Programación 3D Tema 3 – OpenGL (Luces)

Master en Programación de Videojuegos

Javier Alegre Landáburu
javier.alegre@u-tad.com

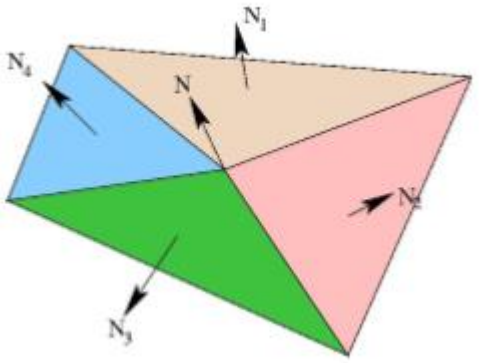
Introducción

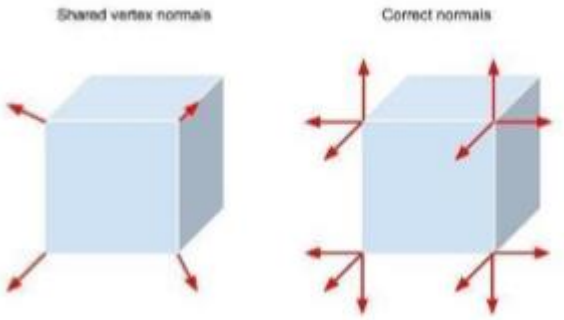
- Hasta ahora, el aspecto de un objeto dependía únicamente de la textura utilizada.
- Es habitual definir fuentes de luz, que influyen también en el aspecto final del objeto.
- Utilizando el fixed pipeline, disponemos de un máximo de 8 fuentes de luz en la escena.
- Utilizando shaders, debemos encargarnos de los cálculos de iluminación nosotros mismos, bien en shader de vértices (vertex lighting), bien en el de fragmentos (per-pixel lighting).
- En este tema, indicaremos las fórmulas que el fixed pipeline aplica para los cálculos de la iluminación, para facilitar su implementación en shaders.

Normales

- Cuando formamos mallas con varios triángulos, estos triángulos tendrán muchos lados en común, es decir, compartirán vértices.

- En OpenGL, las normales se definen a nivel de vértice, y se forman con la combinación de las normales de los triángulos que utilizan dichos vértices.

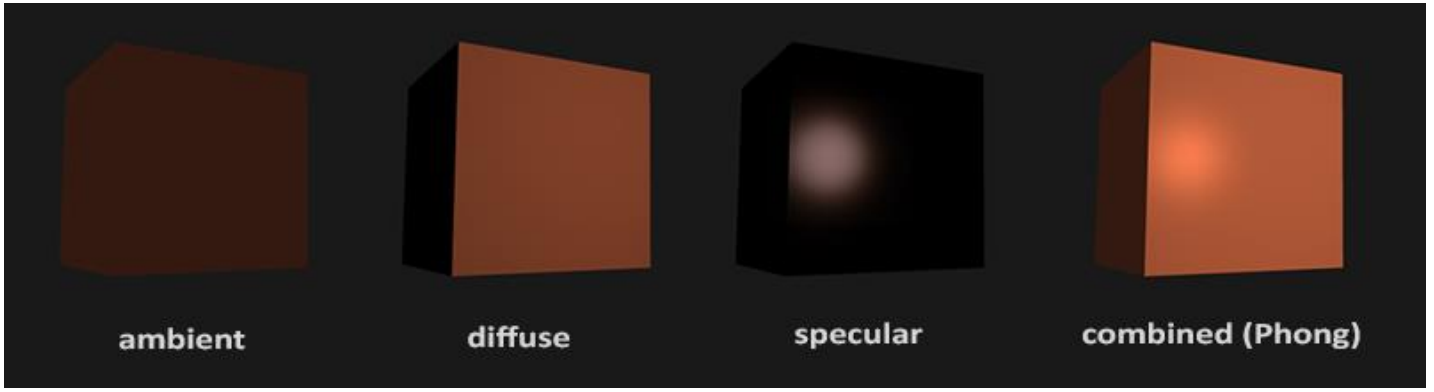




Coordenadas de textura

- Cuando calculamos la iluminación con OpenGL, debemos tener en cuenta los cuatro componentes de iluminación que influyen en el aspecto final del objeto: **ambiente**, **difuso**, **especular** y **emisivo**.

- La información de qué respuesta produce un objeto para cada uno de los componentes de iluminación se definirá en sus materiales.



Componentes de iluminación: ambiente

- Imaginemos una habitación con una mesa. Sobre la mesa, hay una tetera y una lámpara.

- La tetera no está completamente a oscuras siquiera en las partes donde la luz no da directamente. Esto es debido a que la luz rebota en las superficies (la mesa, las paredes) e incide de nuevo sobre la tetera, aunque con una menor intensidad (a esto se llama iluminación indirecta).

- Podríamos entender la luz ambiente como la luz residual que ha rebotado por las superficies de la escena y su origen no puede ser determinado.
-
- En esta asignatura, por simplificación, vamos a considerar que la luz ambiente es uniforme a toda la escena.

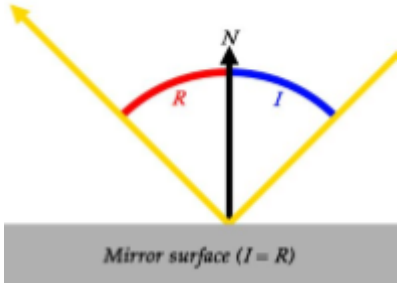
Componentes de iluminación: difuso

- Este componente define la forma en que el color emitido por la fuente de luz es reflejado por la superficie que lo recibe.

- Por ejemplo: una bombilla emite luz blanca, pero al chocar contra una taza con color rojo, la taza se ve de este color y no blanca. Esto es porque el material de la taza ha absorbido los componentes verde y azul de la luz, y ha reflejado el componente rojo.

Componentes de iluminación: especular

- Este componente es el que hace que una superficie parezca reflectante.
- La palabra especular proviene de “espejo”.
- Veamos cómo refleja un espejo la luz:

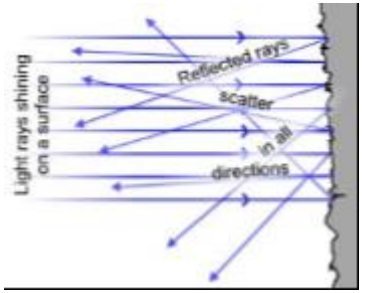


- N es la normal de la superficie.
- I es el ángulo con el que la luz incide.
- R es el ángulo con el que la luz se refleja.

Componentes de iluminación: especular

- En un espejo, el ángulo de incidencia es igual al de reflexión:

- En una superficie irregular, los rayos se dispersan con distintos ángulos (reflejo difuso)



Componentes de iluminación: emisivo

Este componente simula que el objeto emite luz (aunque realmente no introduce iluminación adicional a la escena).

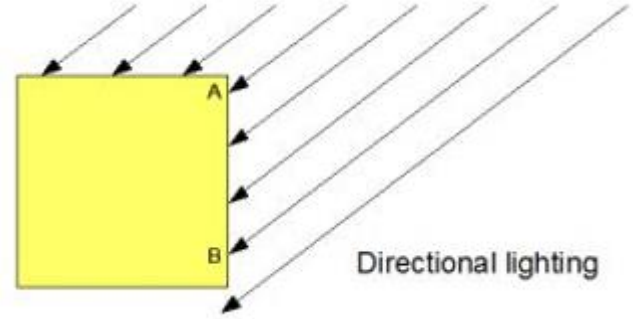
Una vez calculada la influencia de los otros componentes de iluminación, se suma el componente emisivo.

Direccional

La fuente de luz direccional está tan lejana a nosotros que se puede considerar en el infinito. Por ello, no tiene posición sino dirección, y no se ve atenuada con la distancia.

Los rayos de luz emitidos llegan a nosotros como vectores paralelos (todos con la misma dirección).

El ejemplo más claro de luz direccional es el sol.

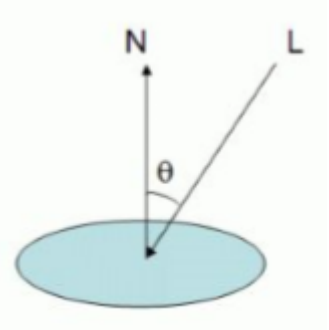


Direccional

- Empezamos por el componente difuso:

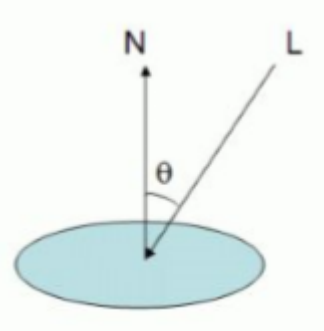
La luz difusa es percibida independientemente de la posición del espectador.

Su intensidad es proporcional al componente difuso de la luz, al coeficiente de reflexión difuso del material sobre el que incide, y al ángulo entre la dirección de la luz y la normal de la superficie.



Direccional

Podemos calcularlo de la siguiente forma:



Diffuse = LightColor * cos(θ)

Donde cos(θ) es el coseno del ángulo que forman N y L.

Esto se conoce como Reflexión Lambertiana o Ley de Beer-Lambert.

Debemos calcular con esta fórmula y sumar la aportación difusa de todas las fuentes de luz para obtener el difuso final.

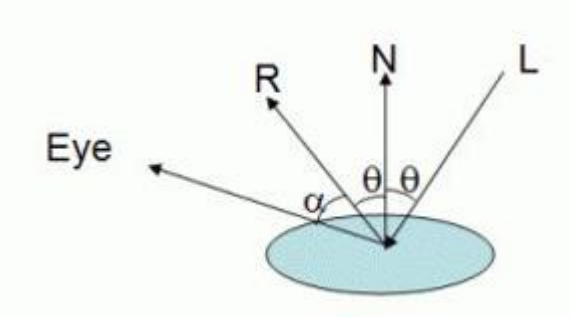
Direccional

Una vez hemos acumulado el difuso final, debemos sumarle el valor ambiente y el emisivo:

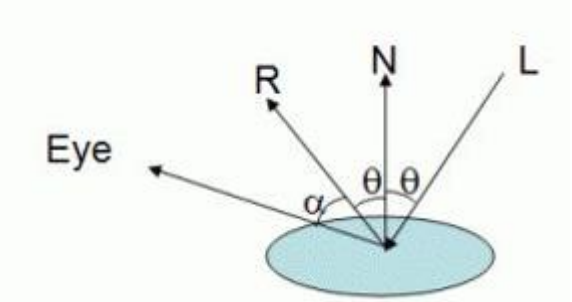
Lighting = Ambient + FinalDiffuse + Emissive

Y este valor de iluminación lo multiplicamos por el color de objeto y su textura.

Por último, hay que añadir el componente especular. Se puede utilizar el modelo de Phong, que en lugar de reflejar el entorno, produce brillos para simular el efecto:



Direccional



Según este modelo, el componente especular es proporcional al coseno del ángulo entre el vector de reflexión de la luz (R) y el vector con la dirección entre el vértice o fragmento de incidencia y el observador (Eye).

Si el vector Eye coincide con R, tenemos la máxima intensidad especular. Según ambos vectores divergen, la intensidad va disminuyendo.

Se utiliza un valor de brillo (shininess) para indicar la velocidad de disminución de la intensidad. Cuanto mayor sea, más rápido disminuye la intensidad.

Direccional

La fórmula para el vector de reflexión es: **R = -2N(L·N) + L**

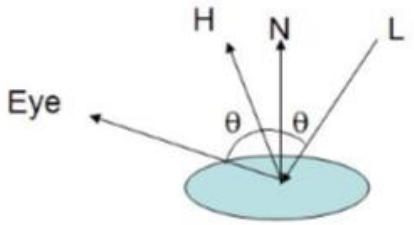
El especular se obtiene con:

Specular = (R·Eye)Shininess * SpecularColor

Direccional

Vamos a utilizar una versión simplificada de este modelo llamada Blinn-Phong, más rápida de calcular.

En lugar de R, se utiliza un vector llamado half vector (H), situado a mitad de camino entre los vectores Eye y L:



La fórmula de este vector es: H = Eye - L

El modelo Blinn-Phong se implementa: Specular = (N · H)Shininess * SpecularColor

Debemos sumar la aportación de todas las fuentes de luz para calcular el especular final.

La contribución final de la iluminación la calculamos con: gl_FragColor = Lighting * Material + FinalSpecular

Donde Material es la combinación de las propiedades del material (Color * Texture -si está presente-).

Puntual

Una luz puntual tiene una posición en el mundo, y emite en todas direcciones desde ese punto.

Al tener una posición, la intensidad de la luz puntual sí sufre atenuación con la distancia de la luz a la superficie.

Un ejemplo de luz puntual es una bombilla.

La diferencia fundamental entre la luz direccional y la luz puntual desde el punto de vista de OpenGL es que la puntual sufre atenuación con la distancia. Esta atenuación se especifica con tres coeficientes: constante, lineal, y cuadrático.

Puntual

El cálculo de la atenuación se realiza con la fórmula:

Att = 1 / (ConstantAtt + LinearAtt*Distance + QuadAtt*Distance2)

Donde Distance es la distancia de la fuente de luz a la superficie.

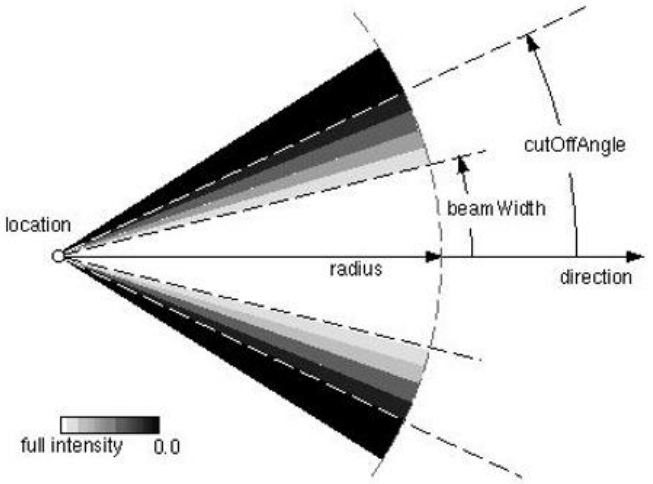
Por defecto, los valores de OpenGL para la atenuación constante, lineal y cuadrática son, respectivamente, 1, 0, 0.

La atenuación se debe multiplicar al cálculo de los componentes difuso y especular de cada fuente de luz.

Focal

- Una luz focal tiene una posición en el mundo, y emite en una determinada dirección especificada por un cono de emisión:
- El cono tiene una apertura dada por un ángulo.
- Y una tasa de atenuación que indica cómo la intensidad de la luz se reduce desde el centro del cono hacia el exterior.
- Un ejemplo de este tipo de luz es una linterna.
- Si el coseno del ángulo entre la dirección del cono y L es mayor al coseno del ángulo de apertura del cono, entonces estamos dentro del mismo.
- Se calcula la iluminación igual que con la luz puntual, pero además debemos multiplicar por este valor:

SpotEffect = (SpotDir·L)SpotAtt



Ejemplo Vertex en el que basarse

```
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

out vec3 FragPos;
out vec3 Normal;
out vec2 TexCoords;
out vec4 FragPosLightSpace;

void main() {
FragPos = vec3(model * vec4(aPos, 1.0));
gl_Position = projection * view * vec4(FragPos, 1.0);
TexCoords = aTexCoords;

Normal = mat3(transpose(inverse(model))) * aNormal;
}
```

Ejemplo Fragment en el que basarse

```
uniform sampler2D diffuse;
struct Material {
   sampler2D diffuse;
   sampler2D specular;
   float shininess;

vec3 ambientColor;
vec3 diffuseColor;
vec3 specularColor;
};

struct DirLight {
   vec3 direction;

   vec3 ambient;
   vec3 diffuse;
   vec3 specular;
};

in vec3 FragPos;
in vec3 Normal;
in vec2 TexCoords;

uniform vec3 viewPos;
uniform DirLight dirLight;
uniform Material material;
uniform int useTexture;

out vec4 FragColor;

vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir);


void main() {
vec3 norm = normalize(Normal);
vec3 viewDir = normalize(viewPos - FragPos);

vec3 result = CalcDirLight(dirLight, norm, viewDir);

FragColor = vec4(result, 1.0);
}

vec3 CalcDirLight(DirLight light, vec3 normal, vec3 viewDir)
{
   vec3 lightDir = normalize(-light.direction);
   // diffuse shading
   float diff = max(dot(normal, lightDir), 0.0);
   // specular shading
   vec3 reflectDir = reflect(-lightDir, normal);
   float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
   // combine results
vec3 ambient  = light.ambient  * material.ambientColor;
vec3 diffuse  = light.diffuse  * diff *  material.diffuseColor;
vec3 specular = light.specular * spec *  material.specularColor;
if (useTexture == 0)
{
ambient  = ambient  * vec3(texture(material.diffuse, TexCoords));
diffuse  = diffuse  * vec3(texture(material.diffuse, TexCoords));
specular = specular * vec3(texture(material.specular, TexCoords));
}
return (ambient + diffuse + specular);

}
```