2.4 Algoritmo

Carregamento dos dados

Visto que as luzes são definidas uma vez apenas, no inıcio do ficheiro XML, toda a sua informação é também lida no inıcio utilizando para isso a função *lightAux()*, que lê todos os atributos de uma dada luz. Assim sendo, o algoritmo geral da função é o seguinte:

- Para cada elemento XML, se for *light*:

  – verificar a existência do atributo *jointCamera*, caso exista atribuir o valor extraıdo na estrutura, se não existir atribuir o valor *false*;

  – verificar o tipo de luz e extrair a informação de acordo com isso:

  1. Se a luz for do tipo *POINT*, extrai a sua posição e as componentes de luz, caso existam;

  2. Se a luz for do tipo *DIRECTIONAL*, extrai a sua direcção e atribui ao campo *directional* na estrutura o valor 0.0;

  3. Se a luz for do tipo *SPOT*, extrai a sua posição, a sua *spotDirection*, bem como o *cutoff* e o exponent. Caso existam, extrai também as componentes de luz.

  – Adicionámos ao array global de luzes, a luz acabada de extrair.

Finalmente, de forma a conseguirmos carregar as componentes dos materiais dos modelos, extendemos o comportamento da função *readModels()* e, neste momento, para cada modelo lido, devem ser extraıdas as componentes luz, colocando-as todas numa estrutura *Lights* e, no final, atribuindo à variável *Lights* da estrutura de informação do modelo dessa estrutura com toda a informação recolhida.

Após o carregamento dos dados com sucesso, as luzes devem ser posicionadas e as componentes aplicadas tanto às próprias luzes como aos objetos. Assim, na função *renderScene()*, é chamada a função *posL(bool before)* antes de posicionar a câmara e após posicionar a câmara, sendo que o argumento *before* que é passado à função indica se queremos posicionar as luzes antes ou depois da câmara (valor *true* indica posicionamento antes da câmara e valor *false* indica posicionamento após a câmara). Além disso, devemos acrescentar a definição dos materiais ao comportamento já definido para cada modelo. Assim, para cada modelo, se a

sua variável *Lights* tiver uma componente X, ela é aplicada consoante os valores que nela figuram, sendo que X pode ser ambiente, difusa, emissiva ou especular. A função *posL(bool before)* é definida pelo seguinte algoritmo:

- De i=0 até i= número de luzes lidas (só são ativas as primeiras 8 , que é o máximo permitido pelo OpenGL):

  – A variável *lt* corresponde à estrutura referente à luz presente na posição *i* do vetor global *lightType* que contém as estruturas das luzes presentes;

  NOTA: Estas instruções são repetidas de j=0 até j=7(no máximo), a única alteração é o número da luz a aplicar em *GL LIGHT*

  – Se (*before==true* $\wedge$ *lt.jointCamera==true* $\vee$ *(before == false* $\wedge$ *lt.jointCamera == false)*) :

  1. Colorir as luzes, através da função *colorL()*;

  2. Se o tipo de luz for *POINT* ou *DIRECTIONAL*, o vetor de 4 posições é dado pelos atributos (r,g,b) e pelo atributo *directional* da estrutura *lt*, que são utilizados na chamada à função *glLightfv()*, em que a luz a utilizar é o número dado por *j*;

  3. Se o tipo de luz for *SPOT*, a posição é dada pelos três atributos (x,y,z), a direcção é dada pelo *Vertex* (estrutura que representa um vértice de coordenadas x,y,z) *spotDir*, o *cutoff* é dado pelo atributo *cut off* e o *exponent* é dado pelo atributo *exponent* da estrutura. Estes são utilizados em consecutivas chamadas à função *glLightfv()*, uma vez para a posição, outra para a *spotDirection*, outra para o *cutoff* e a última para o *exponent*.

  4. Se j==7 , estamos a tratar da última luz possıvel, logo fazemos*break* no ciclo.

Desta forma, falta apenas definir o comportamento da função *colorL(int color, LightType lt)*, em que *color* é o número da luz que estamos a colorir (de 0 até 7) e *lt* corresponde à estrutura onde estão as variáveis necessárias para o processamento de um tipo de luz. Todo o algoritmo é uma estrutura de condição *switch*, em que a única alteração no corpo de cada um dos casos, é a luz a ser chamada, pelo que iremos mostrar o algoritmo apenas uma vez com *GL LIGHTj*, em que j varia de 0 a 7.

Assim sendo a função repete as seguintes instruções para cada uma das componentes:

- Se a componente a tratar existe (valor *true* na estrutura), o vetor de 4 posições para a cor é igual as componentes (r,g ,b) presentes na estrutura mais o valor 1.0;
- Completam-se as instruções, chamando a função *glLightfv(GL LIGHTj, GL componente, vetor de cor).*

Tendo estas funções definidas e aplicadas, o Engine já está preparado para iluminar a cena, faltando executar as primitivas de inicialização após o carregamento dos dados. Isto é feito através da função por nós criada, *enableLighting().*

3.3 Modelo do Sistema Solar

Para atribuir texturas ao Sistema Solar, cada objeto possui a sua, retiradas do Solar System Scope.

No que toca á iluminação, utilizamos uma fonte de luz do tipo *POINT* posicionada na origem, uma vez que no contexto do Sistema Solar o sol é o único objeto emissor de luz. De forma a replicar a visualização de forma mais realista do Sistema Solar, será necessário ter uma componente forte de luz vermelha (devido a elevadas temperaturas + energia do Sol) tendo posteriormente de ser balanceada com as outras componentes.

Isto significa que uma das componentes, neste caso a luz de cor azul, "aumenta"o espectro de luz que radia do sol enquanto que a luz verde fará o contrário. Sendo assim , para o modelo final , estas componentes devem ser equivalentes.

Para este efeito, o ficheiro XML deverá conter o seguinte:

```
<!--Fonte de Luz-->
<light type="POINT" posX="0" posY="0" posZ="0"/>

<!--Sol-->
<model file="sphere.3d" texture="sun.jpg" emissR="0.9" emissG="0.4"
,→ emissB="0.4"/>
```

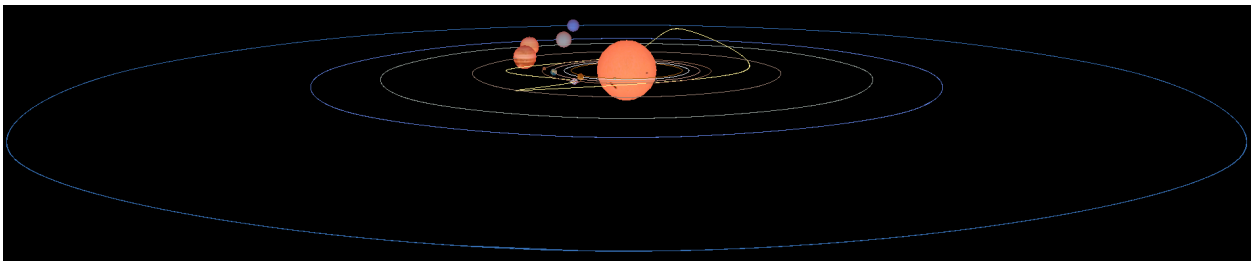

Figura 5: *Representação do Sistema Solar obtido*

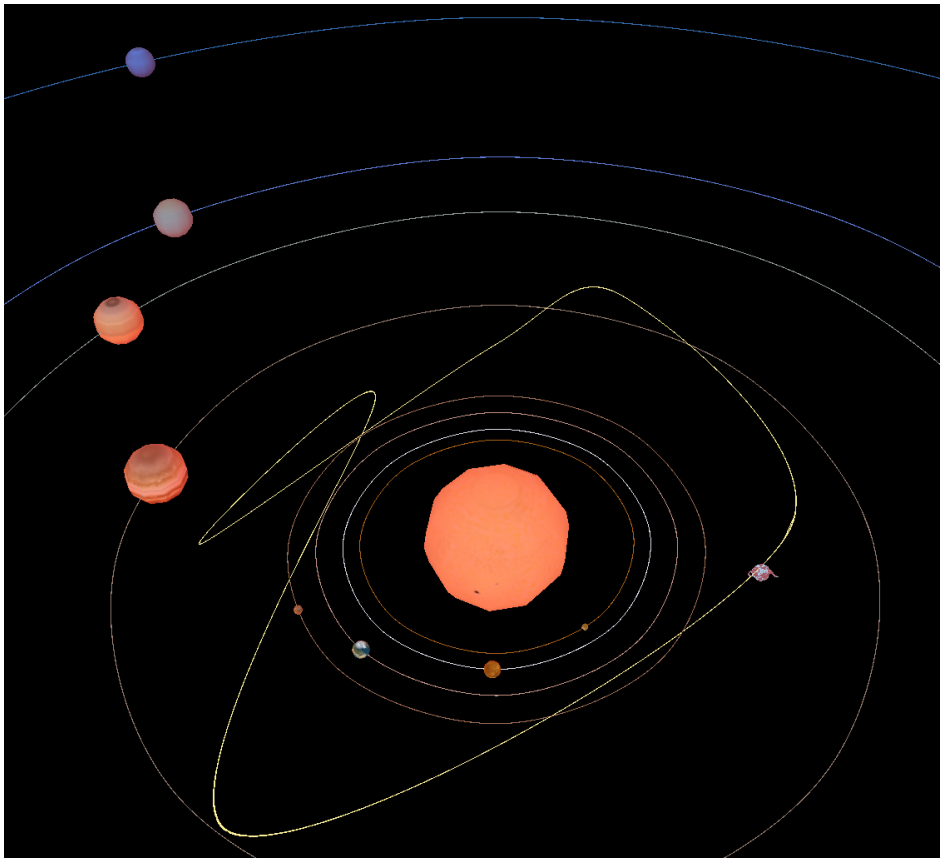

Figura 6: *Representação do Sistema Solar obtido, de outro ângulo*

4 Conclusão

Terminada a realização desta quarta fase do projecto prático, está na altura de uma avaliação crítica, onde consideramos relativamente bem sucedida, visto que pensamos cumprir com todos os requisitos estabelicidos inicialmente. Foi construído um Motor capaz de aplicar texturas e iluminação a um cenário com base nas informações descritas no ficheiro XML. Fomos capazes de determinar as normais e as coordenadas para todas as primitivas necessárias, com a excepção ao cone, que foi a figura geométrica que ao qual não conseguimos aplicar o cálculo das normais, nem dos pontos de textura. Para além do erro de representação das texturas da caixa, mencionado anteriormente.

Este projecto começou com a criação de quatro primitivas, desenvolveu-se, na segunda fase, para a aplicação de transformações geométricas e, com maior complexidade, para a aplicação das curvas, patches de *Bezier* e VBOs na terceira fase. Agora, com a utilização de normais e coordenadas de textura, para a aplicaçãos de iluminação, conseguimos finalmente concretizar o objectivo do projecto nesta ultima fase.

Em relação ao modelo requerido para esta fase (Sistema Solar dinâmico com texturas e iluminação), conseguimos realizar o plano geral requerido, embora fossem implementadas as estruturas para representação de outros tipos de luz, neste projecto, apenas foi utilizada a luz do tipo *POINT*.

Em suma e, após terminar a realização deste prjecto, concluímos que a matéria leccionada foi aplicada durante o desenvolvimento dos diferentes problemas que nos eram propostos, o trabalho realizado está, na nossa opinião, num bom estado, apesar de ter alguns aspetos que, com certeza, poderiam e deveriam ter sido melhorados.