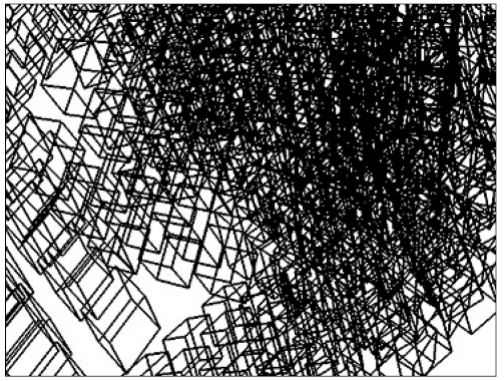
Iluminación

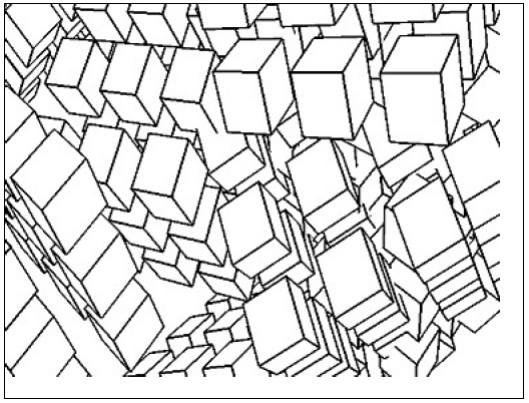
A. Gavilanes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Renderización

- Renderización de escenas
- Jerarquía en los niveles de realismo
  - Renderización basada en armazón de hilos

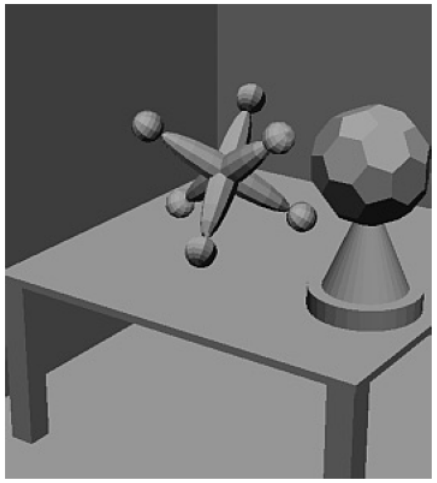


  - Renderización basada en armazón de hilos con eliminación de superficies ocultas

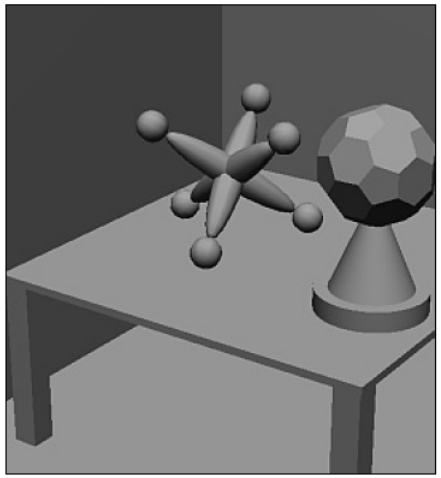


Renderización

- Iluminación plana (flat shading)

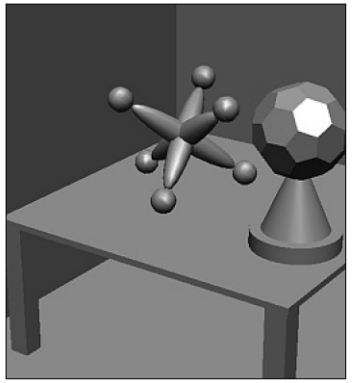


- Iluminación suave (smooth shading)

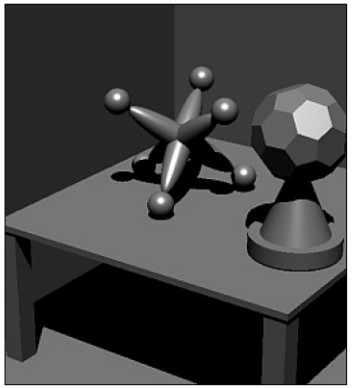


Renderización

- Iluminación con efectos especulares



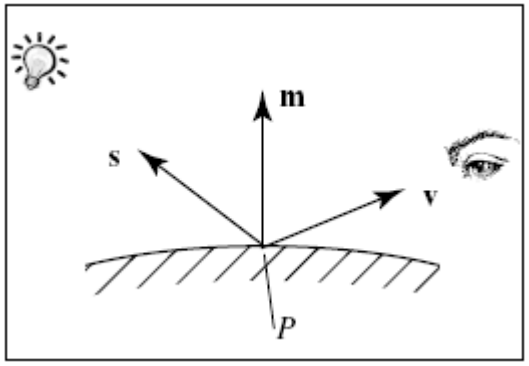
- Iluminación con sombras



- Iluminación con texturas

Lighting models

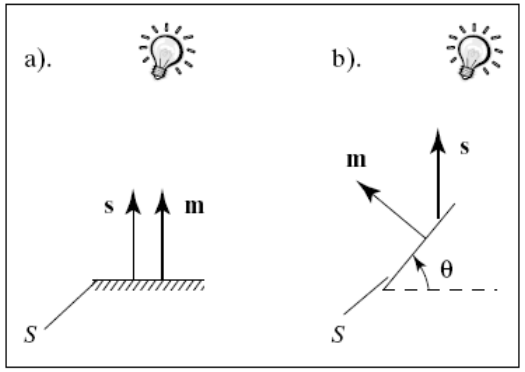
- Un modelo de iluminación dicta cómo se refleja y dispersa la luz, cuando incide sobre una superficie.
- Simplificación con respecto al modelo físico de propagación de la luz (absorción de luz, transmisión de la luz a través de los objetos, intensidad de la luz emitida).
- Fenómenos principales que 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 (**m**) a la superficie en el punto P
  - Vector desde (**v**) P al ojo de la cámara
  - Vector desde (**s**) P a la fuente de luz



Componentes difusa, ...

- Dispersión difusa
  - Luz que, habiendo incidido sobre una superficie, se vuelve a irradiar desde ella, en todas las direcciones.
  - Su intensidad depende del ángulo que forman los vectores **s** y **m**.
  - Para su implementación se usa el modelo de Lambert:
    - `dvec3 diffuse = max(0, cos θ)*diffLight*diffMaterial`

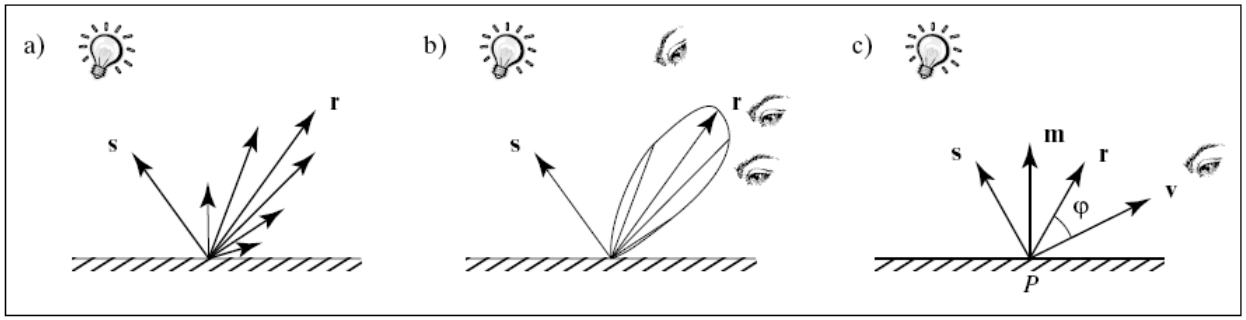
    donde `cos θ = dot(m, s)` (si los vectores están normalizados)
  - Para fuentes de luz lejanas, el vector **s** varía poco, luego la componente difusa cambia poco a lo largo de la superficie



… especular

- Reflexión especular
  - Luz que refleja la superficie.
  - El vector de máxima reflexión **`r(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} **`ϕ)*specLight*specMaterial`**

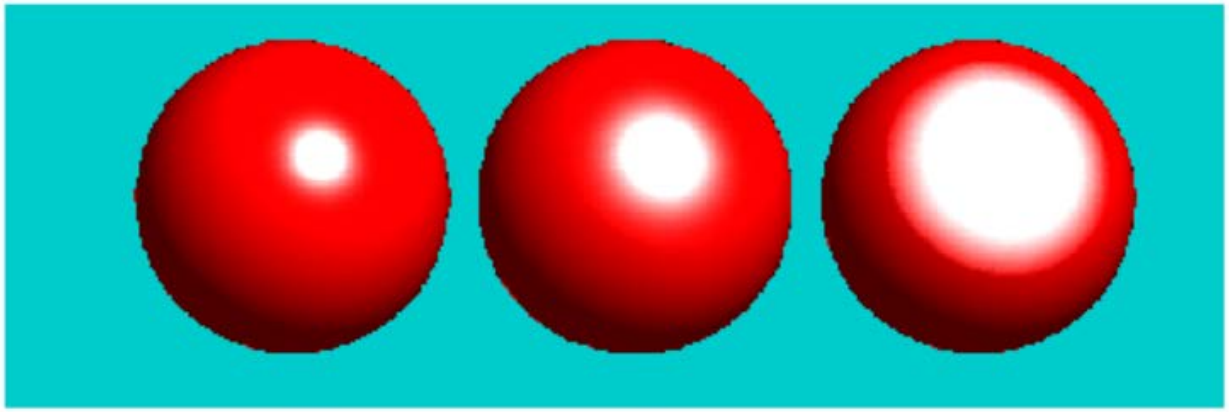
    donde **`cos ϕ = dot(r, v)`** (si los vectores están normalizados) y **`f=1,…,∞`** (espejo) (es un atributo del material)
  - Para fuentes de luz cercanas, los vectores **`r`** y **`s`** varían mucho luego la componente especular cambia mucho a lo largo de la superficie.



… especular

- Modelo de Phong:
  - **dvec3 specular = max(0, cos $^{f}\phi$)*specLight*specMaterial**

  donde **cos ϕ = dot(r, v)** (si los vectores están normalizados) y **f=1,…,∞** (espejo) (es un atributo del material)
- Ejemplo para algunos valores de **f**



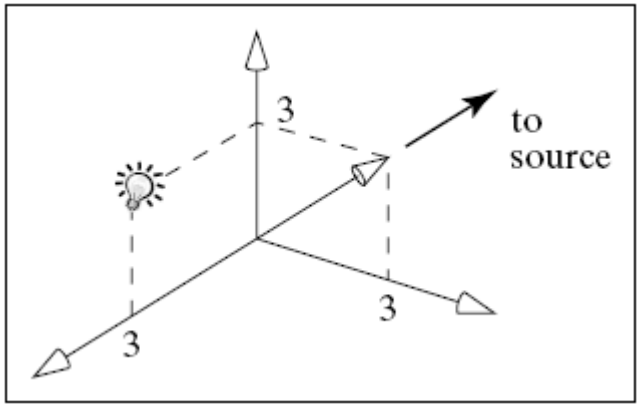
… y ambiente de las fuentes de luz

- 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`

- El color resultante de **una** fuente de luz es la combinación de las tres componentes obtenidas:
  - `dvec3 finalColor = diffuse + specular + ambient`
- 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

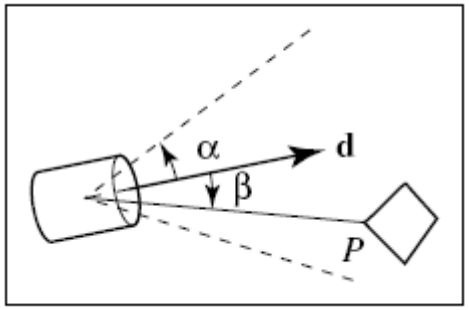
- Las fuentes de luz en OpenGL son de dos tipos:
  - Direccionales (o remotas): se supone que están infinitamente lejos y que, por tanto, sus rayos de luz llegan paralelos a la escena. Por su lejanía infinita se supone que no se atenúan. Se especifican mediante el vector que va **del origen a la fuente 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:
  - 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** β. Cuanto mayor es ε, más se concentra la luz del foco,



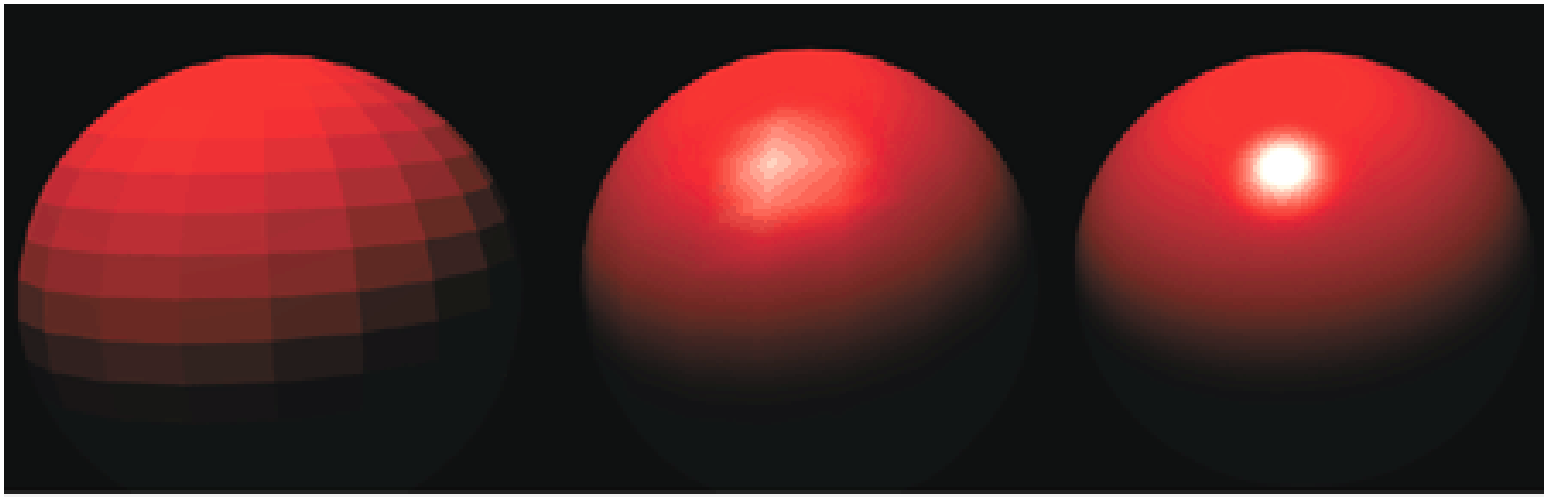
  - Omnidireccionales: El resto de las fuentes de luz locales, que se caracterizan por emitir luz por igual, en todas direcciones.

Shading models en OpenGL

- El modelo de matizado o tonalidad (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)
```

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.



Flat Gouraud Phong

Las fuentes de luz en OpenGL

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

```
GLfloat v[]={3.0, 2.0, 1.0, 1.0};

glLightfv(GL_LIGHT0, GL_POSITION, 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.

Las fuentes de luz en OpenGL

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

```
GLfloat amb0[]={0.2, 0.4, 0.6, 1.0};
GLfloat dif0[]=…;
GLfloat esp0[]=…;
glLightfv(GL_LIGHT0, GL_AMBIENT, amb0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, dif0);
glLightfv(GL_LIGHT0, GL_SPECULAR, 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

Focos en OpenGL

- Los comandos de OpenGL para especificar un foco son (los valores que aparecen son ejemplos):

```
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 4.0);
GLfloat dir[]={2.0, 1.0, -4.0};
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
```

  que especifican un foco con α=45º, ε=4 y **d**=(2, 1, -4).

- Por defecto, α=180º, ε=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.
- El factor de atenuación de una fuente real de luz se modela mediante la fórmula:

$$factor\quad atenuacion = \frac{1}{k_c + k_l d + k_q d^2}$$

$$k_c = GL\_CONSTANT\_ATTENUATION$$
$$k_l = GL\_LINEAR\_ATTENUATION$$
$$k_d = GL\_QUADRATIC\_ATTENUATION$$

  donde *d* es la distancia.

- 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, anbiente) de la fuente de luz. Por ejemplo, en cuanto al color, una superficie con componentes difusa y ambiente verde bajo una luz con componentes difusa y ambiente roja resulta ser ...
- En consecuencia, en OpenGL, los materiales se especifican con las mismas tres componentes (ambiente, difusa y especular) que las fuentes de luz.
- 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 la 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, en los materiales 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.

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

Material	ambient: $\rho_{ar}, \rho_{ag}, \rho_{ab}$	diffuse: $\rho_{dr}, \rho_{dg}, \rho_{db}$	specular: $\rho_{sr}, \rho_{sg}, \rho_{sb}$	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

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

```
glMaterialfv(face, componenteDeLaLuz, RGBA);
```

donde:

face 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

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

```
GLfloat a[]=(…, …, …, 1.0);
```

- **`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.

Materiales usando el modo ColorMaterial

- Con el uso de **`glMaterial()`**, ¿ya no se usa **`glColor()`**?
- El color se puede establecer también mediante lo que se llama el *registro de color*.
- El código para definir los materiales usando el registro de color es el siguiente:

```
glEnable(GL_COLOR_MATERIAL);  // Se activa el registro de color
glColorMaterial(face, componenteDeLaLuz);
glColor3f(…, …, …);
glBegin(GL_TRIANGLES);  // O lo que sea
   glVertex3f(…, …, …);
glEnd();
```

donde **`face`** se define como antes, y **`componenteDeLaLuz`** también, pero sin admitir **`GL_SHININESS`**.

Comandos para tratar las 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)`

La clase Light

```
class Light { // Abstract class
   protected:
      static GLuint cont = 0; // Para generar un nuevo id cada vez
      GLuint id = GL_LIGHT0 + GL_MAX_LIGHTS; // 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 setter's para cambiar el valor de los atributos lumínicos
   public:
      Light();
      virtual ~Light() {disable();}
      void uploadL();
      virtual void upload(glm::dmat4 const& modelViewMat) = 0;
… };
```

La clase Light

- Funcionalidad de la clase

```
Light() {
    if (cont < GL_MAX_LIGHTS) {
            id = GL_LIGHT0 + cont;
            ++cont;
            glEnable(id);}};
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);
      void setPosDir(glm::fvec3 dir);
};
```

donde:

```
void DirLight::upload(glm::dmat4 const& modelViewMat) {
        glMatrixMode(GL_MODELVIEW);
        glLoadMatrixd(value_ptr(modelViewMat));
        glLightfv(id, GL_POSITION, value_ptr(posDir));
        uploadL();
}

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);
    void setPosDir(glm::fvec3 dir);
    void setAtte(GLfloat kc, GLfloat kl, GLfloat kc);
};
```

- donde:

```
void PosLight::upload(glm::dmat4 const& modelViewMat) {
    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();
}

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);
        void setSpot(glm::fvec3 dir, GLfloat cf, GLfloat e);
};
```

- donde:

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

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 setCooper();
};
```

- donde:

```
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 EntityMaterial (Opcional)

```
class EntityMaterial : public Entity {
    public:
        EntityMaterial() : Entity() { };
        virtual ~EntityMaterial() { };

        void setTexture(Texture* tex) {texture = tex;};
        void setMaterial(Material matl) {material = matl;};
    protected:
        Texture* texture = nullptr;
        Material material;
};
```

Clases modificadas para manejar la iluminación

- La clase **`Scene`** contiene gran parte de las luces de la escena en forma de atributos protegidos
- Las luces de la escena se crean, y se especifican sus características, en **`init()`** 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 un dron
- 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 un dron, etc.), debe fijarse su posición en el **`render()`** de la entidad (ver luces dinámicas, más adelante)

El atributo `color`

- Puede facilitar el trabajo con las entidades, las luces y el modo **`ColorMaterial`**, añadir un atributo **`glm::fvec3 color = fvec3(-1.0, -1.0, -1.0)`** a la clase **`Entity`**. Para las entidades diferentes a **`CompoundEntity`**, este atributo vale “el color de la entidad”. En el resto toma el valor por defecto de arriba.
- Entonces, para contemplar el caso en que el material se proporciona con **`glColor()`** y se usa el modo **`ColorMaterial`**, puedes modificar el método **`render()`** en aquellas clases que uses de forma que se active el color, cuando este existe. Por ejemplo, la clase **`Esfera`**, que construye la malla de la esfera por revolución, tiene un método **`render()`** que podría reescribirse de esta forma:

```
void Esfera::render(dmat4 const& modelViewMat) {
     if (mesh != nullptr) {
        uploadMvM(modelViewMat);
        if (color!=fvec3(-1.0,-1.0,-1.0)) glColor3f(value_ptr(color));
        else glColor3f(0.2, 0.4, 0.6);
        mesh->render();}
}
```

Observaciones

- Los cálculos de la iluminación en OpenGL se realizan en coordenadas de cámara. Como se verá después, según las luces sean estáticas o dinámicas, se debe tener en cuenta por tanto si la matriz de modelado-vista ha cambiado o no.
- Hay que establecer las fuentes de luz y encenderlas antes de renderizar los objetos que van a iluminar.
- Un objeto se puede dejar oscuro si se renderiza antes de que actúen las fuentes de luz que lo pueden 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.
- 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);
```

con **mix**: **GL_MODULATE** (o **GL_REPLACE**, **GL_ADD**, **GL_SUBTRACT**, **GL_DECAL**, …)

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.

- Ojo de la cámara para la reflexión especular

```
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

- Coloreado de ambos lados de las caras de una malla

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

  El valor por defecto es **GL_FALSE**. Es útil cuando se quiere que se muestren las caras traseras. OpenGL utiliza entonces como vector normal de una cara, el opuesto del de la cara frontal.

- 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);**

Tipos de luces en una escena

- **Estáticas (o fijas)**
  - No cambian su posición durante la renderización de la escena
  - 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 **`init()`** 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(modelViewMat)`** de **`Scene`**, la luz tiene que llamar a **`upload(modelViewMat)`** donde lo primero que se hace es cargar la matriz de modelado-vista y después se fija la posición de la luz

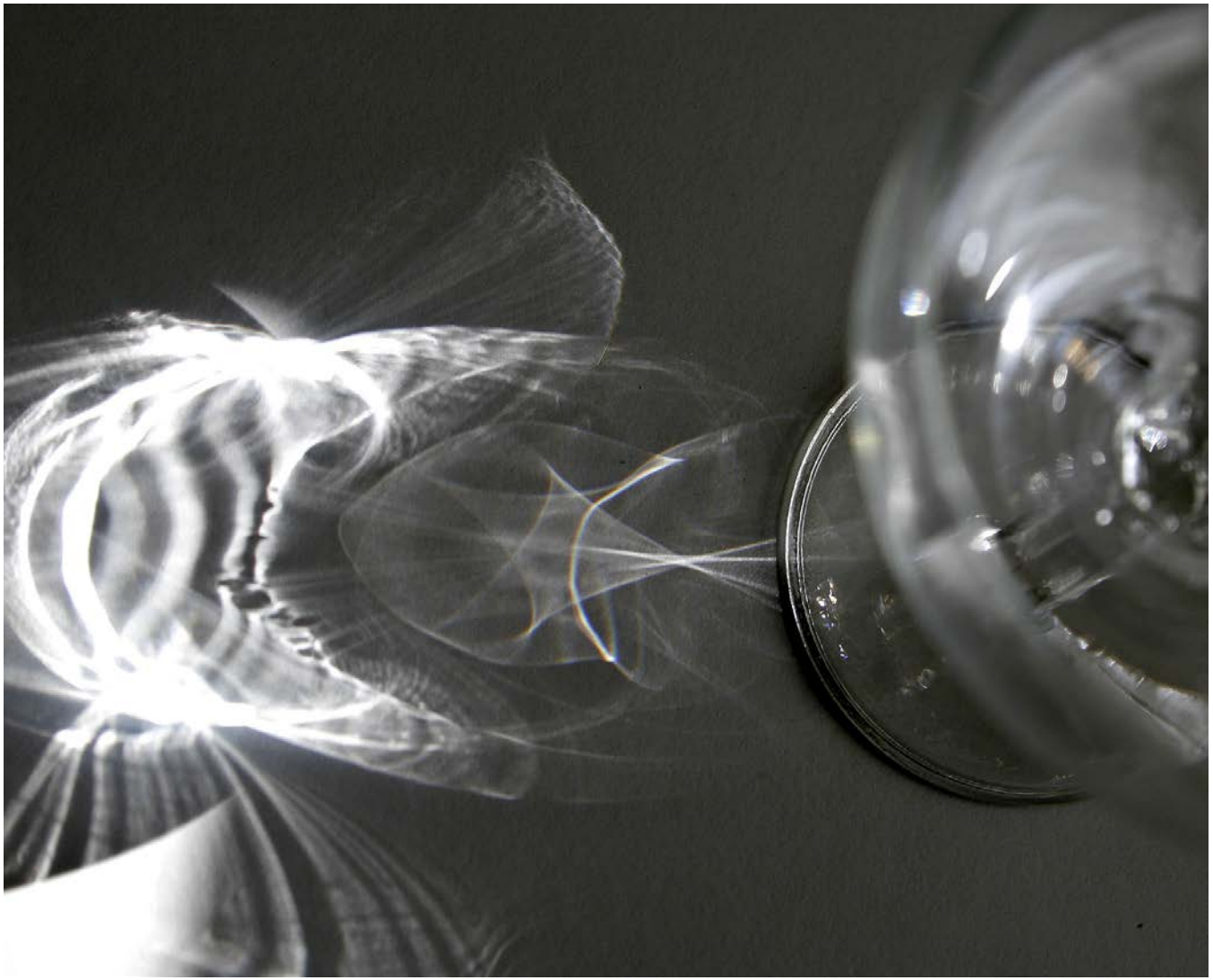
Tipos de luces

- **Dinámicas**
  - Luces que cambian de posición, independientemente de la cámara.
    - **Focos**.
    - La posición se fija después de multiplicar por la matriz **`modelMat`** que coloca el objeto que las contiene.
    - La posición se fija en el método **`render()`** del objeto que las contiene.
  - Luces que se mueven con la cámara.
    - **Luz de minero**.
    - 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 **`init()`** de **`Scene`** y se actualiza en **`render(modelViewMat)`** de **`Scene`** llamando con ella a **`upload(dmat4(1.0))`**

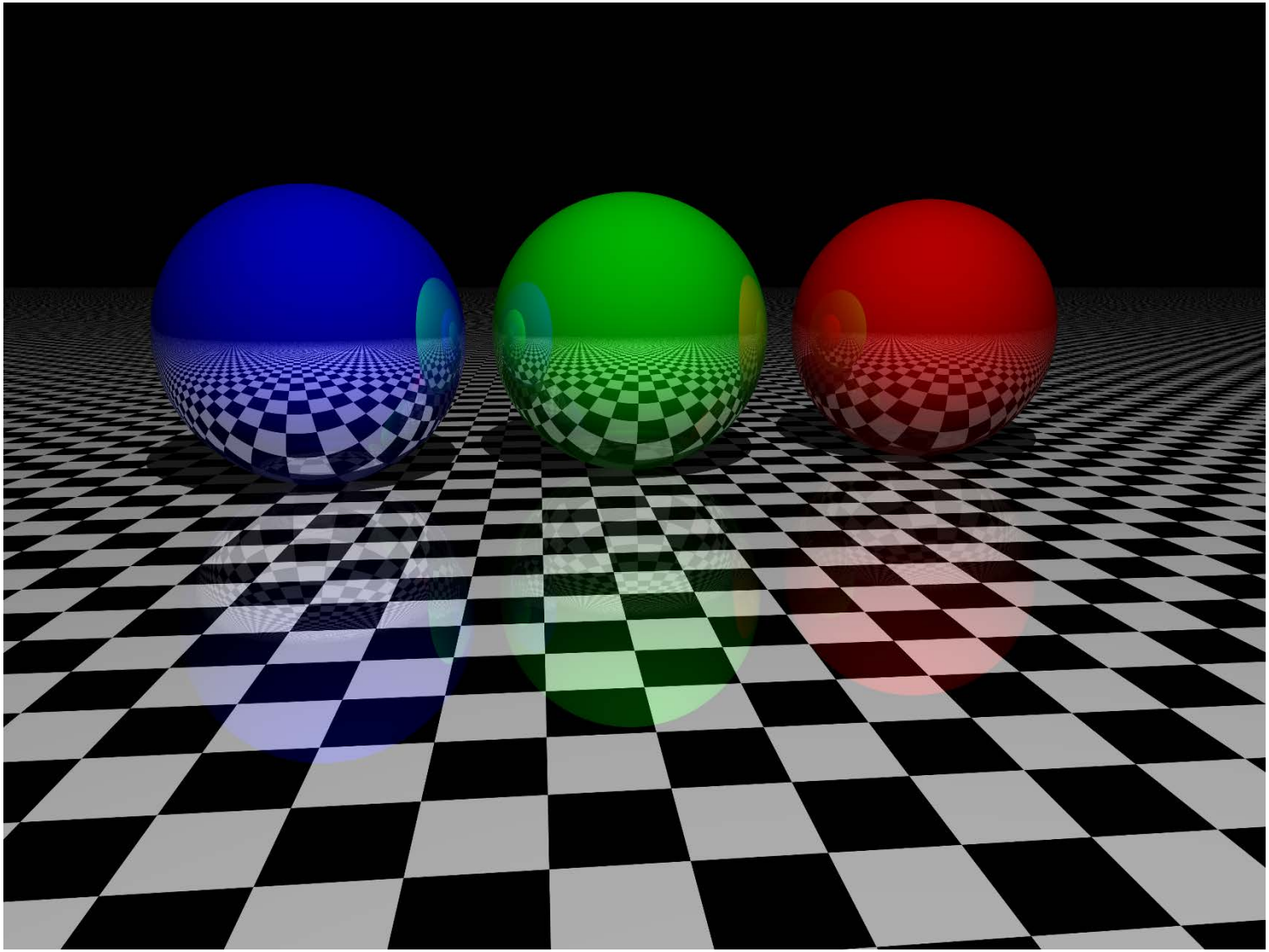
Modelos de iluminación locales vs globales

- 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, cáusticas) 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

Cáusticas



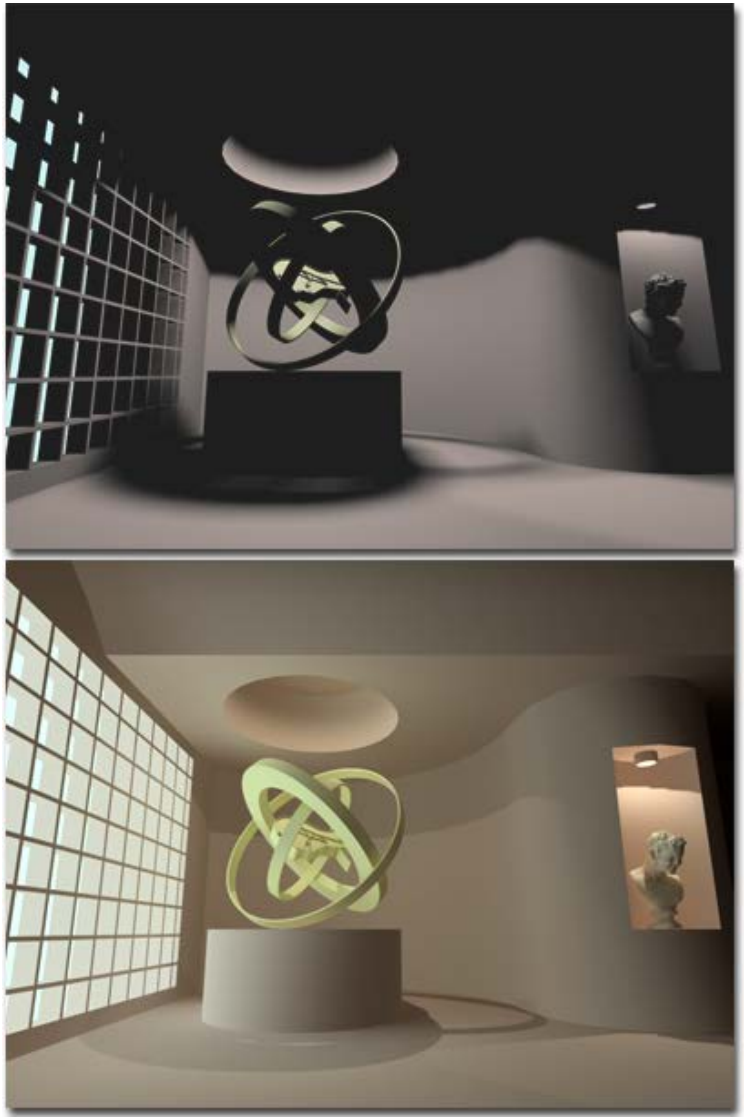
Reflejos



Sombras, reflejos y cáusticas

Modelo por radiosidad

Modelo por radiosidad



Modelo por ray tracing

Modelo por ray tracing