Universidade do Minho

LEI 3ºAno 2ºSemestre

Computação Gráfica

Mini Motor 3D

Fase 4 - Normais e Coordenadas de Textura

Daniel Caldas a67691
José Cortez a67716
Marcelo Gonçalves a67736
Ricardo Silva a67728

31 de Maio de 2015

Conteúdo

1 Introdução

1.1 Contexto

No âmbito da UC de Computação Gráfica, surge um projeto prático para consolidar os conceitos adquiridos ao longo do semestre: a criação de um mini motor 3D.

Usando as tecnologias C++ e OpenGL, será então desenvolvido o motor 3D ao longo de quatro fases, sendo que esta segunda está resumida em tópicos na subsecção seguinte.

1.2 Resumo

Nesta etapa foi efetuada:

- Modificações na estrutura de dados e ficheiro XML;
- Arquitetura final do mini motor 3D;
- Cálculo de **normais** para os planos dos sólidos;
- A partir do ponto anterior introduzir **luminosidade** nas cenas desenhadas;
- Cálculo das **coordenadas de textura** para cada forma geométrica;
- Implementação da câmara do tipo **FPS** (*First Person Shooter*);
- Sistema solar e resultados finais;

2 Estrutura de dados e XML

Por forma a serem acomodadas as novas funcionalidades desta terceira etapa o ficheiro XML que contém a informação para desenhar cada cena, sofreu ligeiras alterações nomeadamente:

- Novo atributo **texture** na tag **model** por forma a podermos referenciar ficheiros para a aplicação de texturas;

- Nova tag **luzes** com tags filho **luz** em que foi definida a seguinte notação para se receberem os diferentes tipos de luzes:

  - Tipo **POINT** com os parâmetros: **posX**, **posY**, **posZ**,**pv**;
  - Tipo **AMBIENT** com os parâmetros: **ambR**, **ambG**, **ambB**;
  - Tipo **DIFFUSE**, com os parâmetros: **diffR**, **diffG**, **diffB**;

```
<grupo prof="1">
    <grupo prof="2">
        <luzes>
          <luz tipo="POINT" posX="0" posY="10" posZ="0" />
        </luzes>
        <rotacao tempo="15" angulo="360" eixoX="0" eixoY="1" eixoZ="0" />
        <modelos>
            <modelo ficheiro="sol.3d" textura="pokeball.png" />
        </modelos>
    </grupo>
    <grupo prof="2">
        <translacao tempo="12.89">
              <ponto X="-42" Y="0" Z="0"/>
              <ponto X="-39.94" Y="0" Z="-9.27"/>
              <ponto X="-33.97" Y="0" Z="-17.63"/>
              <ponto X="-26.77" Y="0" Z="-23.11"/>
              <ponto X="-17.88" Y="0" Z="-27.14"/>
              <ponto X="-10.44" Y="0" Z="-29.05"/>
              <ponto X="0" Y="0" Z="-30"/>
```

Figura 1: Excerto de novo formato do ficheiro XML.

Foram adicionadas variáveis às classes necessárias para que se possam armazenar os dados descritos, mas apenas com destaque para a classe **Luz3D**.

```
class Luz3D {
public:
    int gl_i;      // Índice
    string tipo;   // POSITION, AMBIENT ou DIFUSE
    float *values; // Máximo será 4 para GL_POSITION
};
```

Figura 2: **Classe Luz3D**

Na figura 2 está então codificada a nossa classe Luz3D. No nosso bloco de inicializações do *Glut* acrescentamos agora também o *setup* das luzes e um ciclo para inicialização das luzes.

```
// Luz
glEnable(GL_LIGHTING);
// Ciclo para inicialização das luzes individuais
for (int i = 0; i < luzes.size(); i++){
    glEnable(GL_LIGHT0 + i);
}
```

Figura 3: Bloco de código da função **prepare_glut()** que inicializa as luzes

Como podemos verificar na imagem 3 para cada luz armazenada na estrutra de dados fazemos a respetiva inicialização usando **glEnable(GL_LIGHT0 + i)** em que **i** é o índice da luz na estrutura.

Posteriormente na função **renderScene()**, implementamos um ciclo que percorre os objetos Luz3D armazenados, e em função do tipo de luz é executada a função **glLightfv** com os parâmetros adequados,

```
// Luzes
for (Luz3D luz : luzes){
    if (luz.tipo.compare("POINT") == 0){
        glLightfv(GL_LIGHT0 + luz.gl_i, GL_POSITION, luz.values);
    }
    else if (luz.tipo.compare("AMBIENT") == 0){
        glLightfv(GL_LIGHT0 + luz.gl_i, GL_EMISSION, luz.values);
    }
    else if (luz.tipo.compare("DIFFUSE") == 0){
        glLightfv(GL_LIGHT0 + luz.gl_i, GL_DIFFUSE, luz.values);
    }
}
float white[4] = { 1, 1, 1, 1 };
glMaterialfv(GL_FRONT, GL_DIFFUSE, white);
```

Figura 4: Bloco de código referente a luzes da função **renderScene()** que inicializa as luzes

como podemos observar na figura 4.

3 Normais e Coordenadas de Textura

Nesta fase um dos principais objectivos era gerar as normais e as coordenadas de textura para cada vértice, no momento da criação do ficheiro de uma primitiva.

3.1 Áreas Planas

Para estas figuras (Triângulos,Círculos e Rectângulos), foi adoptado a mesma técnica, depois de gerados os vértices, são geradas as normais e em seguida as coordenadas de textura.

As normais aos vértices destas figuras sã todas perpendiculares ao eixo dos 'z' e dos 'x' e paralelas ao eixo dos 'y', ou seja, (0 , 1 , 0) para o caso da face superior, e (0, -1 , 0) no caso da face inferior da figura. As normais são assim representadas pois as figuras são geradas no plano xOz.

Apresentamos em seguida, Figura 5, um exemplo das normais geradas.

```
file << 36 << "\n";

file << 0 << " " << 1 << " " << 0 << "\n";
file << 0 << " " << 1 << " " << 0 << "\n";
file << 0 << " " << 1 << " " << 0 << "\n";

file << 0 << " " << 1 << " " << 0 << "\n";
file << 0 << " " << 1 << " " << 0 << "\n";
file << 0 << " " << 1 << " " << 0 << "\n";

file << 0 << " " << -1 << " " << 0 << "\n";
file << 0 << " " << -1 << " " << 0 << "\n";
file << 0 << " " << -1 << " " << 0 << "\n";

file << 0 << " " << -1 << " " << 0 << "\n";
file << 0 << " " << -1 << " " << 0 << "\n";
file << 0 << " " << -1 << " " << 0 << "\n";
```

Figura 5: **Exemplo do momento da geração das normais, no caso do Rectângulo**

As coordenadas de textura foram geradas a pensar em que em cada face se consiga ver a textura completa, ou seja, no caso do rectângulo, as coordenadas de textura foram geradas para que os vertices do rectângulo coincidam com os vértices da textura. Podemos observar o pretendido na Figura 6.

Na Figura 7 podemos observar o resultado final. O que foi desenvolvido para este caso do rectângulo, foi também aplicado no caso de todas as figuras planas desenvolvidas, sendo que dependendo da figura o resultado final é diferente de figura para figura.

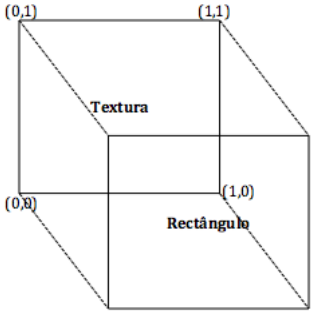

Figura 6: **Aplicação da textura na face**

Figura 7: **Resultado final**

3.2 Paralelepipedo

Para o caso do paralelepipedo, foi adoptado a mesma técnica das figuras anteriores, ou seja, depois de gerados os vértices, são geradas as normais e em seguida as coordenadas de textura.

As normais aos vértices do paralelepipedo, foram criadas tendo em vista a face do paralelepidedo que está a ser gerada. Apresentamos em seguida, Figura 8, as normais geradas para o caso dos paralelepipedo.

As coordenadas de textura foram geradas com o intuito de garantir que a textura escolhida seja repetida em cada face do paralelepipedo. Podemos observar o resultado na Figura 9.

```
//topo
file << 0 << " " << 1 << " " << 0 << "\n";
file << 0 << " " << 1 << " " << 0 << "\n";
file << 0 << " " << 1 << " " << 0 << "\n";

file << 0 << " " << 1 << " " << 0 << "\n";
file << 0 << " " << 1 << " " << 0 << "\n";
file << 0 << " " << 1 << " " << 0 << "\n";

//base
file << 0 << " " << -1 << " " << 0 << "\n";
file << 0 << " " << -1 << " " << 0 << "\n";
file << 0 << " " << -1 << " " << 0 << "\n";

file << 0 << " " << -1 << " " << 0 << "\n";
file << 0 << " " << -1 << " " << 0 << "\n";
file << 0 << " " << -1 << " " << 0 << "\n";
```

Figura 8: **Exemplo do momento da geração das normais, no caso do Paralelepipedo**

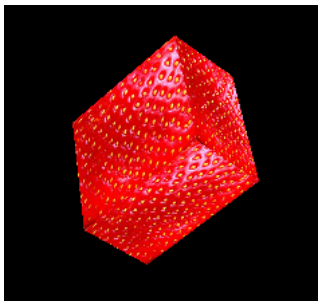

Figura 9: **Resultado final**

3.3 Esfera

Para o caso da esfera, foi adoptado a mesma técnica das figuras anteriores, ou seja, depois de gerados os vértices, são geradas as normais e em seguida as coordenadas de textura. No entanto as normais e as coordenadas de textura não são calculadas de forma direta, isto é, não as conseguimos deduzir só a olhar para a figura, como foi feito nos casos anteriores.

As normais aos vértices da esfera são geradas através dos ângulos que o vetor, que vai da origem até ao vértice correspondente, faz com o eixo dos 'y', e com o eixo dos 'z'. Através desta informação a normal é cálculada e gerada para o ficheiro correspondente.

Apresentamos em seguida, Figura 10, as normais geradas para o caso das esferas.
As coordenadas de textura foram geradas com o intuito de garantir que a textura escolhida englobe toda a esfera, sem repetição da mesma parte da textura em dois triângulos. Podemos observar o resultado na Figura 11.

```
aVertex[pos]=r*sin(angv)*sin(angh);
aNormal[pos++]=sin(angv)*sin(angh);

aVertex[pos]=r*cos(angv);
aNormal[pos++]=cos(angv);

aVertex[pos]=r*sin(angv)*cos(angh);
aNormal[pos++]=sin(angv)*cos(angh);

aTexture[tpos++] = 1+((float)j/(float)(f+1)) ;
aTexture[tpos++] = (float)i/(float)(c+1) ;
```

Figura 10: **Exemplo do momento da geração das normais, no caso da Esfera**

Figura 11: **Resultado final**

3.4 Cone

Para o caso do cone, foi adoptado a mesma técnica da esfera. No caso do cone temos, o problema dividido em dois, a base do cone e o corpo.

As normais para o caso da base é fácil de observar, como o cone é gerado na origem, que as normais serão paralelas ao eixo dos 'y' e perpendiculares ao eixo dos 'x' e 'z', e serão para baixo, ou seja, (0, -1, 0).

No caso do corpo as normais serão são geradas através do ângulo que o vetor faz com o eixo dos 'y'. Através desta informação a normal é cálculada e gerada para o ficheiro correspondente.

Apresentamos em seguida, Figura 12, as normais para o corpo do cone geradas.

As coordenadas de textura foram geradas com o intuito de garantir que a textura escolhida englobe todo, sem repetição da mesma parte da textura em dois triângulos. Podemos observar o resultado na Figura 13.

```
aVertex[pos] = fr*sinf(ang);
aNormal[pos++] = sinf(ang);
aVertex[pos] = alt;
aNormal[pos++] = 0;
aVertex[pos] = fr*cosf(ang);
aNormal[pos++] = cosf(ang);

aTexture[tpos++] = (float)i/(float)ncamadas ;
aTexture[tpos++] = (float)f/(float)nlados ;
```

Figura 12: **Exemplo do momento da geração das normais, no caso da Cone**

Figura 13: **Resultado final**

4 Câmara em Modo FPS

Atendendo aos extras que se sugeriram para implementar o grupo integrou no projeto a câmara em modo explorado, até porque é útil para verificarmos o detalhe das cenas desenhadas.

No ficheiro **fps.cpp** e respetivo *header file* temos então algumas funções que efetuam os cálculos necessários para fazer a câmara mover-se livremente no nosso mundo.

Por forma a melhor explicitar a implementação e fazer a ponte com a teoria por detrás do movimento desta câmara o grupo criou um pequeno esquema que apresentamos de seguida que ilustra com detalhe quais as variáveis no código que representam os parâmetros teóricos na figura da esfera, ainda um teclado que deixamos como uma espécie de instrução para como utilizar a câmara.

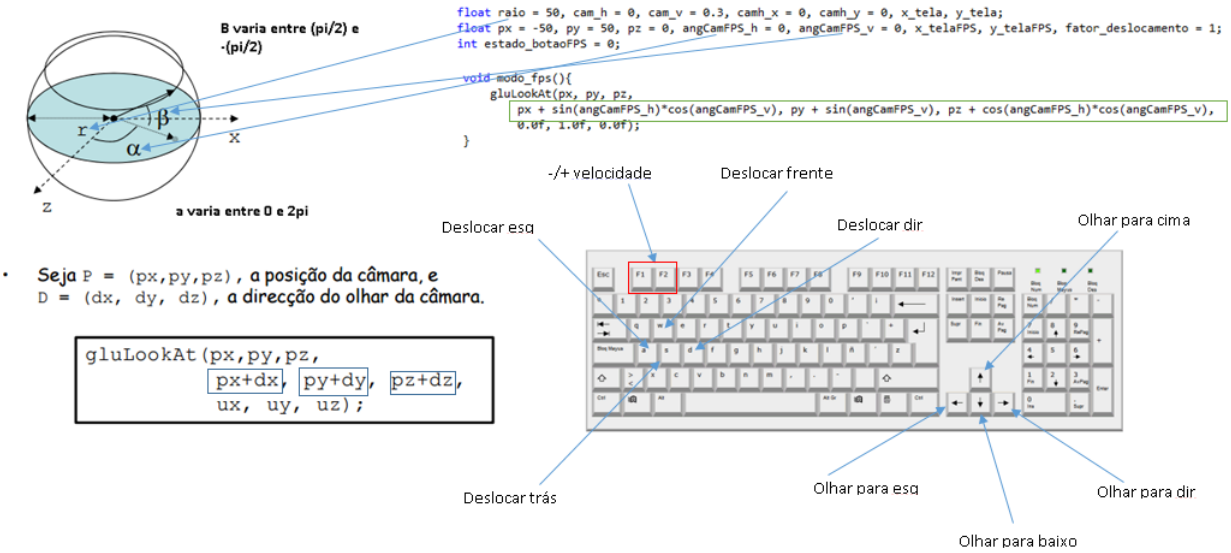

Figura 14: Esquema

Podemos então observar o esquema explicitado anteriormente na figura 14.

5 Sistema Solar

Finalmente, juntando todas as novas peças para desenhar uma cena mais realista, agora contemplando luzes e texturas, desenhamos então a nossa última versão do sistema solar. Mas primeiro acrescentamos alguma informação ao nosso ficheiro xml, que contém os dados para desenhar o sistema solar, nomeadamente o **nome do ficheiro fonte da textura** para o respetivo planeta,

```
<imagem>
  <luzes>
        <!-- Luz 0 -->
        <luz id="0" tipo="POINT" posX="0" posY="17" posZ="0" pv="1"/>
        <luz id="0" tipo="AMBIENT" ambR="0.2" ambG="0.2" ambB="0.2" />
        <luz id="0" tipo="DIFFUSE" diffR="0" diffG="1" diffB="1" />
        <!-- Luz 1 -->
        <luz id="1" tipo='POINT' posX="0" posY="-17" posZ="0" pv="1"/>
        <luz id="1" tipo="DIFFUSE" diffR="1" diffG="1" diffB="1" />
        <!-- Luz 2 -->
        <luz id="2" tipo='POINT' posX="17" posY="0" posZ="0" pv="1"/>
        <luz id="2" tipo="DIFFUSE" diffR="1" diffG="1" diffB="1" />
        <!--Luz 3-->
        <luz id="3" tipo='POINT' posX="-17" posY="0" posZ="0" pv="1"/>
        <luz id="3" tipo="DIFFUSE" diffR="1" diffG="1" diffB="1" />
        <!--Luz 4-->
        <luz id="4" tipo='POINT' posX="0" posY="0" posZ="17" pv="1"/>
        <luz id="4" tipo="DIFFUSE" diffR="1" diffG="1" diffB="1" />
        <!--Luz 5-->
        <luz id="5" tipo='POINT' posX="0" posY="0" posZ="-17" pv="1"/>
        <luz id="5" tipo="AMBIENT" ambR="0.2" ambG="0.2" ambB="0.2" />
        <luz id="5" tipo="DIFFUSE" diffR="1" diffG="1" diffB="1" />
  </luzes>
  <grupo prof="1">
      <grupo prof="2">
          <rotacao tempo="15" angulo="360" eixoX="0" eixoY="1" eixoZ="0" />
          <modelos>
              <modelo ficheiro="sol.3d" textura="sol.jpg" />
          </modelos>
      </grupo>
      <grupo prof="2">
          <translacao tempo="12.89">
```

Figura 15: Excerto do ficheiro xml que contém os dados para desenhar o sistema solar.

e claro as luzes a incluir na cena que são definidas dentro do bloco <**luzes**></**luzes**>, declarado no início do ficheiro xml, antes sequer de se abrir qualquer grupo.

De seguida apresentamos alguns exemplos do nosso sistema solar, de ângulos diferentes obtidos a partir da navegação com a câmara no modo **FPS**.

5.1 Exemplos

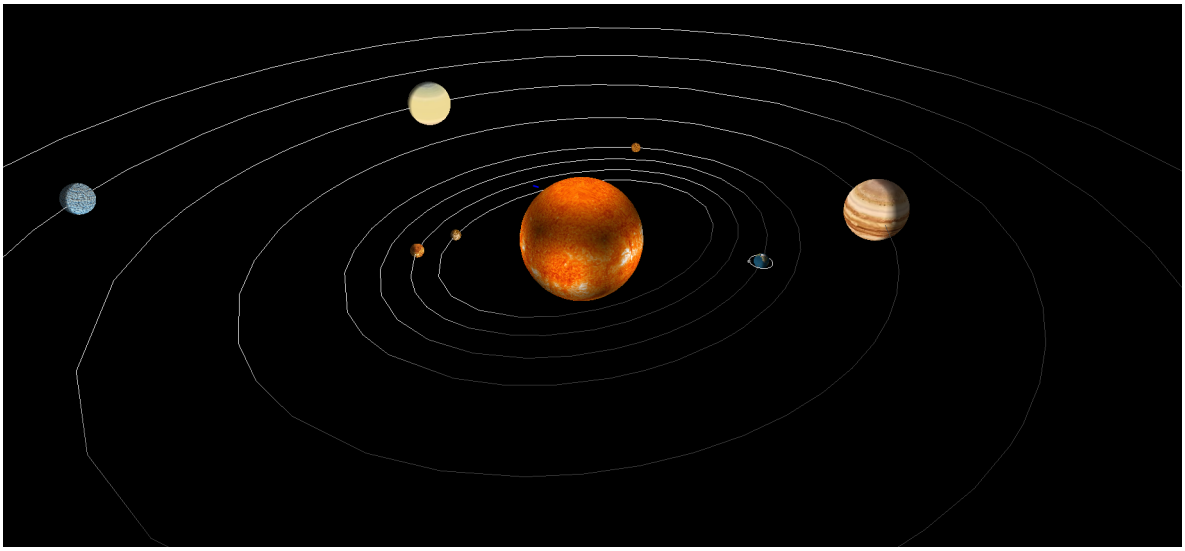

Figura 16: O sistema solar com a aplicação das texturas e iluminação.

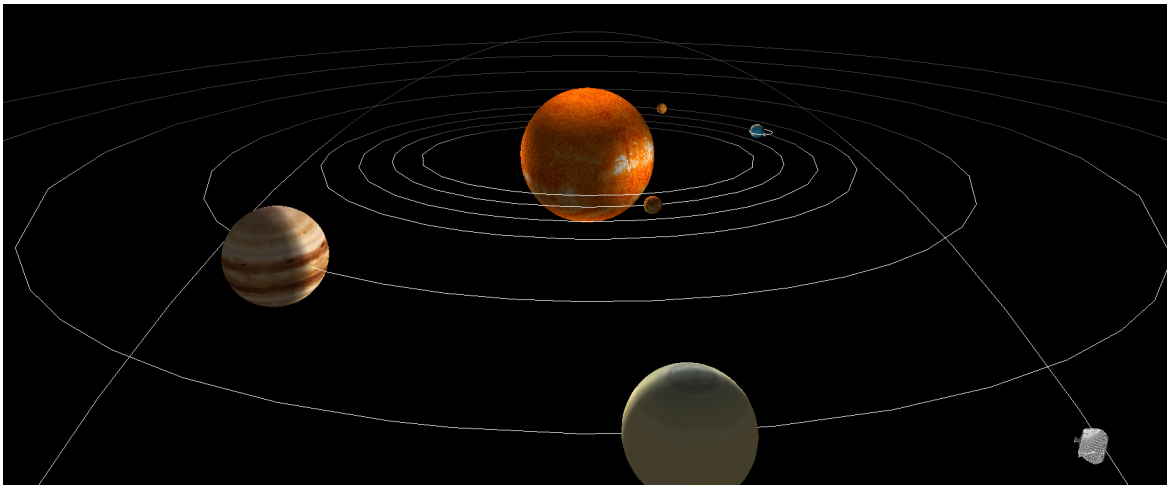

Figura 17: O sistema solar agora com um cometa, o **teapot**

6 Conclusão

Nesta quarta etapa estudamos algumas das ferramentas do **Open GL** que nos permitem personalizar e trazer aspetos do mundo real para uma cena que desenhamos, são eles a iluminação e a aplicação de texturas a objetos neste nosso caso, a planetas.

Fazendo um balanço final do trabalho realizado agora nesta quarta etapa, o grupo cumpriu com os requisitos das quatro etapas e ainda conseguiu explorar uma funcionalidade extra.