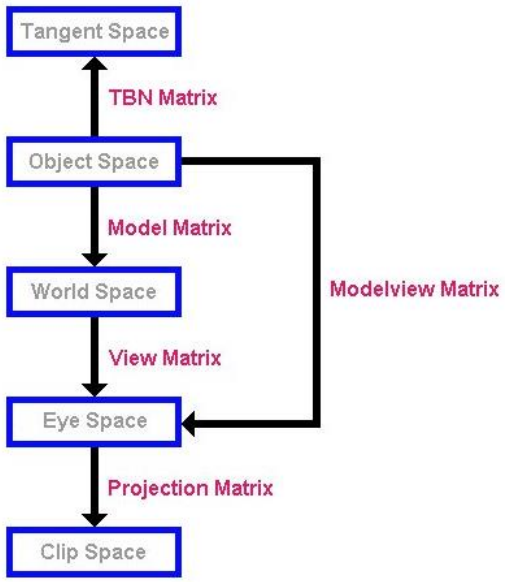
Universidade Federal de Santa Maria
Departamento de Eletrônica e Computação
Prof. Cesar Tadeu Pozzer
Disciplina: Computação Gráfica Avançada
pozzer@inf.ufsm.br
19/08/2017 – TODO - TBN

Iluminação com Shader – Conceitos Básicos

Iluminação Difusa

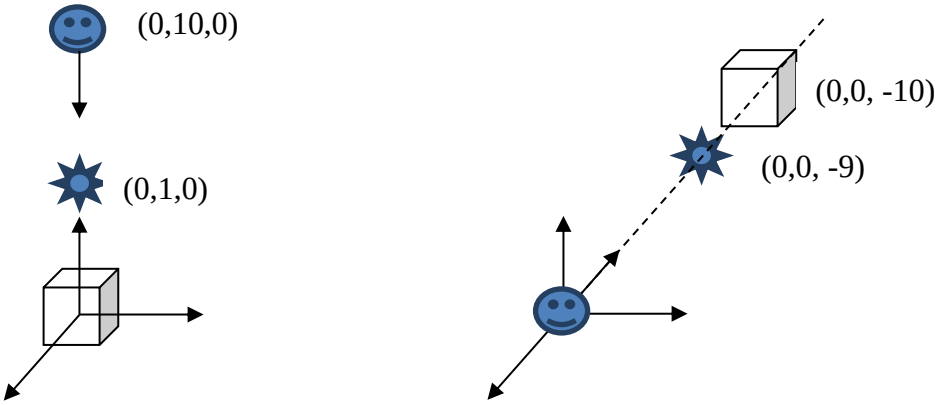
Antes de iniciar a renderização e iluminação de uma cena, a coordenada da luz é transformada para coordenadas de olho (câmera) antes de chegar no pixel shader. Para que isso ocorra, a transformação da posição da luz deve ser feita logo após a chamada da função lookAt() e antes de qualquer transformação sobre objetos. Para a transformação da luz é utilizada a matriz de ModelView, para que a translação/rotação sofrida pela câmera seja considerada [2].

A matriz de ModelView (neste caso apenas a matriz de View, pois a matriz não possui nenhuma transformação aplicada sobre modelos) contém translações e rotações para levar a câmera à origem do sistema de coordenada do mundo, neste caso a coordenada (0,0,0), olhando em direção ao eixo z negativo (para dentro da tela). Esta sequência de etapas deve ser usada quando a luz estiver em coordenadas do mundo, e não associada à posição de algum objeto da cena, como no caso de uma luminária colocada sobre uma mesa.



O vetor lightPos deve ser definido em coordenadas homogêneas, com a componente *w*=1, para que transformações de translação também sejam levadas em conta. Se a componente *w* for zero, apenas as operações de rotação serão aplicadas. Deve-se observar que após a aplicação da matriz

de View, as coordenadas relativas de posição de fonte luminosa, olho e vértices devem ser as mesmas (para o vetor normal se aplica outra regra). O seguinte exemplo ilustra a transformação de uma luz posicionada em (0,1,0), objeto centrado na origem e o observador na posição (0, 10, 0). Ao final da transformação, o observador está na origem do sistema.



Como a posição da luz é definida como uma variável Uniform, esta pode ser transformada pela matriz de View (sem transformações aplicadas sobre objetos) antes de ser passada para o programa de shader (mais eficiente, pois o cálculo é feito uma única vez para toda a cena), ou processada no vertex shader. No seguinte exemplo, `lightPos` é definida como um vec4, não esquecendo de colocar a coordenada homogênea como 1.

```
void display(void)
{
   LookAt(0,10,0,  0,0,0,  0,0,-1);
   …
   shader.setUniform("VM", viewMatrix); //definido pelo comando LookAt
   shader.setUniform("lightPos", lightPos);
   renderScene();
}
```

```
uniform mat4 VM;
uniform vec4 lightPos;

out     vec3 outLightPos; //posicao transformada da Luz enviada para o Fragment shader

void main()//Vertex shader
{
   outLightPos = VM * lightPos;

   gl_Position = ...;
}
```

Para aplicar a matriz de ModelView sobre a luz antes de ser enviada ao pipeline gráfico, deve-se inicialmente ler a matriz da classe MyGL com o comando getModelViewMatrix(). Para o exemplo em questão, tem-se a seguinte matriz de ModelView, lembrando-se que ela contém apenas a transformação do comando LookAt(). Como exemplo, multiplicando-se o ponto da luz [0,1,0,1] por esta matriz obtém-se o ponto [0,0,-9,1], como mostrado na figura acima.

$$ModelView = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -10 & 1 \end{bmatrix}$$

Para a transformação de normais, deve-se utilizar a matriz inversa transposta da ModelView $(M^{-1})^T$. Essa matriz, de dimensão 3x3, é definida no pipeline de shader do OpenGL 3.0 como gl_NormalMatrix. Isso se faz necessário caso a ModelView tenha matrizes de escala, deixando assim de ser uma matriz ortogonal (a transporta não é igual a inversa). Se a matriz for ortogonal, temos que $M^{-1}=M^T$, ou seja, $(M^T)^T=M$. Caso a matriz for ortogonal, pode-se aplicar a ModelView diretamente também sobre os vetores normais, desconsiderando-se neste caso a componente homogênea (parte que contem translações). Deve-se lembrar de que vetores normais são sempre definidos em relação à origem, e não estão associados a uma coordenada no espaço, não necessitando assim sofrer transformações de translação.

Outra forma de fazer a iluminação é usando coordenadas do objeto. Neste caso, tanto coordenadas de normal, olho, vértices e fontes luminosas são mantidos de forma inalterada no processo de iluminação, como mostrado no seguinte exemplo, relativo à configuração de cena 3D mostrada anteriormente. Observe que não é utilizado gl_Position no cálculo de iluminação.

```
uniform vec4 lightPos;

out vec3 outLightPos;
out vec3 normal;
out vec3 coord;

void main()//Vertex shader
{
    outLightPos = lightPos;
    normal = normalize(glNormal);
    coord  = vec3(gl_Vertex); //ou gl_Vertex.xyz
    gl_Position = ftransform();
}


in vec3 luzPos, normal, coord;
out vec4 fragColor

void main()//Fragment shader
{
   float intensity;
   vec3  vetDirLuz;
   //o vetor direcao deve ser normalizado apos a subtracao.
   vetDirLuz = normalize( luzPos - coord );
   intensity = max(dot(normal,vetDirLuz),0.0);
   fragColor = vec4(intensity,intensity,intensity,0);
}
```

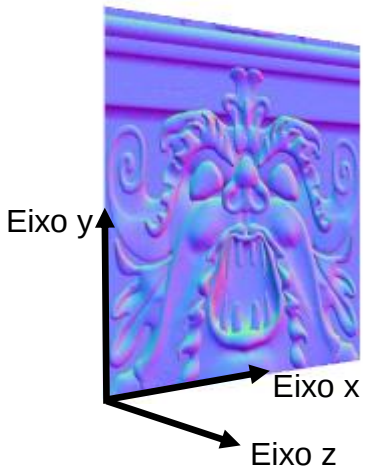
Iluminação por Toon Shading

Para simular iluminação tipo toon shading deve-se definir intervalos do ângulo da luz com o vetor normal da superfície, como no seguinte exemplo.

```
in vec3 normal, lightDir;
void main()//Fragment shader
{
   float intensity;
   vec4 color;
   intensity = max(dot(lightDir,normal),0.0);
   if (intensity > 0.98)
      color = vec4(0.8,0.8,0.8,1.0);
   else if (intensity > 0.5)
      color = vec4(0.4,0.4,0.8,1.0);
   else if (intensity > 0.25)
      color = vec4(0.2,0.2,0.4,1.0);
   else
      color = vec4(0.1,0.1,0.1,1.0);
   fragColor = color;
}
```

Bump Mapping

Como o *bump mapping* consiste em perturbar a normal da superfície, deve-se ter uma função para coordenar essa perturbação. Para isso costuma-se usa um *normal map*, que nada mais é que uma textura que armazena informação sobre um vetor direção em cada pixel da imagem. A componente *red* de cada pixel armazena o valor do vetor direção no eixo *x*, o *green* para o *y* e o *blue* para o eixo *z*. Dessa forma, como os vetores normais geralmente são perpendiculares a superfície, há uma predominância da cor azul, como mostrado na seguinte figura.

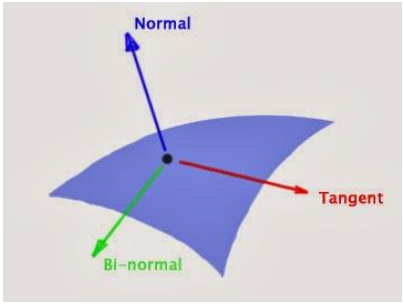


Considerando-se vetores normais normalizados, cada componente *xyz* varia entre [-1, 1]. Como a textura somente pode conter valores positivos, deve-se fazer uma conversão de valores, utilizando-se a seguinte equação: `cor = (normal+1)*127,5`. Isso leva os valores entre [0, 255]. No programa de shader, cada pixel é lido com valor entre [0, 1]. Fazendo-se `normal=(cor*2)-1` retoma-se o valor original da

normal. Existem diversos programas usados para fazer a conversão de uma imagem convencional em um *normal map*. Nessa conversão deve-se tomar a imagem original, obter um mapa de alturas e pela derivada de pixels vizinhos e encontrar o vetor normal de cada pixel, como mostrado na seguinte figura.

A aplicação desse mapa de normal em superfícies alinhadas aos eixos cartesianos é uma tarefa relativamente fácil (ver demo glsl_bump do site). Considerando-se, por exemplo, uma superfície no plano *xz* (chão), basta fazer as normais, que estavam no eixo *z*, apontar para o eixo *y* e aplicar a técnica de *bump-mapping*.

Caso a textura for aplicada em superfícies quaisquer, faz-se uso de um formalismo matemático mais elaborado. Uma vez que a normal de cada face pode apontar em qualquer direção, para cada normal deve-se fazer uma transformação no vetor direção da luz, isso porque os vetores normais não estão no mesmo espaço que o vetor da luz. Para fazer essa conversão, deve-se levar a luz e a câmera para um espaço chamado **espaço tangente**, que é definido por uma base ortonormal (vetores ortogonais e normalizados) de 3 vetores para cada vértice.



Um dos vetores é o próprio **vetor normal** da superfície (N), o outro é um **vetor tangente** (T) a superfície, e por último o **vetor binormal** (B – ou **vetor bitangente**) definido pelo produto vetorial dos outros dois vetores. Esses 3 vetores definem a matriz TBN, que é dada por

$$TBN = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix}$$

Tendo-se essa matriz, pode-se definir o vetor de luz no espaço tangente, multiplicando o vetor direção da luz relativo a cada vértice por essa matriz. Esse processo matemático é semelhante à uma transformação de lookAt.

Tanto os vértices, coordenadas de textura, normal, tangente e bi-tangente podem ser definidos por VBOs e passados para o programa de vértices. .

Para mais detalhes veja [1, 2, 3].

Referencias

[1] Jacobo Rodriguez Villar. TyphoonLabs' OpenGL Shading Language tutorials. Disponível em http://www.opengl.org/sdk/docs/tutorials/TyphoonLabs/

[2] http://www.paulsprojects.net/tutorials/simplebump/simplebump.html

[3] http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/