

Iluminación

Informática Gráfica I

Material de: **Antonio Gavilanes y Ana Gil**
Adaptado por: **Elena Gómez y Rubén Rubio**
{mariaelena.gomez,rubenrub}@ucm.es



Contenido

1 Introducción

- Lighting models
- Componentes

2 Fuentes

- Posicionales
- Shading models

3 Comandos

- Fuentes
- Focos
- Atenuación

4 Materiales

5 Componentes

- Globales
- Locales vs Globales

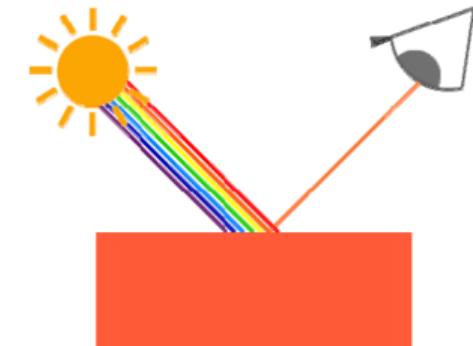
6 Implementación

- Light
- DirLight
- PosLight
- SpotLight
- Material
- EntityWithMaterial
- Otras

Luz y color

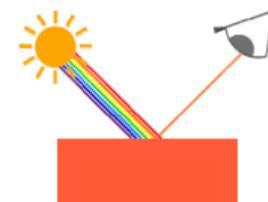
- El **color con el que vemos un objeto** depende de la luz que incide en la superficie: Parte de la luz es reflejada y parte es absorbida por el objeto.
- El color que percibimos son los colores reflejados.
- El color del objeto es el que percibimos con luz blanca: $(1, 1, 1)$

```
dvec3 lightColor(1.0, 1.0, 1.0);
dvec3 objColor(1.0, 0.5, 0.31);
dvec3 result = lightColor * objColor; // = (1.0, 0.5, 0.31);
dvec3 lightColor(0.0, 1.0, 0.0);
dvec3 objColor(1.0, 0.5, 0.31);
dvec3 result = lightColor * objColor; // = (0.0, 0.5, 0.0);
```



Luz y color

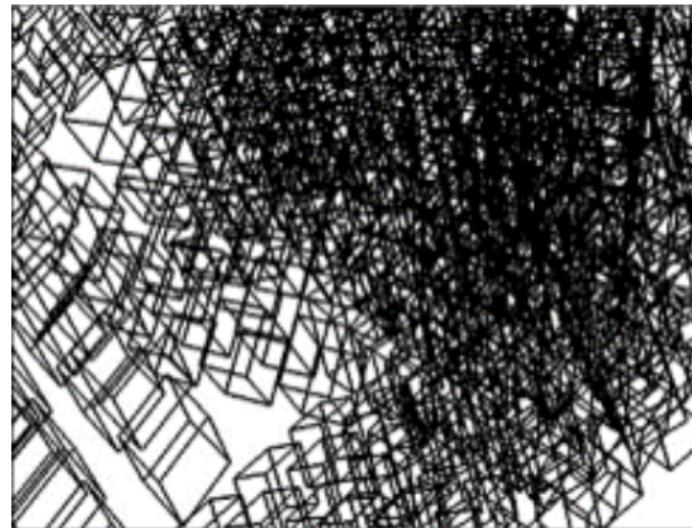
- El color con el que vemos un punto de un objeto se define en función de
 - La luz que incide en el punto:
 - Color e intensidad de la luz (RGBA)
 - Dirección de incidencia (vector)
 - Las propiedades del material (reflectancia):
 - Color (RGBA): porcentaje de la luz incidente que será reflejada (coeficiente de reflexión)
 - Brillo: La forma en que se refleja
 - Del punto de vista (cámara), pues la luz se refleja:
 - Uniformemente en todas las direcciones
 - En dirección especular con la de incidencia



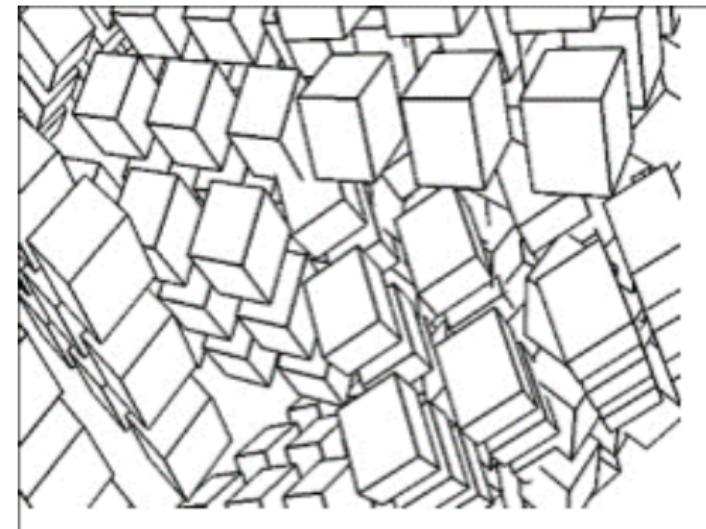
Renderización

- Renderización de escenas
- Jerarquía en los niveles de realismo

Renderización basada en armazón de hilos

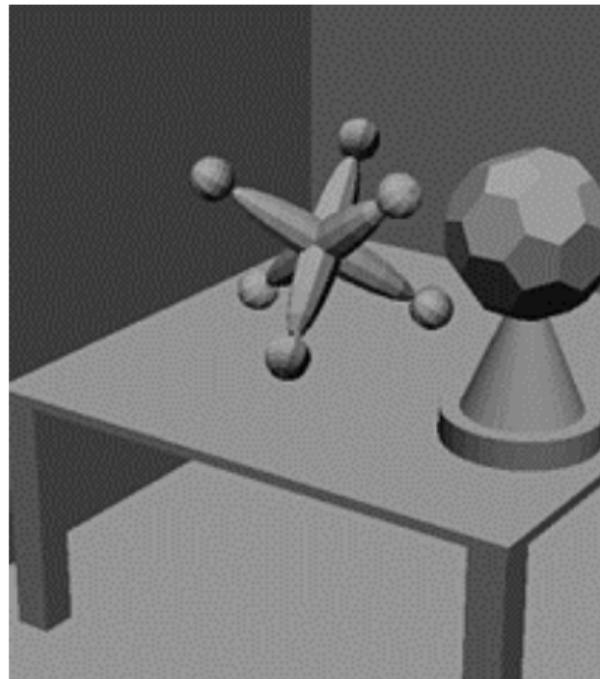


Renderización basada en armazón de hilos con eliminación de superficies ocultas

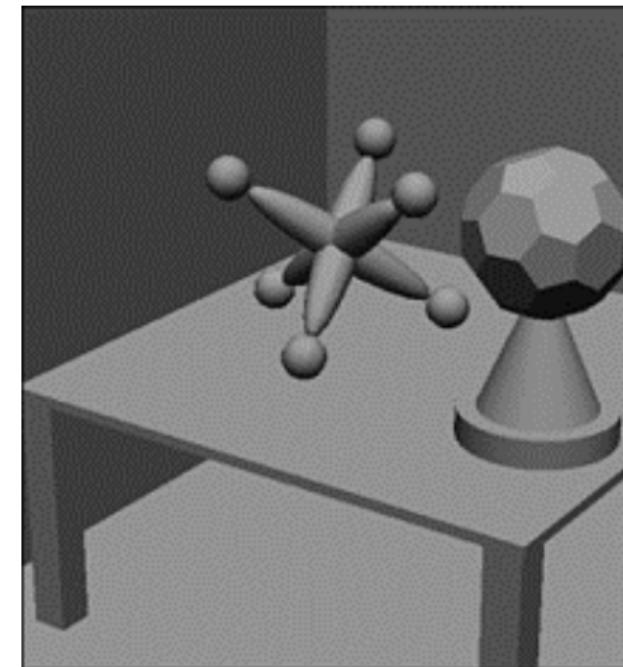


Renderización

Iluminación plana (*flat shading*)

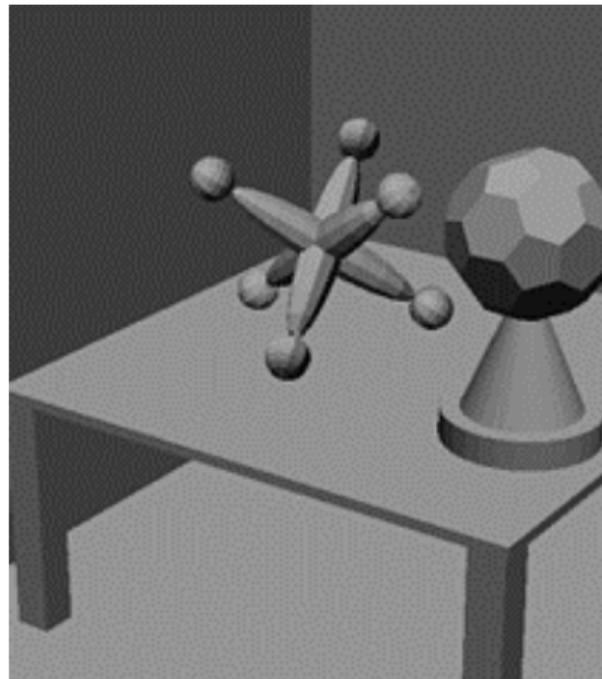


Iluminación suave (*smooth shading*)

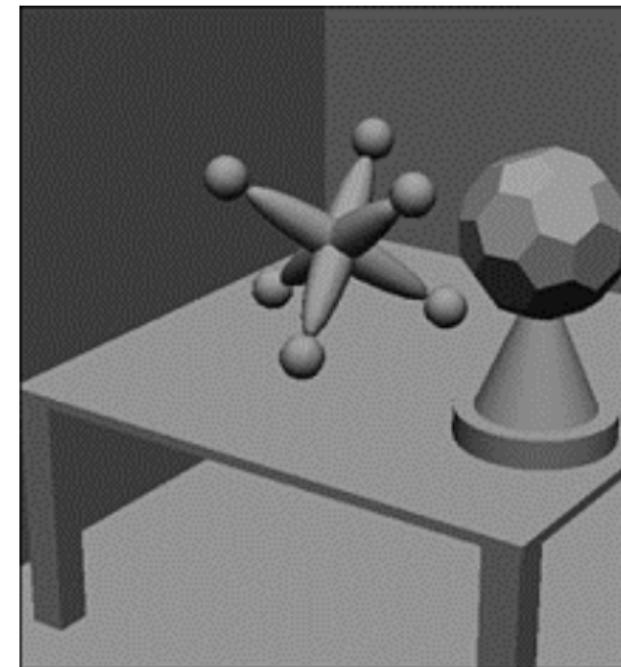


Renderización

Iluminación plana (*flat shading*)

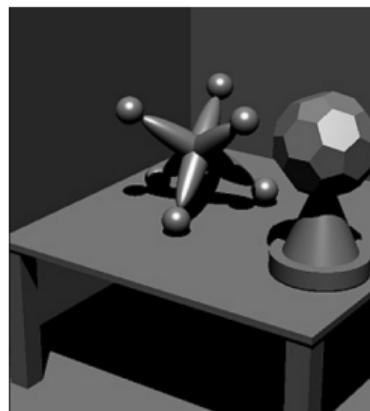


Iluminación suave (*smooth shading*)

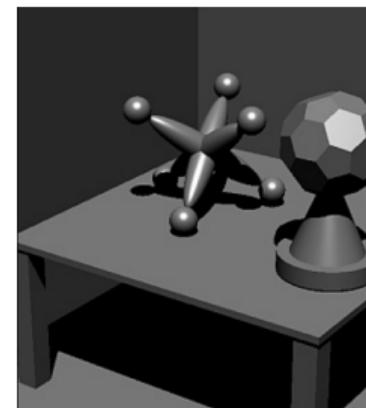


Renderización

Iluminación con efectos especulares



Iluminación con sombras

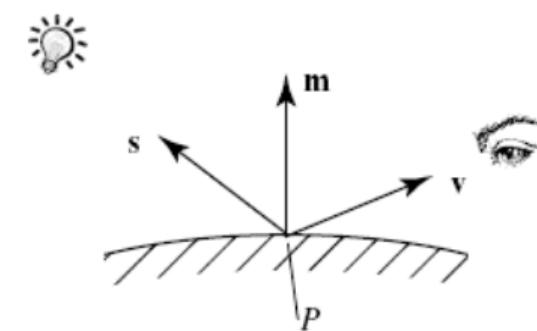


Iluminación con texturas



Lighting models

- Un modelo de iluminación dicta cómo se refleja y dispersa la luz, cuando incide sobre una superficie.
- Simplificación con respecto al modelo físico de propagación de la luz (absorción de luz, transmisión de la luz a través de los objetos, intensidad de la luz emitida).
- Fenómenos principales que se tienen en cuenta cuando la luz incide en una superficie:
 - La **dispersión difusa**, responsable del color de la superficie
 - El **reflejo especular**, responsable del material de la superficie
- Ingredientes geométricos:
 - **Vector** normal a la superficie en el punto $P(m)$
 - **Vector** desde P al ojo de la cámara (v)
 - **Vector** desde P a la fuente de luz (s)



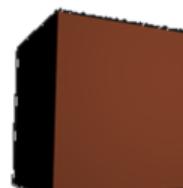
Phong Lighting Model

- La luz reflejada por una superficie consta de una combinación de tres componentes: **difusa, especular** y **ambiente**
- Las componentes difusa y ambiente representan el color del material
- La componente especular representa los reflejos del material
- Tanto el color y la intensidad de la luz, como la reflectancia del material se definen para cada una de las tres componentes, mediante un valor RGBA:

```
diffLightColor / diffMaterialColor,  
specLightColor / specMaterialColor,  
ambLightColor / ambMaterialColor
```



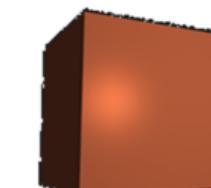
ambiente



difusa



especular



combinadas

Componente Difusa

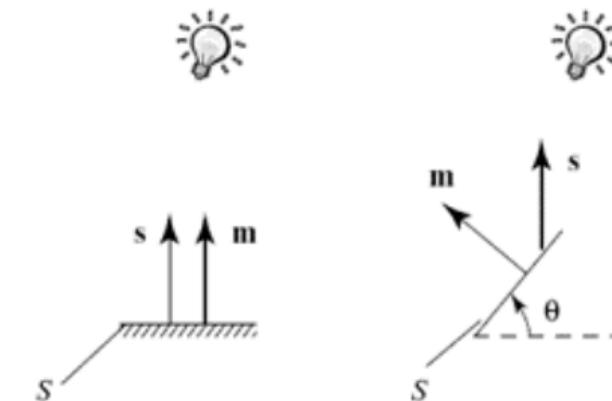
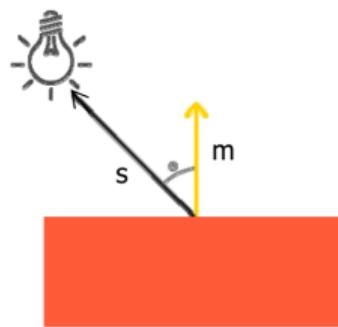
- **Dispersión difusa**

- Luz que, habiendo incidido sobre una superficie, se vuelve a irradiar desde ella, en todas las direcciones (independiente del punto de vista)
- Su intensidad depende del ángulo θ que forman los vectores s y m .
- Para su implementación se usa el modelo de **Lambert**:

```
dvec3 diffuse = max(0, cos Theta) * diffLight * diffMaterial
```

donde $\cos\theta = \text{dot}(m, s)$ (si los vectores están normalizados)

- Para fuentes de luz lejanas, el vector s varía poco, luego la componente difusa cambia poco a lo largo de la superficie



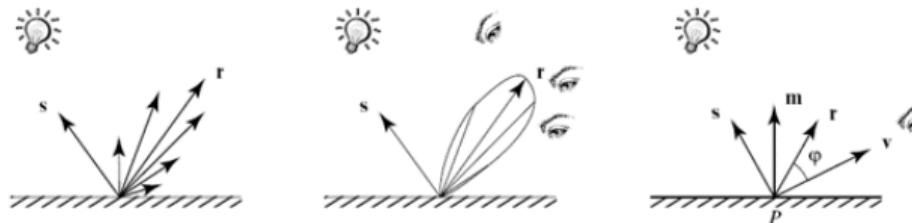
Componente Especular

• Reflexión especular

- Luz refleja la superficie
- El vector de máxima reflexión $r=reflect(s, m)$ coincide con el rayo reflejado. La intensidad de la luz reflejada depende pues del ángulo que forman los vectores r y v .
- Para su implementación se usa el modelo de **Phong**:

```
dvec3 specular = max(0, cos^f Theta) * specLight * specMaterial
```

donde $\cos^f \theta = \text{dot}(r, v)$ (si los vectores están normalizados) y $f = 1, \dots, \infty$ (espejo) es un atributo del material

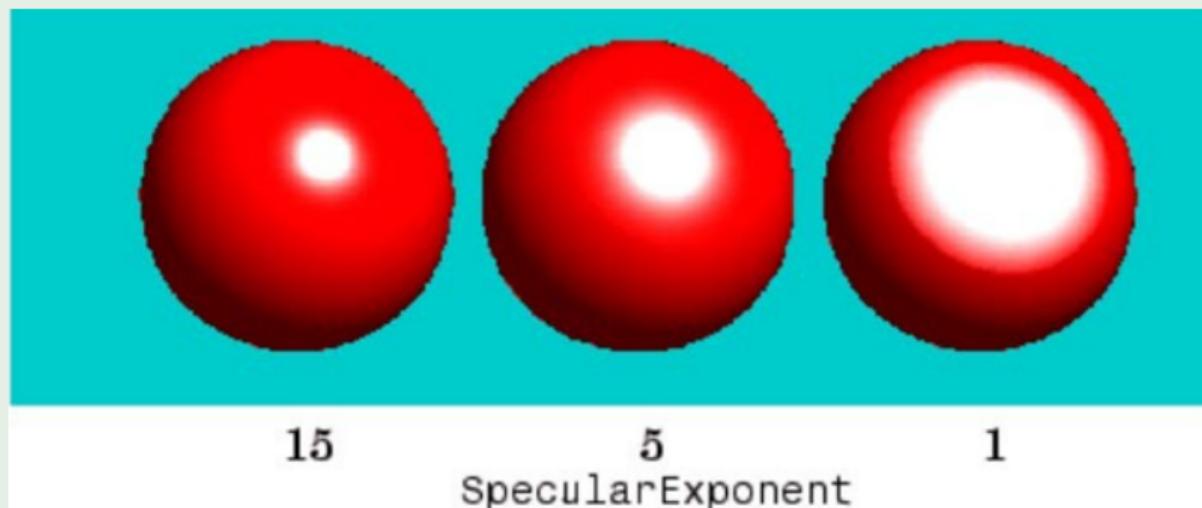


Para fuentes de luz lejanas, los vectores r y s varían mucho, luego la componente especular cambia mucho a lo largo de la superficie.

Componente Especular

```
dvec3 specular = max(0, cos^f Theta) * specLight * specMaterial
```

Valores de f



Componente Ambiente

• Ambiente de las fuentes de luz

- Luz que rebota en numerosas superficies de la escena
- Luz que alcanza una superficie aunque no esté expuesta a la fuente de luz.
- Es una luz que ilumina a todos los objetos por igual.
- Se implementa siguiendo un modelo simple:

```
dvec3 ambient = ambLight * ambMaterial
```

Resultado final

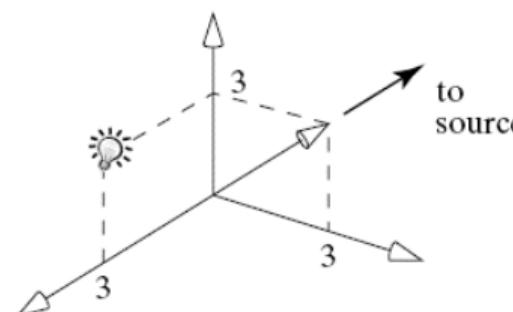
- El color resultante de una fuente de luz es la combinación de las tres componentes obtenidas:

```
dvec3 finalColor = diffuse + specular + ambient
```

- En una escena pueden haber varias fuentes de luz
- El color final de un punto de la superficie de un objeto es la suma de los colores finales por cada fuente de luz, teniendo en cuenta además otros factores como la atenuación de las fuentes, la emisividad del material, la pertenencia a conos de luz.

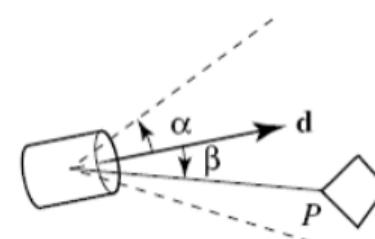
Fuentes de luz en OpenGL

- Las fuentes de luz en OpenGL son de dos tipos
 - **Direccionales** (o remotas): se supone que están infinitamente lejos y que, por tanto, sus rayos de luz llegan paralelos a la escena. Por su lejanía infinita se supone que no se atenúan. Se especifican mediante el vector que va del origen a la fuente de luz.
 - **Posicionales** (o locales): se supone que están localizadas en un punto y que, por tanto, los objetos de la escena son más o menos iluminados según su exposición. Se especifican mediante el punto en el que se encuentra la fuente de luz. En estos casos, el vector `s` en un punto del objeto se obtiene por `s = (lightPosition-pointPosition).normalize()` y se puede conocer también la distancia de la fuente al punto por lo que estas fuentes admiten atenuación.



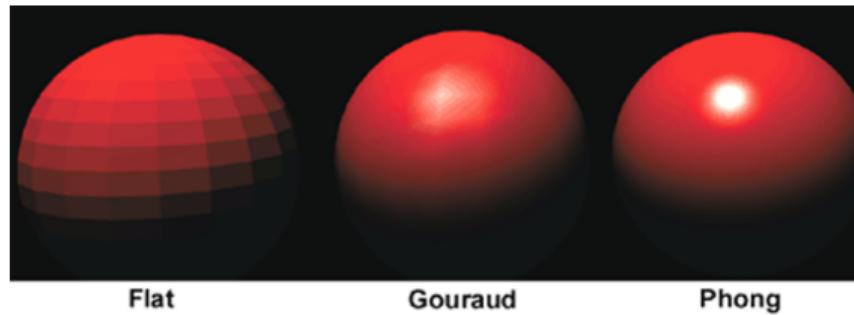
Fuentes de luz en OpenGL

- En OpenGL se admiten dos tipos de fuentes de luz posicionales:
 - Omnidireccionales**: El resto de las fuentes de luz locales, que se caracterizan por emitir luz por igual, en todas direcciones.
 - Focos**: Están caracterizados por una dirección de emisión (el vector d), y una amplitud de emisión (el ángulo α). Los puntos que se encuentran fuera del cono de emisión de luz que determinan estos dos parámetros no reciben luz del foco.
Los puntos que se encuentran dentro del cono de emisión reciben una cantidad de luz que varía según el ángulo β en un factor que es una potencia ε del $\cos\beta$. Cuanto mayor es ε , más se concentra la luz del foco.



Shading models

- Modelo de matizado o tonalidad (shading model)
 - **Flat Shading:** Todos los fragmentos del interior de un triángulo se colorean igual. Los tres vértices del triángulo tienen el mismo vector normal: la **normal al triángulo**.
 - **Gouraud Shading:** los vértices del triángulo tiene distinto vector normal: la **normal a la superficie que aproxima**. Se calcula el color para cada uno de los tres vértices y el color de cada fragmento se obtiene por interpolación de los colores de los vértices.
 - **Phong Shading:** los vértices del triángulo tiene distinto vector normal, el calculo del color se realiza para cada fragmento.



Shading models en OpenGL

- El *shading model* en OpenGL se especifica con los comandos:

```
glShadeModel(GL_FLAT);      // Para el matizado plano  
glShadeModel(GL_SMOOTH);   // Para el matizado suave (o de Gouraud)  
                            // Es el que se activa por defecto
```

Existe también el modelo de matizado de Phong, solo en las versiones de OpenGL 2.x y posteriores, que requiere ser definido por el usuario.

Comandos para tratar con luces en OpenGL

- El modo de iluminación se activa/desactiva con los comandos:

`glEnable(GL_LIGHTING)/glDisable(GL_LIGHTING)`

- OpenGL permite definir hasta `GL_MAX_LIGHTS` fuentes de luz

`GL_LIGHT0 , GL_LIGHT1 , ... , GL_MAX_LIGHTS -1`

cumpliéndose que `GL_LIGHTi = GL_LIGHT0 + i.`

- Cada fuente de luz de OpenGL tiene un identificador (`id`) interno (que se guarda en la tarjeta gráfica GPU)

- El identificador es un entero (`GLuint`) que varía sobre el rango

`GL_LIGHT0,..., GL_LIGHT0 + GL_MAX_LIGHTS - 1`, donde `GL_LIGHT0` y `GL_MAX_LIGHTS` son dos constantes `GLuint`.

- Una fuente de luz particular (con identificador `id`) se enciende/apaga con los comandos:

`glEnable(id)/glDisable(id)`

Comandos para tratar con luces en OpenGL

- El cálculo del coseno del ángulo formado por dos vectores, necesario en el manejo de las componentes difusa y especular de las luces, requiere que los vectores estén normalizados. El comando:

```
glEnable(GL_NORMALIZE);
```

- El método `setGL()` de la clase `Scene` define el entorno para manejar las luces en la escena:

```
// Se pone el fondo azul
glClearColor(0.7, 0.8, 0.9, 1.0);
// Se activa el test de profundidad
glEnable(GL_DEPTH_TEST);
// Se activan las texturas, por si las hay
glEnable(GL_TEXTURE_2D);
// Se activa la iluminación
glEnable(GL_LIGHTING);
// Se activa la normalización de los vectores normales
glEnable(GL_NORMALIZE);
```

Fuentes de luz en OpenGL

- La posición de la fuente de luz 0 se define con el comando:

```
glm::fvec4 v={3.0, 2.0, 1.0, 1.0};  
glLightfv(GL_LIGHT0, GL_POSITION, value_ptr(v));
```

- Las coordenadas de una fuente de luz posicional son mundiales.
- La forma en que OpenGL distingue si una fuente de luz es remota o local es por la cuarta componente de la posición con que se especifica. Si es distinto de 0, la fuente es local; si es 0, es remota.
- Por defecto, todas las fuentes de luz tienen posición (0,0,1,0), es decir, son direccionales y miran a la parte negativa del eje Z.

Fuentes de luz en OpenGL

- Para definir en OpenGL las componentes difusa, especular y ambiente de la fuente de luz 0 se usan los comandos:

```
glm::fvec4 amb0={0.2, 0.4, 0.6, 1.0};  
glm::fvec4 dif0=...;  
glm::fvec4 esp0=...;  
glLightfv(GL_LIGHT0, GL_AMBIENT, value_ptr(amb0));  
glLightfv(GL_LIGHT0, GL_DIFFUSE, value_ptr(dif0));  
glLightfv(GL_LIGHT0, GL_SPECULAR, value_ptr(esp0));
```

- Los valores por defecto son los siguientes:
 - (0,0,0,1) para la componente ambiente de todas las luces (representa la oscuridad)
 - (1,1,1,1) para las componentes difusa y especular de la luz **GL_LIGHT0** (representa el mayor brillo)
 - (0,0,0,1) para las componentes difusa y especular de las restantes luces.

Fuentes de luz en OpenGL

- La única luz hasta ahora en la práctica es la que aparece en el método de `Scene` `sceneDirLight(Camera const&cam)`.

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glm::fvec4 posDir = { 1, 1, 1, 0 };
glMatrixMode(GL_MODELVIEW);
glLoadMatrixd(value_ptr(cam.viewMat()));
glLightfv(GL_LIGHT0, GL_POSITION, value_ptr(posDir));
glm::fvec4 ambient = { 0, 0, 0, 1 };
glm::fvec4 diffuse = { 1, 1, 1, 1 };
glm::fvec4 specular = { 0.5, 0.5, 0.5, 1 };
glLightfv(GL_LIGHT0, GL_AMBIENT, value_ptr(ambient));
glLightfv(GL_LIGHT0, GL_DIFFUSE, value_ptr(diffuse));
glLightfv(GL_LIGHT0, GL_SPECULAR, value_ptr(specular));
```

Es una luz llamada `GL_LIGHT0` que es direccional (la cuarta componente de la posición es 0) y cuyos rayos llegan a la escena en una dirección $(-1, -1, -1)$ (el vector del origen a la fuente de luz es $(1, 1, 1)$), de color blanco (componente difusa), de color negro en los objetos no enfrentados a ella (componente ambiente), que proporciona poca reflexión (componente especular). Además la fuente de luz es fija pues su posición se determina una vez con respecto a la posición de la cámara).

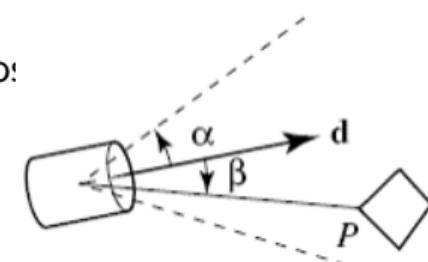
Focos de luz en OpenGL

- Los comandos de OpenGL para especificar un foco en la luz 0 son los siguientes (los valores que aparecen son ejemplos):

```
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 4.0);
glm::fve3 dir={2.0, 1.0, -4.0};
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, value_ptr(dir));
```

que especifican un foco con $\alpha = 45$, $\epsilon = 4$ y $d = (2, 1, -4)$.

- Por defecto, $\alpha = 180$, $\epsilon = 0$ y $d = (0, 0, -1)$, que supone que el foco apunta al lado negativo del eje Z, es omnidireccional y la luz se distribuye uniformemente por todo el cono de emisión.
- Los focos, como luces posicionales que son, admiten ser atenuados:



Atenuación de las fuentes de luz locales en OpenGL

- En OpenGL se puede definir la atenuación de las fuentes de luz locales especificando el factor de atenuación.

$$\text{factor atenuación} = \frac{1}{k_c + k_l d + k_q d^2}$$

k_c = `GL_CONSTANT_ATTENUATION`

k_l = `GL_LINEAR_ATTENUATION`

k_d = `GL_QUADRATIC_ATTENUATION`

donde d es la distancia.

Atenuación de las fuentes de luz locales en OpenGL

- En OpenGL se pueden definir las tres constantes de atenuación mediante los siguientes comandos:

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, ...);  
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, ...);  
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, ...);
```

- Por defecto, $kc = 1$, $kl = 0$, $kq = 0$, luego, para estos valores, el factor de atenuación es 1 y, por tanto, no habrá atenuación. En realidad, la atenuación de luces remotas es posible, pero no lo es cambiar los valores por defecto de las constantes.

Materiales

- OpenGL simula materiales por la forma en que reaccionan ante la luz roja, verde y azul en cada componente (difusa, especular, ambiente) de la fuente de luz. Por ejemplo, en cuanto al color, una superficie con componentes difusa y ambiente verde (0, 1, 0) bajo una luz con componentes difusa y ambiente roja (1, 0, 0) resulta ser $(0 \times 1, 1 \times 0, 0 \times 0) = (0, 0, 0)$
- En consecuencia, en OpenGL, los materiales se especifican con las mismas tres componentes (ambiente, difusa y especular) que las fuentes de luz

Materiales

- En el caso de los materiales, las componentes se llaman coeficientes de reflexión o reflectancias del material
- Cada coeficiente del material indica cómo reacciona la superficie ante la correspondiente componente de cada fuente de luz. Por ejemplo, una superficie con coeficiente de reflexión especular alto parecerá brillante bajo una fuente de luz con componente especular alta
- Como los coeficientes de reflexión ambiente y difuso son los responsables del color, y el color de una entidad no cambia se enfrente a una luz o no, en los materiales estos coeficientes se suelen tomar iguales
- Como el coeficiente especular solo es responsable de los reflejos que tiene la superficie, en los materiales este coeficiente se suele tomar gris de forma que el color de la superficie no se vea alterado por el color de la luz incidente

Materiales

- Coeficientes de reflexión para simular materiales comunes (junto con el valor del exponente para usarse en la componente especular):

Material	ambient $\rho_a = \rho_{ag} \rho_{ab}$	diffuse $\rho_d = \rho_{dg} \rho_{db}$	specular $\rho_s = \rho_{sg} \rho_{sb}$	exponent f
Black Plastic	0.0 0.0 0.0	0.01 0.01 0.01	0.50 0.50 0.50	32
Brass	0.329412 0.223529 0.027451	0.780392 0.568627 0.113725	0.992157 0.941176 0.807843	27.8974
Bronze	0.2125 0.1275 0.054	0.714 0.4284 0.18144	0.393548 0.271906 0.166721	25.6
Chrome	0.25 0.25 0.25	0.4 0.4 0.4	0.774597 0.774597 0.774597	76.8
Copper	0.19125 0.0735 0.0225	0.7038 0.27048 0.0828	0.256777 0.137622 0.086014	12.8
Gold	0.24725 0.1995 0.0745	0.75164 0.60648 0.22648	0.628281 0.555802 0.366065	51.2
Pewter	0.10588 0.058824 0.113725	0.427451 0.470588 0.541176	0.3333 0.3333 0.521569	9.84615
Silver	0.19225 0.19225 0.19225	0.50754 0.50754 0.50754	0.508273 0.508273 0.508273	51.2
Polished Silver	0.23125 0.23125 0.23125	0.2775 0.2775 0.2775	0.773911 0.773911 0.773911	89.6

Materiales

- El comando de OpenGL para definir el material es el siguiente:

```
glMaterialfv(cara, componenteDeLaLuz, value_ptr(a));
```

donde `cara` puede tomar alguno de los siguientes valores:

- `GL_FRONT`, para definir coeficientes de reflexión (ambiente, difusa o especular) para caras frontales
- `GL_BACK`, para caras traseras
- `GL_FRONT_AND_BACK`, para caras frontales y traseras

`a` es un vector con tres componentes RGB y una componente α , para especificar los coeficientes de reflexión:

```
glm::fvec4 a = {..., ..., ..., 1.0};
```

Materiales

- `componenteDeLaLuz` puede ser:

- `GL_AMBIENT`, para definir la componente ambiente
- `GL_DIFFUSE`, para definir la componente difusa
- `GL_SPECULAR`, para definir la componente especular
- `GL_AMBIENT_AND_DIFFUSE`, para definir iguales y a la vez las componentes ambiente y difusa
- `GL_EMISSION`, para simular objetos que emiten luz. Su luz no ilumina, sólo los hace parecer más brillantes. El valor que tiene por defecto es $(0, 0, 0, 1)$
- `GL_SHININESS`, para hacer parecer más brillante la superficie de un objeto. Es un valor entre 0 y 128. El valor 128 da aspecto metálico. Es el exponente que se usa como potencia en la reflexión especular

El modo ColorMaterial

- ¿Cómo reaccionan las entidades a las fuentes de luz?
- Hasta ahora se hacía estableciendo lo que se llama el *registro de color*
- El código para definir los materiales usando el registro de color es lo que aparece en el método `render()` de las entidades

```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(cara, componenteDeLaLuz);
    // cara es GL_FRONT, GL_BACK o ambas,
    // y componenteDeLaLuz es GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, ...
glColor3f..., ...);
mesh->render();
GlColor3f(1.0, 1.0, 1.0);
glDisable(GL_COLOR_MATERIAL);
```

Componentes globales del modelo de luz en OpenGL

- Color de la luz ambiente global

```
GLfloat amb[] = {..., ..., ..., 1.0};  
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, amb);
```

- El valor por defecto es (0.2, 0.2, 0.2, 1.0). La escena está pues siempre algo iluminada, a menos que no haya fuentes de luz y esta componente ambiente global se haya definido negra, en cuyo caso todo se verá negro salvo el color de fondo.
- Punto de vista de la cámara para el cálculo de la reflexión especular (local o remoto)

```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_FALSE);
```

El segundo parámetro es un booleano que especifica si el ojo es remoto o no. Por defecto es `false`. Si es `true`, el vector `v` al ojo que se tiene en cuenta en la reflexión especular pasa a ser $(0, 0, 1)$ siempre.

Componentes globales del modelo de luz en OpenGL

- Sombreado de ambos lados de las caras de una malla

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

El valor por defecto es `GL_FALSE` y OpenGL solo ilumina caras frontales. Si se prevé que se van a mostrar también caras traseras, se usa `GL_TRUE`. OpenGL utiliza entonces, como vector normal de una cara trasera, el opuesto del de la cara frontal.

Componentes globales del modelo de luz en OpenGL

- Aplicación de la luz especular antes de las texturas o no

```
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,  
              GL_SEPARATE_SPECULAR_COLOR);
```

que hace que OpenGL:

- mantenga dos colores para cada vértice: primario (obtenido como se ha explicado, sin tener en cuenta componentes especulares) y secundario (obtenido teniendo en cuenta componentes especulares)
- combine la textura solo con el color primario, si hace el caso
- añada después el color secundario.

En OpenGL 1.4 y superiores el color secundario se puede fijar con

```
glSecondaryColor3f(..., ..., ...);
```

No tiene componente alfa. Se activa con `glEnable(GL_COLOR_SUM);`

Iluminación en OpenGL

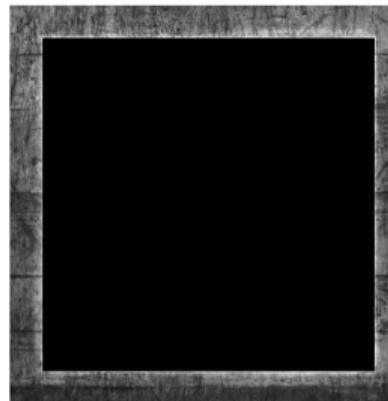
- OpenGL realiza los **cálculos de iluminación para cada uno de los vértices** de la primitiva gráfica. Los **resultados se interpolan** para colorear los fragmentos del interior. La calidad depende, en gran medida, del **número de vértices de la malla**.
- OpenGL realiza los **cálculos de iluminación en coordenadas de la cámara**. Una vez transformados los elementos geométricos (puntos y vectores) por la matriz de modelado y vista.
- Los **elementos geométricos de las fuentes de luz** (posición o dirección de la luz) también tienen que ser transformados por la **matriz de modelado y vista**. Hay que establecer las luces antes que los objetos a los que pueden iluminar.

Iluminación en OpenGL

- En las nuevas versiones de OpenGL la iluminación se puede programar a nivel de vértices o de fragmentos.
- Se pueden utilizar dos texturas simultáneamente para ampliar la definición del material de los objetos a nivel de fragmentos: **lighting maps**
 - Diffuse map: texturas de coeficientes de reflexión difuso
 - Specular map: textura de coeficientes de reflexión especular
- Se pueden utilizar otros datos, por ejemplo, para incorporar rugosidad al material se utiliza una textura de vectores normales (esta técnica requiere el uso de vectores tangentes).

Iluminación en OpenGL

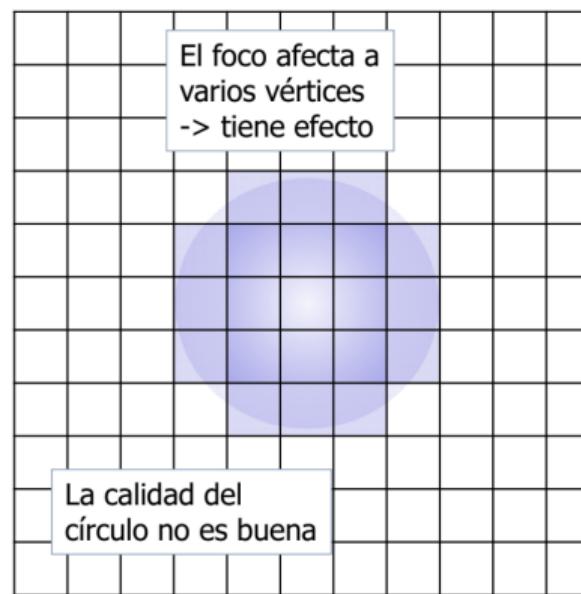
- Diffuse map: Una textura que define, por fragmento, el coeficiente de reflexión difuso. Sustituye a los coeficientes difuso y ambiente del material.
- Specular map: Una textura, de colores grises, que define, por fragmento, el coeficiente de reflexión especular.



Specular map: Los bordes son de metal y brillan (grises), el interior es de madera y no produce brillos (negro).

Iluminación en OpenGL

- La calidad del sombreado de Gourard (*smooth*) depende, en gran medida, del número de vértices de la malla.



Rejilla (11 × 11)



Rectángulo

Modelos de iluminación locales vs globales

● Modelo local

- El color de un vértice depende del material y de las fuentes de luz
- No se tienen en cuenta sombras ni luces reflejadas
- Solo interacción objeto-fuentes de luz
- Fenómenos ópticos comunes (sombras, reflejos, caústicas) difíciles de reproducir

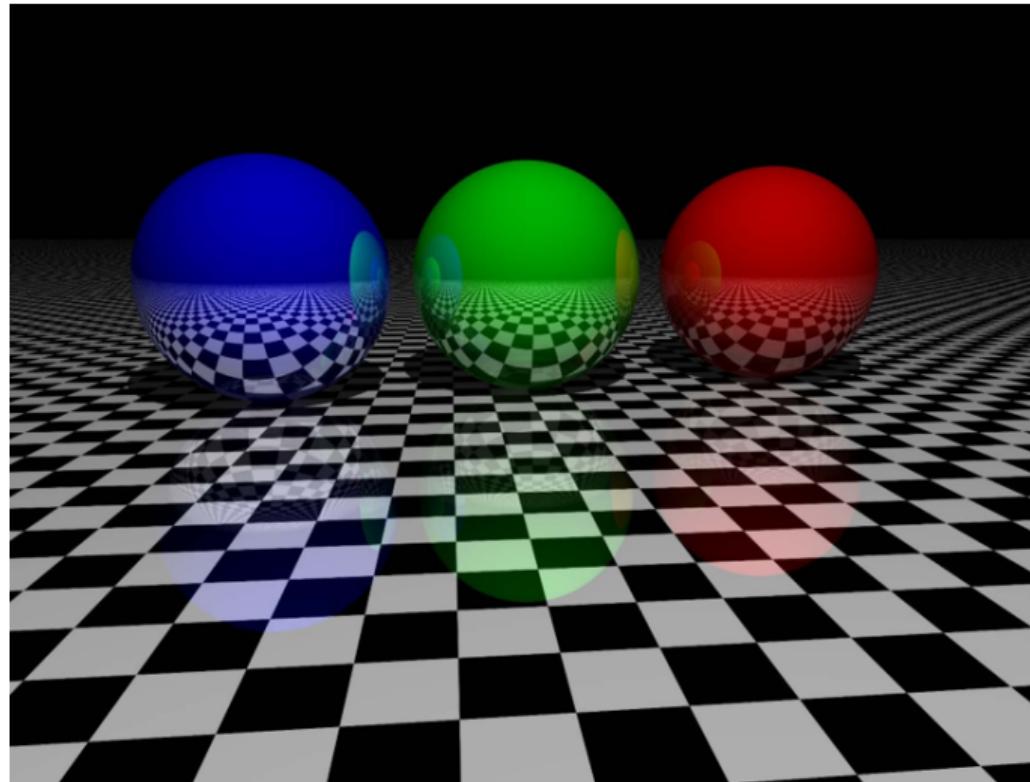
● Modelo global

- El color de un vértice depende del material y de la luz que le llegue de las fuentes de luz
- Interacción objeto-fuentes de luz y también objeto-objeto
- Fenómenos ópticos comunes no son tan costosos de reproducir
- Ray tracing, radiosidad

Caústicas



Reflejos



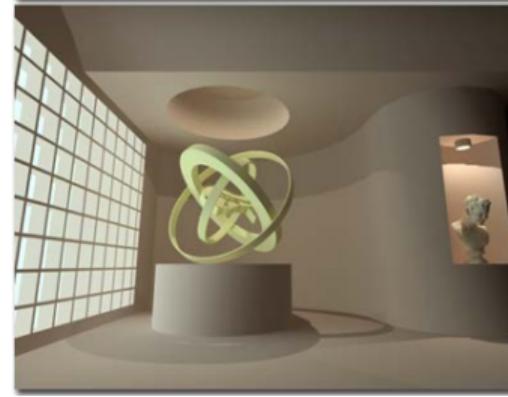
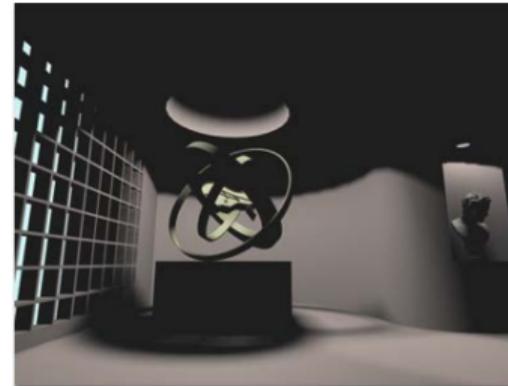
Sombras, reflejos y cáusticas



Modelo de iluminación por radiosidad



Modelo de iluminación por radiosidad



Modelo de iluminación por *ray tracing*



Modelo de iluminación por *ray tracing*



La clase Light

```
class Light { // Abstract class
protected:
    static GLuint cont; // Atributo para poder generar un nuevo id cada vez
    GLuint id = GL_LIGHT0 + GL_MAX_LIGHTS; // Primer id no válido
    // Atributos lumínicos y geométrico de la fuente de luz
    glm::fvec4 ambient = {0.1, 0.1, 0.1, 1};
    glm::fvec4 diffuse = {0.5, 0.5, 0.5, 1};
    glm::fvec4 specular = {0.5, 0.5, 0.5, 1};
    glm::fvec4 posDir = {0, 0, 1, 0};
    // Añade setters para cambiar el valor de los atributos lumínicos
public:
    Light();
    virtual ~Light() { disable(); }
    void uploadL();
    // Método abstracto
    virtual void upload(glm::dmat4 const& modelViewMat) const = 0;
...};
```

La clase Light

```
GLuint Light::cont = 0;  
Light() {  
    if (cont < GL_MAX_LIGHTS) {  
        id = GL_LIGHT0 + cont;  
        ++cont;  
        glEnable(id);  
    }  
};
```

La clase Light

```
void Light::uploadL() {
    // Transfiere las características de la luz a la GPU
    glLightfv(id, GL_AMBIENT, value_ptr(ambient));
    glLightfv(id, GL_DIFFUSE, value_ptr(diffuse));
    glLightfv(id, GL_SPECULAR, value_ptr(specular));
}

void disable() {if (id<GL_LIGHT0+GL_MAX_LIGHTS) glDisable(id);}
void enable() {if (id<GL_LIGHT0+GL_MAX_LIGHTS) glEnable(id);}
void setAmb(glm::fvec4 amb){
    ambient = amb; upLoadL();
}; //setDiff(), setSpec()
```

La clase DirLight

```
class DirLight : public Light {  
public:  
    virtual void upload(glm::dmat4 const& modelViewMat) const;  
    void setPosDir(glm::fvec3 dir);  
};  
  
void DirLight::upload(glm::dmat4 const& modelViewMat) const {  
    glMatrixMode(GL_MODELVIEW);  
    glLoadMatrixd(value_ptr(modelViewMat));  
    glLightfv(id, GL_POSITION, value_ptr(posDir));  
    uploadL();  
}  
// Ojo al 0.0 que determina que la luz sea remota  
void DirLight::setPosDir(glm::fvec3 dir) {  
    posDir = glm::fvec4(dir, 0.0);  
}
```

La clase PosLight

```
class PosLight : public Light {  
protected:  
    // Factores de atenuación  
    GLfloat kc = 1, kl = 0, kq = 0;  
public:  
    virtual void upload(glm::dmat4 const& modelViewMat) const;  
    void setPosDir(glm::fvec3 dir);  
    void setAtte(GLfloat kc, GLfloat kl, GLfloat kq);  
};
```

La clase PosLight

```
void PosLight::upload(glm::dmat4 const& modelViewMat) const {
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixd(value_ptr(modelViewMat));
    glLightfv(id, GL_POSITION, value_ptr(posDir));
    glLightf(id, GL_CONSTANT_ATTENUATION, kc);
    glLightf(id, GL_LINEAR_ATTENUATION, kl);
    glLightf(id, GL_QUADRATIC_ATTENUATION, kq);

    uploadL();
}

// Ojo al 1.0 que determina que la luz sea local
void PosLight::setPosDir(glm::fvec3 dir) {
    posDir = glm::fvec4(dir, 1.0);
}
```

La clase SpotLight

```
class SpotLight : public PosLight {  
protected:  
    // Atributos del foco  
    glm::fvec4 direction = { 0, 0, -1, 0 };  
    GLfloat cutoff = 180;  
    GLfloat exp = 0;  
public:  
    SpotLight(glm::fvec3 pos = { 0, 0, 0 }) : PosLight() {  
        posDir = glm::fvec4(pos, 1.0);  
    };  
    virtual void upload(glm::dmat4 const& modelViewMat) const;  
    void setSpot(glm::fvec3 dir, GLfloat cf, GLfloat e);  
};
```

La clase SpotLight

```
void SpotLight::upload(glm::dmat4 const& modelViewMat) const {
    PosLight::upload(modelViewMat);
    glLightfv(id, GL_SPOT_DIRECTION, value_ptr(direction));
    glLightf(id, GL_SPOT_CUTOFF, cutoff);
    glLightf(id, GL_SPOT_EXPONENT, exp);
}

// Ojo al 0.0: la dirección de emisión del foco es vector
void setSpot(glm::fvec3 dir, GLfloat cf, GLfloat e) {
    direction = glm::fvec4(dir, 0.0); cutoff = cf;
    exp = e;
}
```

La clase Material

```
class Material {  
protected:  
    // Coeficientes de reflexión  
    glm::fvec4 ambient = {0.2, 0.2, 0.2, 1.0};  
    glm::fvec4 diffuse = {0.8, 0.8, 0.8, 1.0};  
    glm::fvec4 specular = {0.0, 0.0, 0.0, 1.0};  
    GLfloat expF = 0;      // Exponente para la reflexión especular  
    GLuint face = GL_FRONT_AND_BACK;  
    GLuint sh = GL_SMOOTH; // Tipo de matizado  
public:  
    Material() {};  
    virtual ~Material() {};  
    virtual void upload();  
    void setCopper();  
};
```

La clase Material

```
void Material::upload() {  
    glMaterialfv(face, GL_AMBIENT, value_ptr(ambient));  
    glMaterialfv(face, GL_DIFFUSE, value_ptr(diffuse));  
    glMaterialfv(face, GL_SPECULAR, value_ptr(specular));  
    glMaterialf(face, GL_SHININESS, expF);  
    glShadeModel(sh);  
    //glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_FALSE); // Defecto  
}  
  
void Material::setCopper() {  
    ambient = {0.19125, 0.0735, 0.0225, 1};  
    diffuse = {0.7038, 0.27048, 0.0828, 1};  
    specular = {0.256777, 0.137622, 0.086014, 1};  
    expF = 12.8;  
}
```

La clase Material

```
class EntityWithMaterial : public Abs_Entity {  
public:  
    EntityWithMaterial () : Abs_Entity() { };  
    virtual ~EntityWithMaterial () { };  
  
    void setMaterial(Material* matl) {material = matl};  
  
protected:  
    Material* material = nullptr;  
};
```

Clases modificadas para manejar la iluminación

- La clase `Scene` contiene gran parte de las luces de la escena en forma de atributos
- Las luces de la escena se crean, y se especifican sus características, en `setLights()` de `Scene`
- En `render()` de `Scene` se hace `upload()` de todas ellas
- El resto de las luces aparecen en la escena porque suelen formar parte de entidades; por ejemplo, la luz de una lámpara de una habitación, los focos de un coche, o la luz de exploración de una nave sobre un planeta
- Estas luces se declaran como atributos de la entidad de la que forman parte y se construyen y establecen sus características en la constructora de esa entidad
- Si una luz está asociada a una entidad y se mueve de forma solidaria con la entidad (por ejemplo, los focos de los faros de un coche, el foco de exploración de una nave, etc.), debe fijarse su posición en el `render()` de la entidad (ver luces dinámicas, más adelante)

Tipos de luces en una escena

• Estáticas (o fijas)

- Son las luces, ya sean remotas, locales, y dentro de estas, focos o no, que no cambian su posición durante la renderización de la escena
- La forma de comprobar que una luz es estática consiste en mover la cámara y ver que los efectos de la luz no cambian
- La posición debe restablecerse cada vez que cambie la matriz de modelado-vista
- Si la cámara no se va a mover, su posición se fija solo una vez en `setLights()` de `Scene`
- Si la cámara se va a mover, hay que volver a fijar la posición de la luz cada vez que se mueva. Por ello, al principio de `render(cam)` de `Scene`, la luz tiene que llamar a `upload(cam.viewMat())` donde lo primero que se hace es cargar la matriz de modelado-vista y después se fija la posición de la luz

Tipos de luces en una escena

• Dinámicas

- Luces que cambian de posición, independientemente de la cámara
 - **Focos en objetos de la escena que se mueven** (por ejemplo, la luz de exploración debajo de una nave)
 - La posición inicial se fija en la constructora de la entidad que las contiene; generalmente esa posición suele ser el punto $(0, 0, 0)$
 - La posición se actualiza después de multiplicar por la matriz `modelMat`, en el `render()` de la entidad que las contiene
- Luces que se mueven con la cámara
 - **Luz de minero posicional** (que puede ser foco o no) en la cámara
 - La posición se establece una vez, después de que la matriz de modelado-vista se ha fijado
 - Si se trata de un foco colocado en la cámara, su dirección de emisión es $(0, 0, -1)$
 - La posición se fija en `setLights()` de `Scene` y se actualiza en `render()` de `Scene` llamando con ella a `upload(dmat4(1.0))`

Observaciones

- Los cálculos de la iluminación en OpenGL se realizan en coordenadas de cámara.
- Hay que establecer las fuentes de luz y encenderlas antes de renderizar los objetos que van a iluminar.
- Los focos deben iluminar vértices de una malla para que se vea el cono de emisión. Si no lo hacen su luz no aparece.
- Texturas:
 - Para no alterar los colores de la textura se suelen utilizar materiales grises.
 - Los colores de la textura se suelen mezclar con el de la iluminación con el comando:

```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, mix);
```

siendo `mix`: `GL_MODULATE` (o `GL_REPLACE`, `GL_ADD`, `GL_SUBTRACT`, `GL_DECAL`, ...)