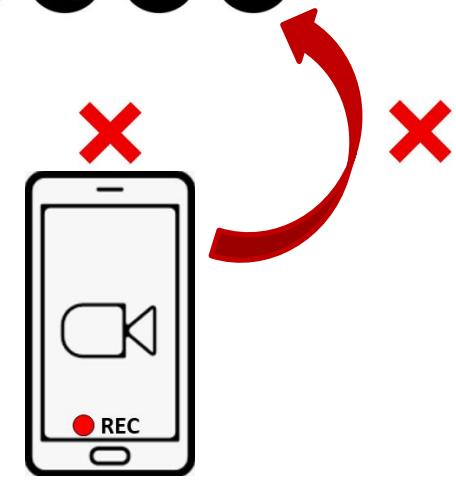
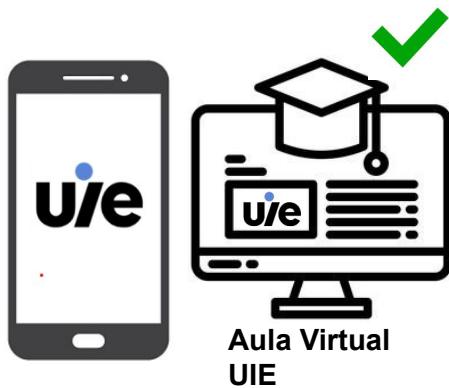
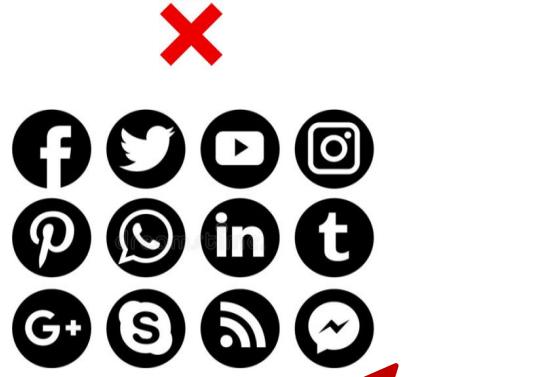
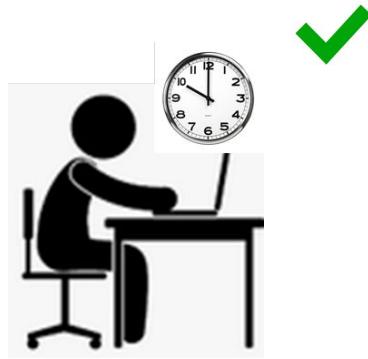


Asignatura

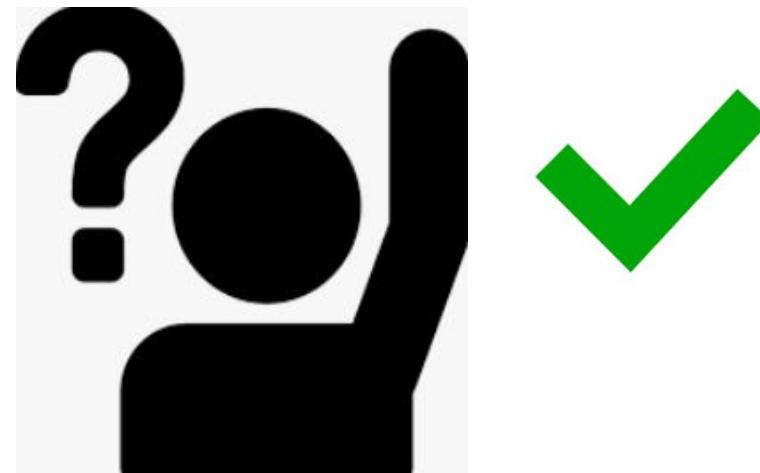
Computación gráfica

Profesor

David Cereijo Graña







Participar

Sesión 09

Unidad III

Gráficos 3D

Tema 3.4

Renderizado y sombreado de escenas 3D

Sombreado

Interacciones básicas entre la luz y una superficie

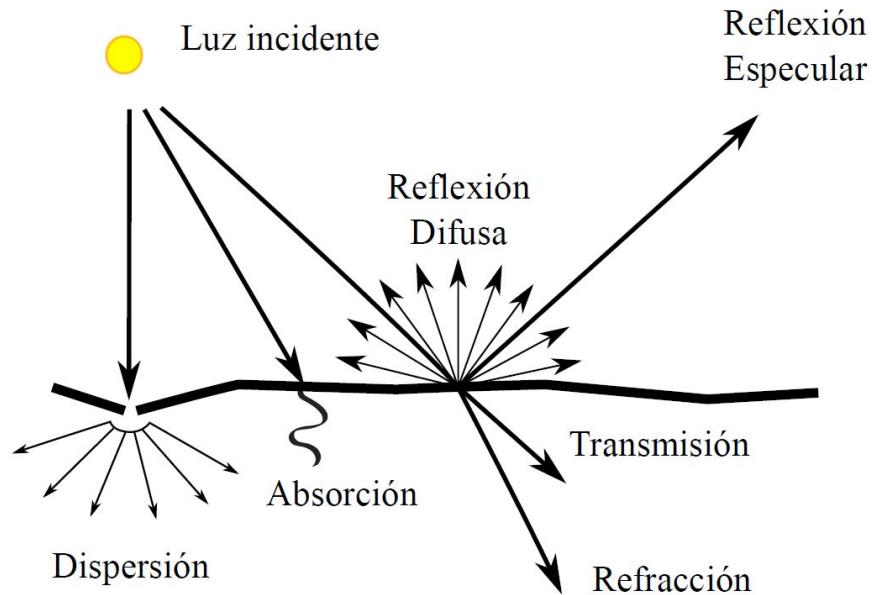
La luz está formada por partículas llamadas fotones, que se desplazan en línea recta en forma de rayos de luz. Cuando un rayo de luz incide sobre una superficie, puede experimentar diferentes fenómenos:

- **Absorción:** parte de la luz puede ser absorbida por la superficie. En este caso, la energía de los fotones se transfiere al material, convirtiéndose generalmente en calor.
- **Reflexión:** otra parte de la luz puede ser reflejada, es decir, los fotones rebotan sobre la superficie y cambian su dirección. Dependiendo de la naturaleza de la superficie, la reflexión puede ser **especular** (como en un espejo, donde la luz se refleja en una dirección definida) o **difusa** (como en superficies rugosas, donde los fotones se reflejan en múltiples direcciones).
- **Transmisión:** Una porción de la luz también puede ser transmitida a través del material, cruzando la superficie y atravesando el objeto.
- **Refracción:** en algunos casos, la luz puede cambiar de dirección al entrar en otro medio debido a la refracción.
- **Dispersión:** al interactuar con partículas o irregularidades, la luz también puede dispersarse en diferentes direcciones.

Interacciones básicas entre la luz y una superficie

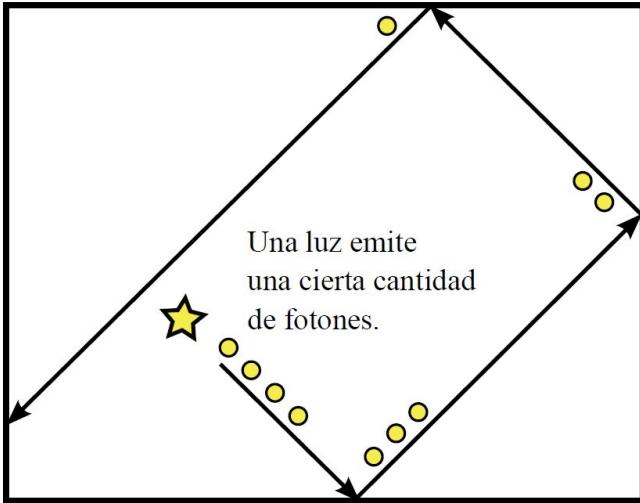
Estos procesos no son excluyentes y pueden ocurrir simultáneamente en diferentes proporciones. Es decir una misma superficie puede reflejar una parte de la luz, transmitir otra parte, y absorber otra fracción.

La combinación y proporción de estos efectos depende de factores como el ángulo de incidencia de la luz, el índice de refracción del material, y las propiedades ópticas específicas de la superficie.



Interacciones básicas entre la luz y una superficie

Eventualmente
toda la energía es
absorbida.



Un porcentaje
de fotones es absorbido
y otro es reflejado.

Este proceso de interacción entre la luz y las superficies se repite hasta que toda la energía de los fotones ha sido completamente transferida.

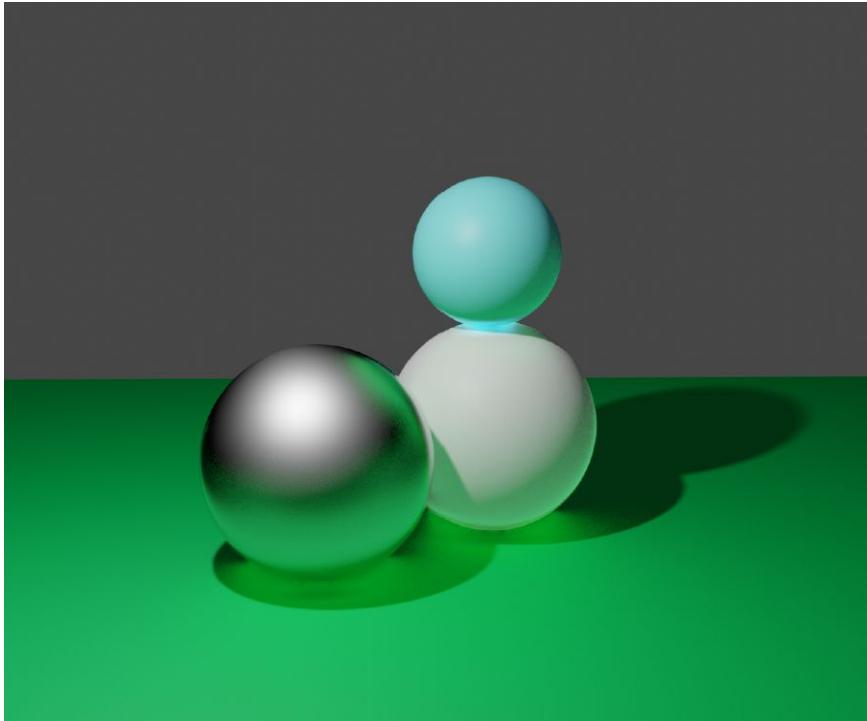
Modelos de iluminación global vs modelos de iluminación local

En el mundo real las superficies reciben luz de otras superficies, incluso si estas no producen luz o no son iluminadas directamente, por lo que podemos clasificar a las fuentes de luz en dos tipos:

- **Fuente de luz primaria o emisor:**
produce la luz que emite.
- **Fuente de luz secundaria o reflector:**
refleja la luz que producen otras fuentes.



Modelos de iluminación global vs modelos de iluminación local



La dispersión recursiva de la luz entre las distintas superficies da lugar a efectos sutiles como:

- **Sombras:** producidas entre objetos adyacentes.
- **Luz indirecta:** luz recibida a través de la reflexión de la luz en objetos cercanos.
- **Color bleeding:** el color de un objeto es influenciado por el color de otros objetos que le rodean.

La suma de todos estos efectos se conoce como **iluminación global**.

Modelos de iluminación global vs modelos de iluminación local

Dado que la luz viaja a gran velocidad ($\approx 3 \cdot 10^8$ m/s), independientemente del tamaño de la escena, la luz tardará un tiempo insignificante en propagarse por todo el entorno.

Sin embargo, reproducir la iluminación global en una escena virtual es una tarea compleja y computacionalmente muy costosa.

Por tanto, para facilitar los cálculos, nos enfocaremos en modelos de **iluminación local**, donde evaluaremos únicamente las contribuciones de las fuentes de luz primarias sobre un punto de la superficie, considerando siempre que tanto la fuente de luz como el punto son visibles.

De este modo, si un objeto debería generar una sombra sobre otro, ésta no se vería, obteniendo un resultado menos realista.

Modelos de iluminación global vs modelos de iluminación local

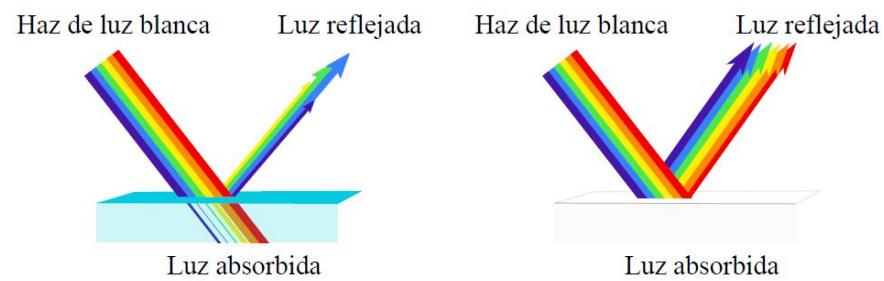
Por tanto, a la hora de simular cómo interactúa la luz con las superficies en una escena podemos utilizar dos grandes modelos:

- **Modelos de iluminación local:** solo consideran la luz que llega directamente desde una fuente hacia un objeto, sin tener en cuenta cómo interactúa la luz con otros objetos de la escena. Estos modelos, como el modelo de iluminación **Phong**, son rápidos y eficientes, pero no pueden simular efectos complejos como las sombras suaves, la reflexión entre objetos, o la luz difusa que se refleja de otras superficies.
- **Modelos de iluminación global:** consideran todas las interacciones posibles de la luz en la escena, incluyendo reflexiones, refracciones y la luz indirecta que se refleja entre los objetos. Estos modelos, como el **trazado de rayos** o **ray tracing** son más precisos y realistas, pero requieren más potencia de cálculo y tiempo de procesamiento.

Modelos de iluminación y color

Por otra parte, cuando un objeto es iluminado por una fuente de luz, el color que percibimos que tiene es el resultado de la luz que está reflejando.

Por ejemplo, si un haz de luz blanca choca contra un objeto y éste se ve de un color azul turquesa, quiere decir que ha absorbido todos los colores, menos los azules y verdes, que son los que componen el color que refleja.



Modelos de iluminación y color

Esta interacción del color en las superficies puede ser modelada mediante el modelo **RGB**.

Consideremos que tenemos una **fuente de luz blanca** que ilumina un objeto que, de color verde bosque, entonces el color que veríamos estaría dado por:

COLOR DE LA LUZ		COLOR DEL OBJETO		COLOR REFLEJADO
(1.0, 1.0, 1.0)	×	(0.121, 0.498, 0.121)	=	(0.121, 0.498, 0.121)

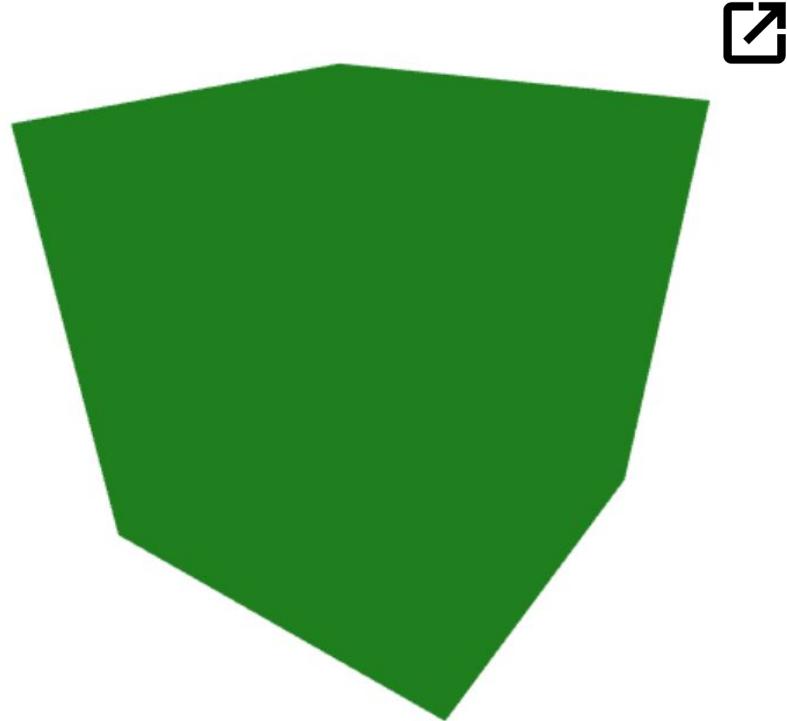
Y si utilizásemos una **fuente de luz monocromática de color rojo**:

COLOR DE LA LUZ		COLOR DEL OBJETO		COLOR REFLEJADO
(1.0, 0.0, 0.0)	×	(0.121, 0.498, 0.121)	=	(0.121, 0.0, 0.0)

Modelos de iluminación y color

La aplicación de este modelo, sin más, tiene una importante limitación, pues si repetimos el cálculo para todos los puntos del objeto tendremos como resultado una representación de color uniforme y plano, como se puede apreciar en la figura.

Necesitamos un modelo de iluminación local que nos permita obtener una aproximación aceptable de cómo debería lucir un objeto en el mundo real.



Modelo de iluminación de Phong

El **modelo de iluminación de Phong**, fue desarrollado en **1973** por informático vietnamita Bui Tuong Phong, y es uno de los modelos de iluminación local más populares en computación gráfica.

El modelo de Phong se basa en tres componentes principales:

- **Luz ambiental (*ambient light*)**
- **Luz difusa (*diffuse light*)**
- **Luz especular (*specular light*)**

Luz ambiental

- La luz ambiental representa la luz que está **presente en toda la escena** y no proviene de ninguna fuente en particular.
- Simula la **luz que rebota en los distintos objetos del entorno** antes de alcanzar al objeto en cuestión.
- **Proporciona un brillo base a todos los objetos**, asegurándose de que incluso en áreas donde no llega luz directa, los objetos no se vean completamente negros.

Matemáticamente:

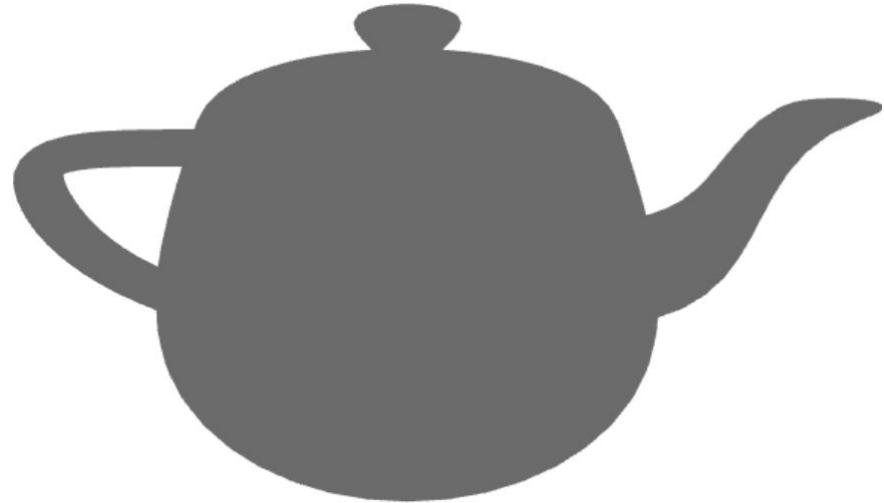
Sea I_a la intensidad de la **luz ambiental en la escena**, y k_a el coeficiente de reflexión ambiental característico del material, con $0 \leq k_a \leq 1$, entonces, la intensidad de la luz ambiental en un punto P de la superficie, puede ser modelada como:

$$I_{\text{ambiental}} = k_a \cdot I_a = k_a \cdot (I_{\text{a}}^R, I_{\text{a}}^G, I_{\text{a}}^B)$$

Luz ambiental

La luz ambiental agrega una cierta cantidad de luz que parece provenir de todas las direcciones con la misma intensidad, iluminando cada parte del objeto de una manera uniforme.

Debe utilizarse con cuidado, pues en exceso puede llegar a desvanecer algunos detalles.



Luz difusa: la Ley del Coseno de Lambert

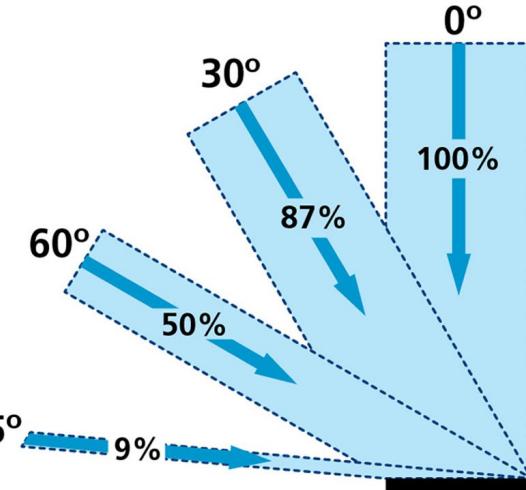
La **Ley del Coseno de Lambert** establece que la intensidad de la luz reflejada por una superficie difusa, I_{difusa} , es proporcional al coseno del ángulo θ entre el vector normal de la superficie y el vector de luz incidente.

- Si la luz incide de manera **perpendicular** a la superficie (es decir, el ángulo de incidencia es **0** grados), dado que el coseno de **0** es **1**, la luz reflejada es máxima.
- Si la luz incide de manera **tangencial** a la superficie (es decir, el ángulo de incidencia es **90** grados), dado que el coseno de **90** es **0**, no se reflejará luz.

Matemáticamente:

$$I_{\text{difusa}} = I_{\text{luz}} \cdot k_d \cdot \cos \theta$$

Donde k_d es el coeficiente de reflexión difusa de la superficie.



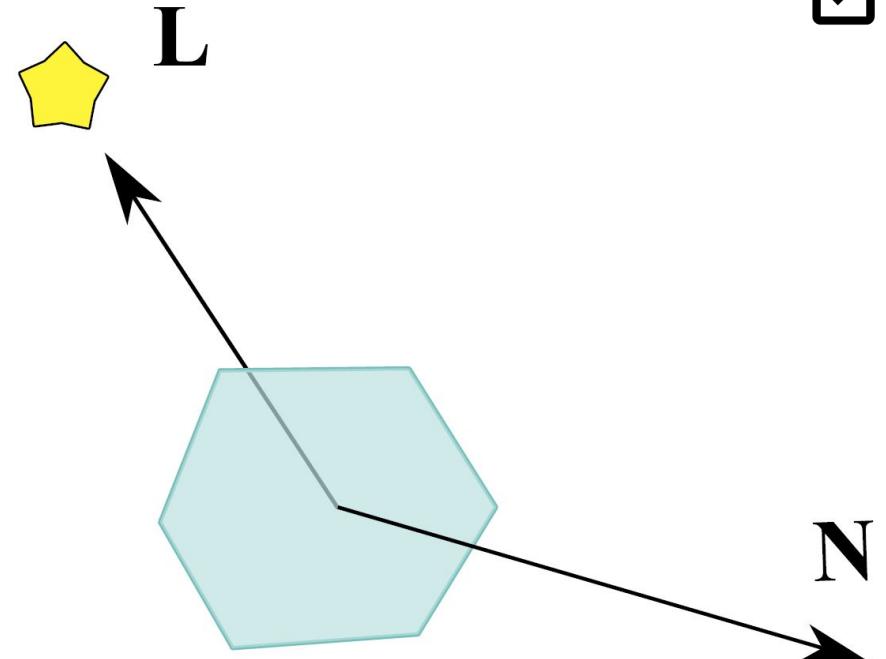
Luz difusa: la “luz negativa”

Sea I_{luz} la intensidad de la fuente de luz, \mathbf{N} el vector unitario normal a la superficie en el punto P , \mathbf{L} el vector unitario de luz incidente que va de la fuente de luz al punto P , y θ el ángulo que forman \mathbf{L} y \mathbf{N} .

El valor $\cos \theta$ puede obtenerse a partir del producto escalar de los vectores normalizados \mathbf{N} y \mathbf{L} . Pero, dado que el **producto escalar puede ser negativo** cuando la superficie apunta en dirección contraria a la luz, el valor del producto escalar se suele limitar al intervalo $[0, 1]$, ya que de otro modo tendríamos “luz negativa”.

Por tanto, la expresión de la Ley de Lambert corregida es:

$$I_{difusa} = I_{luz} \cdot k_d \cdot \max(0, \cos \theta) = I_{luz} \cdot k_d \cdot \max(0, \mathbf{N} \cdot \mathbf{L})$$



Luz difusa

- La luz difusa **representa la luz que incide directamente sobre una superficie y se dispersa en todas direcciones.**
- Sigue la **Ley del Coseno de Lambert**, lo que significa que la cantidad de luz reflejada depende del ángulo entre la fuente de luz y la normal de la superficie.
- **Proporciona sensación de volumen**, haciendo que las áreas más perpendiculares a la luz se vean más brillantes que las áreas inclinadas.

Matemáticamente:

Dado un punto \mathbf{P} de la superficie, sea I_{luz} la intensidad de la fuente de luz, k_d el coeficiente de reflexión difusa de la superficie, \mathbf{N} el vector unitario normal a la superficie en el punto \mathbf{P} , \mathbf{L} el vector unitario de luz incidente que va de la fuente de luz al punto \mathbf{P} , y θ el ángulo que forman \mathbf{L} y \mathbf{N} .

La intensidad de la luz difusa, I_{difusa} , en el punto \mathbf{P} puede ser modelada como:

$$\begin{aligned}I_{difusa} &= k_d \cdot I_{luz} = k_d \cdot I_{luz} \cdot \max(0, \cos \theta) = \\&= k_d \cdot I_{luz} \cdot \max(0, \mathbf{N} \cdot \mathbf{L})\end{aligned}$$

Luz difusa

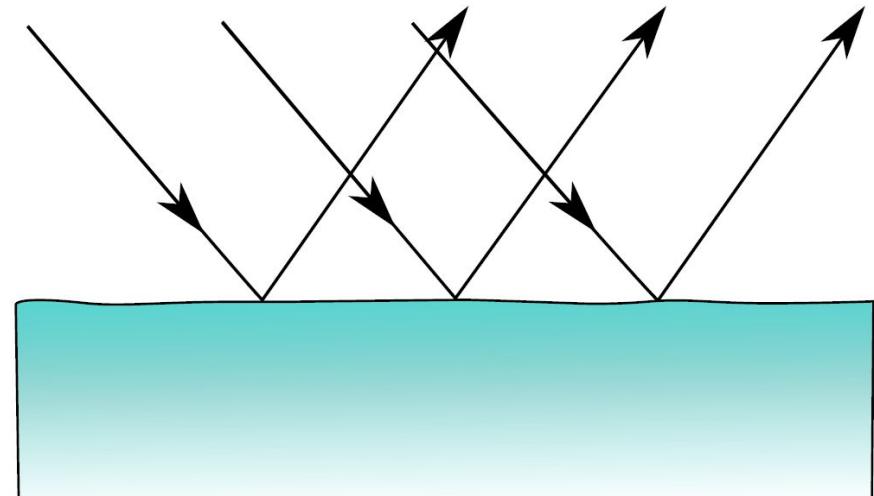


Luz especular

Cuando, a nivel microscópico, una superficie es muy lisa, los rayos de luz rebotan en la misma como si se tratara de un espejo.

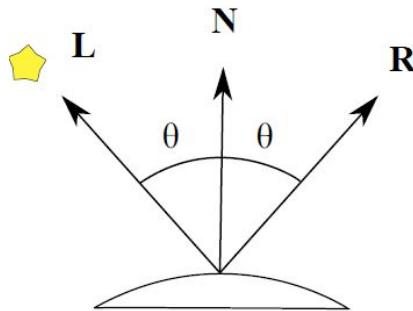
Cuanta menos irregularidades tenga la superficie, más se asemejará a un espejo, acercándose a lo que se conoce como **superficie especular ideal o perfecta**.

En la figura se muestra una superficie especular ideal.

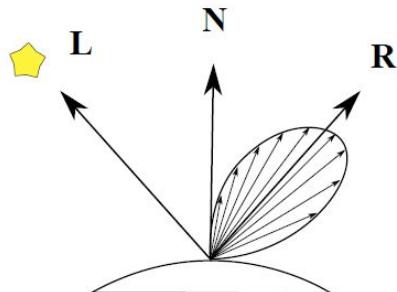


Luz especular

Superficie Especular Ideal



Superficie Especular No Ideal



Una superficie especular ideal refleja toda la luz incidente exactamente con el mismo ángulo, según un vector de reflexión **R**.

Si, por el contrario, la superficie no es ideal, una parte de la luz se reflejará a lo largo de este vector de reflexión **R**, mientras que otra cantidad lo hará de manera difusa alrededor del mismo.

Luz especular

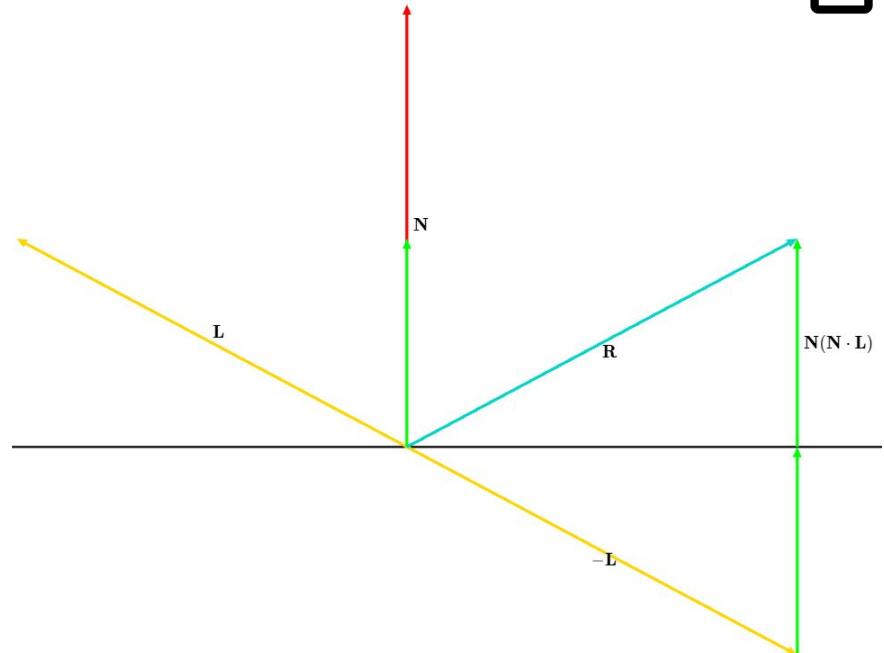
Sea \mathbf{N} el vector normal y \mathbf{L} la dirección de la luz incidente, ambos normalizados.

Podemos calcular expresar el vector de reflexión \mathbf{R} como:

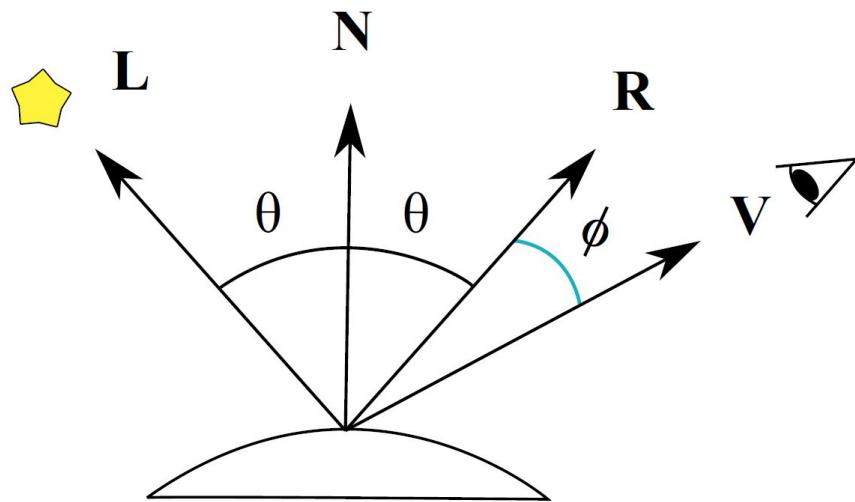
$$\mathbf{R} = 2\mathbf{N}(\mathbf{N} \cdot \mathbf{L}) - \mathbf{L}$$

Si asumimos que el observador está viendo hacia la superficie en la misma dirección que el vector espejo, \mathbf{R} , este podrá reflejar un brillo más tenue o más fuerte en función del material de la superficie.

En una superficie ideal, todos los rayos serán reflejados hacia el observador, mientras que en una no ideal, parte de los rayos serán dispersados en direcciones aleatorias.



Luz especular



En caso de que el observador estuviera en otra posición, es decir, el vector de vista **V** que va del punto en la superficie a la cámara, es distinto al vector **R**, la cantidad de luz que se refleja decrementa a medida que se incrementa el ángulo ϕ entre **R** y **V**.

Esto es, a medida que ϕ se incrementa, el número de rayos que son reflejados en la misma dirección que **V** disminuye.

Luz especular

- La luz especular simula los **reflejos brillantes que se presentan en superficies pulidas o satinadas**, como metales y vidrios.
- A diferencia de la luz difusa y ambiental, la luz especular **depende tanto de la dirección de la luz incidente como de la posición del observador**, siendo más intensa cuando el observador está en la misma dirección que el reflejo de la luz en la superficie.
- Este componente **aporta el brillo que hace que los objetos parezcan pulidos o metálicos**.

Matemáticamente:

Sea I_{luz} la intensidad de la luz fuente de luz, k_s el coeficiente de reflectancia especular del material, R el vector de reflexión de la luz respecto a la superficie, V el vector de dirección del observador, ambos normalizados, y α el exponente de brillo o constante especular.

Entonces, la intensidad de la luz especular en un punto P de la superficie, puede ser modelada como:

$$I_{especular} = k_s \cdot I_{luz} \cdot (R \cdot V)^\alpha$$

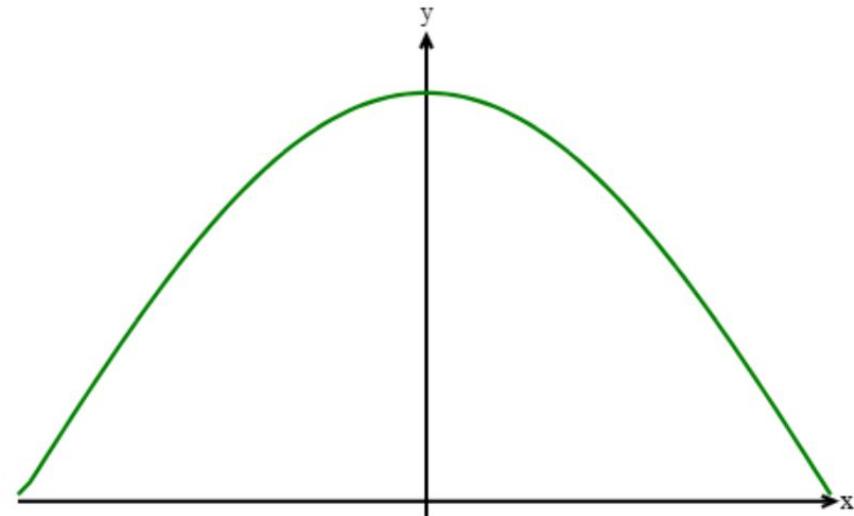
Con $0 \leq k_s \leq 1$ y α un número entero positivo que controla el tamaño y definición del brillo.

Luz especular

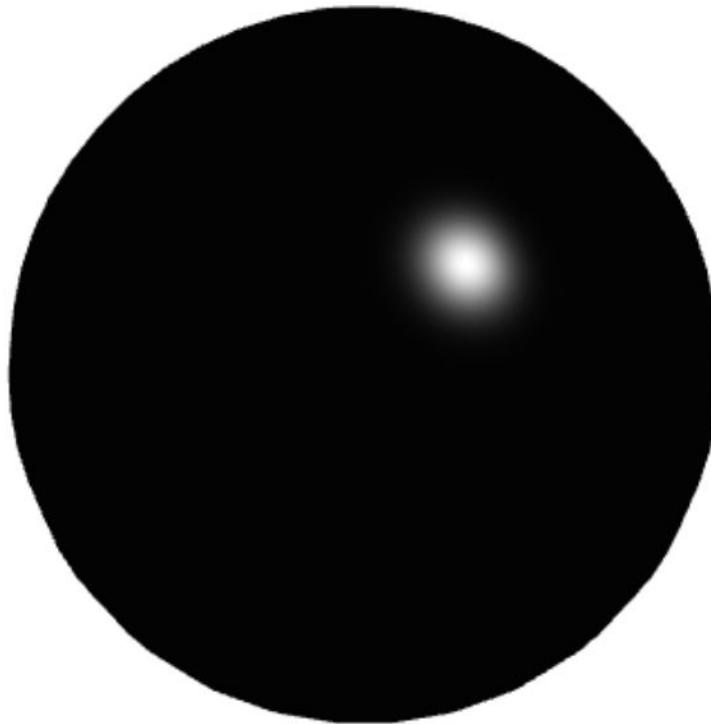
A medida que la constante especular, α , se va haciendo más grande, la curva se va estrechando alrededor del eje y_

- **Valores pequeños de α :** producen un brillo mate que se desvanece a una distancia relativamente grande.
- **Valores grandes de α :** generan un brillo más definido y pequeño que se desvanece rápido cuando los valores de **R** y **V** divergen.

Al igual que en la luz difusa, si $\mathbf{N} \cdot \mathbf{L} < 0$, quiere decir que la fuente de luz está detrás de la superficie, de modo que no presentaría brillo especular.



Luz especular



Modelo de iluminación de Phong

En el mundo real, las superficies no son puramente difusas o especulares, sino que suelen estar compuestas por ambas propiedades. Además, la reflexión de la luz en los diferentes objetos del entorno genera una componente de luz ambiental. Tomando esto con consideración, Phong describió la luz reflejada en un punto P de la superficie de un objeto como:

$$I_p = I_{\text{ambiental}} + I_{\text{difusa}} + I_{\text{especular}}$$

$$I_p = k_a \cdot I_a + k_d \cdot I_{\text{luz}} \cdot \max(0, N \cdot L) + k_s \cdot I_{\text{luz}} \cdot (R \cdot V)^\alpha$$

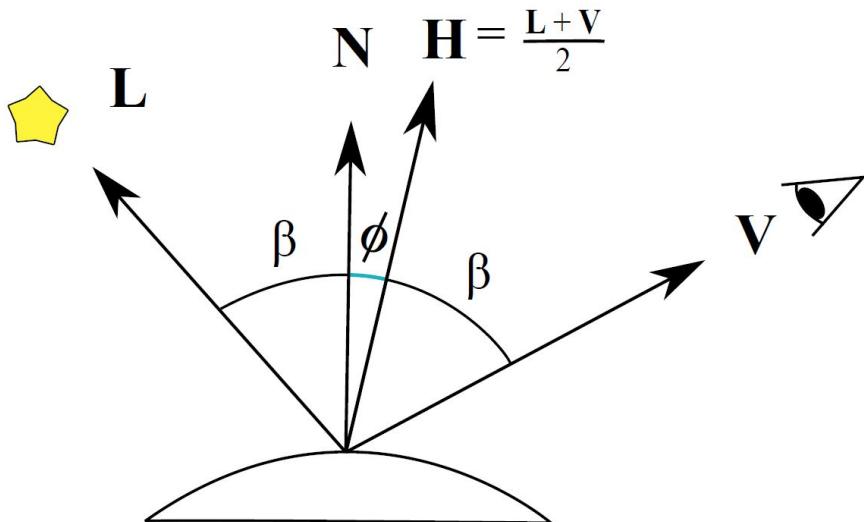
Y, en el caso de tener múltiples fuentes de luz, la reflexión de cada una de estas luces en P se acumula para obtener la cantidad total de luz reflejada:

$$I_p = k_a \cdot I_a + \sum_{i=1}^n [k_d \cdot I_{\text{luz}} \cdot \max(0, N \cdot L) + k_s \cdot I_{\text{luz}} \cdot (R \cdot V)^\alpha]$$

Modelo de iluminación de Phong



Modelo de iluminación de Blinn-Phong



El **modelo de iluminación de Blinn-Phong** es una variante del modelo de Phong propuesta por James Blinn que busca simplificar el cálculo de la iluminación especular.

Para ello, en vez de calcular el vector de reflexión, **R**, para cada punto de la superficie, calcula el vector intermedio o half vector, **H**, como el vector que se encuentra entre el vector de vista, **V**, y el de la luz incidente, **L**. Es decir, $\mathbf{H} = (\mathbf{L} + \mathbf{V}) / 2$.

De este modo, la reflexión especular se calcularía en función del ángulo formado entre el vector intermedio, **H**, y la normal del punto, **N**, como:

$$I_p = k_s \cdot I_{luz} \cdot \max(0, \mathbf{N} \cdot \mathbf{H})^\alpha$$

Este modelo permite simplificar los cálculos, por lo que se ha vuelto el más utilizado en gráficos **3D** en tiempo real.

Modelo de Phong vs modelo de Blinn - Phong

Modelo de Phong



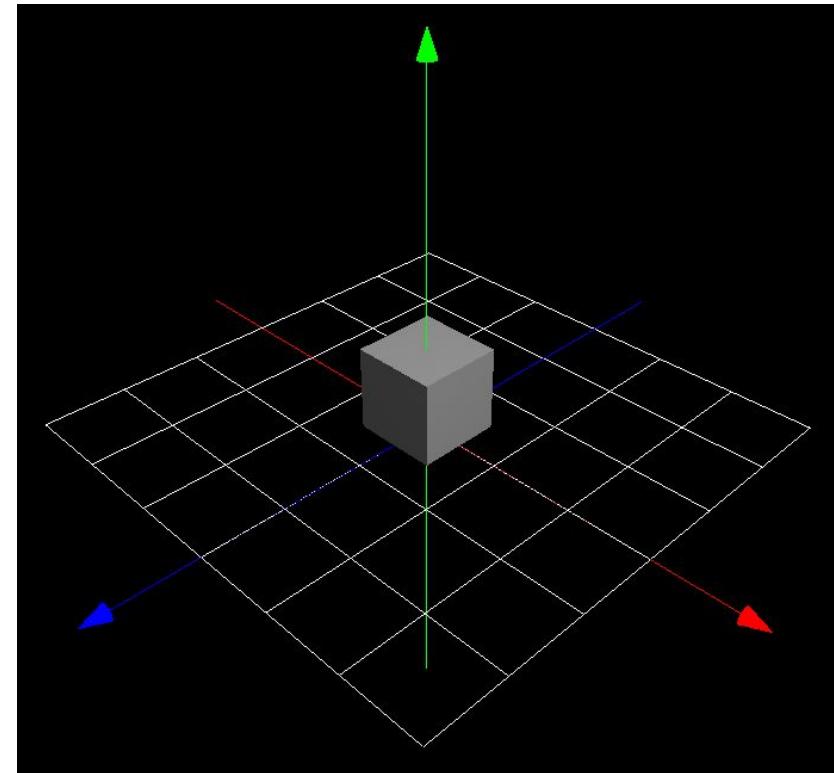
Modelo de Blinn - Phong



Implementación del modelo de Phong en OpenGL

La configuración típica para implementar el modelo de iluminación Phong en OpenGL comprende los siguientes pasos:

- Definición de normales.
- Configuración de las fuentes de luz.
- Activación de la iluminación.
- Suavización de sombreados (opcional).



Definición de normales

```
37     # Definición de normales
38     normales = [
39         # Cara R (Right): x = +0.5
40         (1, 0, 0), (1, 0, 0),
41         # Cara L (Left): x = -0.5
42         (-1, 0, 0), (-1, 0, 0),
43         # Cara U (Up): y = +0.5
44         (0, 1, 0), (0, 1, 0),
45         # Cara D (Down): y = -0.5
46         (0, -1, 0), (0, -1, 0),
47         # Cara F (Front): z = +0.5
48         (0, 0, 1), (0, 0, 1),
49         # Cara B (Back): z = -0.5
50         (0, 0, -1), (0, 0, -1),
51     ]
```

Un vector normal es un vector perpendicular a la superficie de un objeto.

En los gráficos **3D**, las normales se utilizan para describir la orientación de una superficie, y se pueden asignar a cada cara o a cada vértice

En el modelo de Phong las normales son necesarias para calcular las componentes de luz difusa y especular.

En OpenGL las normales se pueden asignar mediante la función **glNormal3f()**.

Definición de normales

```
53     def cubo():
54         """Dibuja el modelo del cubo utilizando triángulos con iluminación."""
55
56         Recorre la lista de triángulos y dibuja cada uno en el espacio 3D utilizando
57         la función GL_TRIANGLES de OpenGL, que conecta los tres vértices de cada triángulo.
58         """
59
60         for i in range(len(triangulos)):
61             glBegin(GL_TRIANGLES)
62                 # Asigna la normal correspondiente a la cara del cubo
63                 glNormal3fv(normales[i])
64                 for indice in triangulos[i]:
65                     glVertex3fv(vertices[indice]) # Define cada uno de los vértices del triángulo
66             glEnd()
```

Configuración de las fuentes de luz

El siguiente paso consiste en configurar las fuentes de luz mediante la función `glLightfv()`:

`glLightfv(light, pname, params)`: permite definir las propiedades de una fuente de luz, donde:

- **light**: especifica el número de la fuente de luz (`GL_LIGHT0`, `GL_LIGHT1`, ...)
- **pname**: especifica el parámetro que se desea configurar en la fuente de luz, como `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR` o `GL_POSITION`.
- **params**: especifica el valor que se desea asignar al parámetro.

```
5 def configurar_luces(): 4 usages
6     """Configura las luces de la escena en OpenGL."""
7     # Luz ambiente
8     luz_ambiente = [0.2, 0.2, 0.2, 1.0]
9     glLightfv(GL_LIGHT0, GL_AMBIENT, luz_ambiente)
10
11    # Luz difusa.
12    luz_difusa = [0.8, 0.8, 0.8, 1.0]
13    glLightfv(GL_LIGHT0, GL_DIFFUSE, luz_difusa)
14
15    # Luz especular
16    luz_especular = [1.0, 1.0, 1.0, 1.0]
17    glLightfv(GL_LIGHT0, GL_SPECULAR, luz_especular)
18
19    # Posición de la luz (x, y, z, w)
20    posicion_luz = [5.0, 5.0, 5.0, 1.0]
21    glLightfv(GL_LIGHT0, GL_POSITION, posicion_luz)
```

Activación de la iluminación

```
23     # Activa la luz 0 (GL_LIGHT0)
24     glEnable(GL_LIGHT0)
25
26     # Activa la iluminación en general
27     glEnable(GL_LIGHTING)
```

A continuación debemos activar la iluminación (en general), y cada una de las fuentes de luz específicas:

- **glEnable(GL_LIGHTING)**: activa la iluminación general.
- **glEnable(GL_LIGHT0)**: activa una fuente de luz específica.

Suavización de sombreados (opcional)

Opcionalmente, podemos indicar a OpenGL que aplique un sombreado suave o interpolado, también conocido como sombreado de Gouraud.

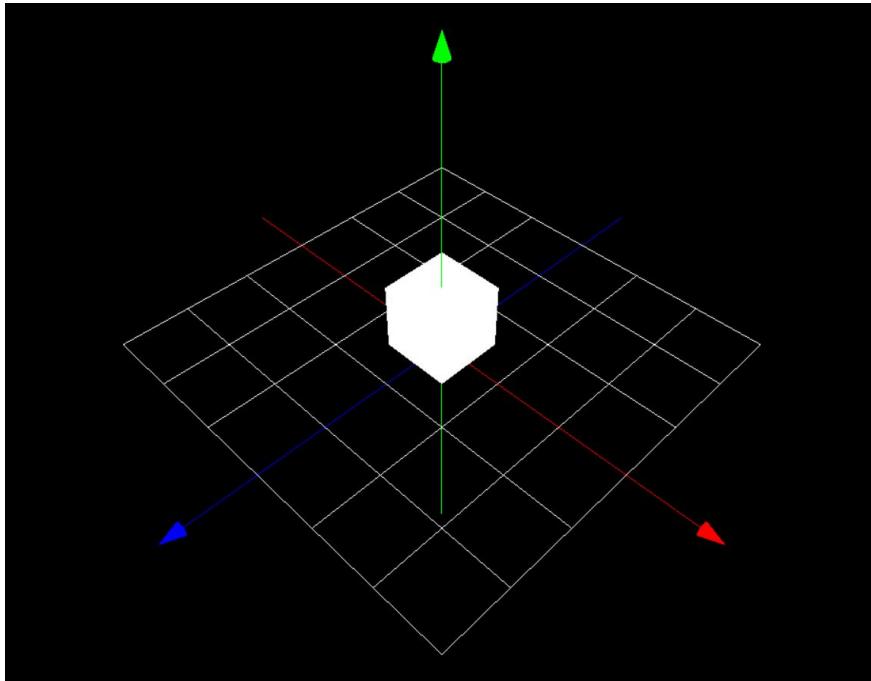
- **glShadeModel(GL_SMOOTH)**: indica a OpenGL que debe interpolar los colores entre los vértices de las primitivas, lo que da como resultado una transición suave de un color a otro.

El sombreado de Gouraud proporciona una transición de color e iluminación más suave y continua, proporcionando una apariencia más realista y natural al objeto.

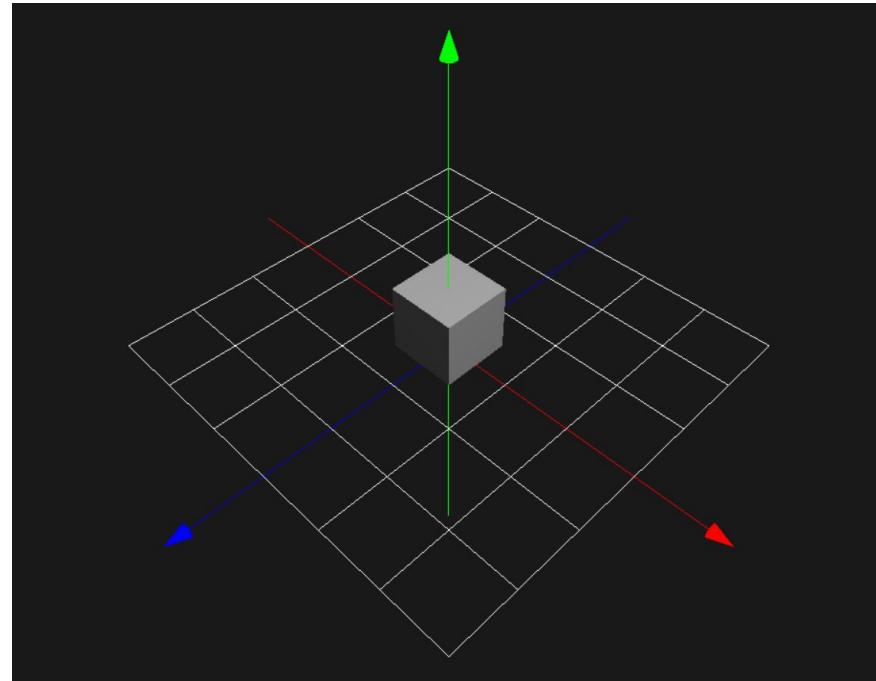
```
20     # Inicialización de la escena
21     def inicializar_escena():
22         """Iniciaiza la ventana gráfica con Pygame y configura los ajustes de OpenGL para la escena 3D.
23
24     Returns:
25         screen: Objeto de la ventana gráfica creada por Pygame.
26         """
27
28     pygame.init() # Inicializa Pygame
29     # Crea la ventana con doble buffer y compatibilidad con OpenGL
30     screen = pygame.display.set_mode(size=(SCREEN_WIDTH, SCREEN_HEIGHT), DOUBLEBUF | OPENGL)
31     pygame.display.set_caption('Visualizador 3D') # Establece el título de la ventana
32
33     # Configuración de OpenGL
34     glClearColor(red=0, green=0, blue=0, alpha=1) # Fondo negro
35     glEnable(GL_DEPTH_TEST) # Activa el z-buffer para la profundidad
36     glShadeModel(GL_SMOOTH) # Activa el sombreado suave
37     glMatrixMode(GL_PROJECTION) # Selecciona la matriz de proyección
38     gluPerspective(FOV, SCREEN_ASPECT_RATIO, NEAR_PLANE, FAR_PLANE) # Configura la proyección en perspectiva
39     glMatrixMode(GL_MODELVIEW) # Selecciona la matriz de modelo-vista
40     glLoadIdentity() # Restablece la matriz a la identidad
41
42     configurar_luces() # Activa la configuración de luces después de inicializar OpenGL
43
44     return screen # Devuelve la referencia a la ventana creada
```

Renderizado sin iluminación vs renderizado con iluminación local (Phong)

`main.py`

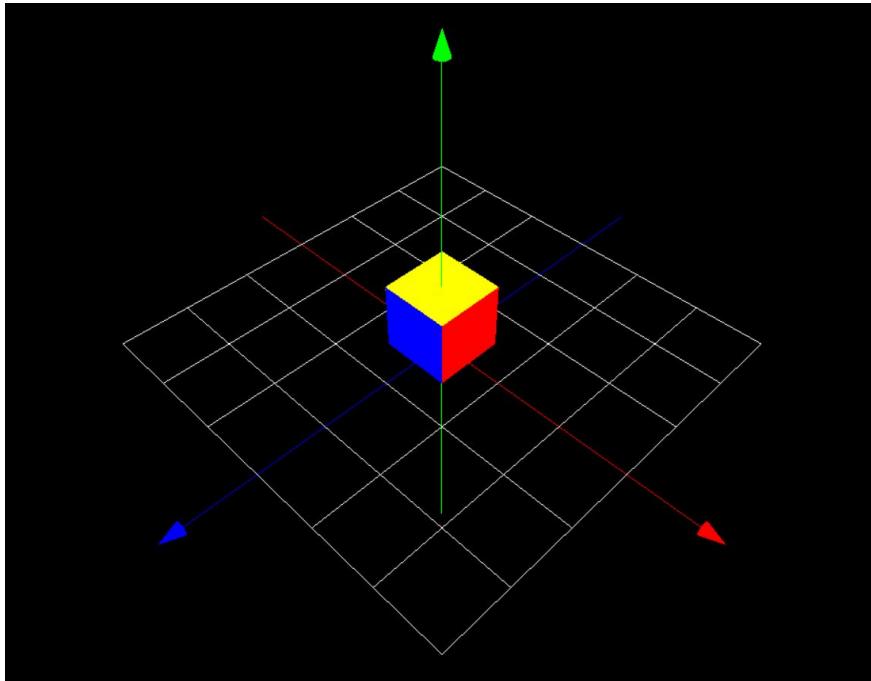


`main_phong.py`

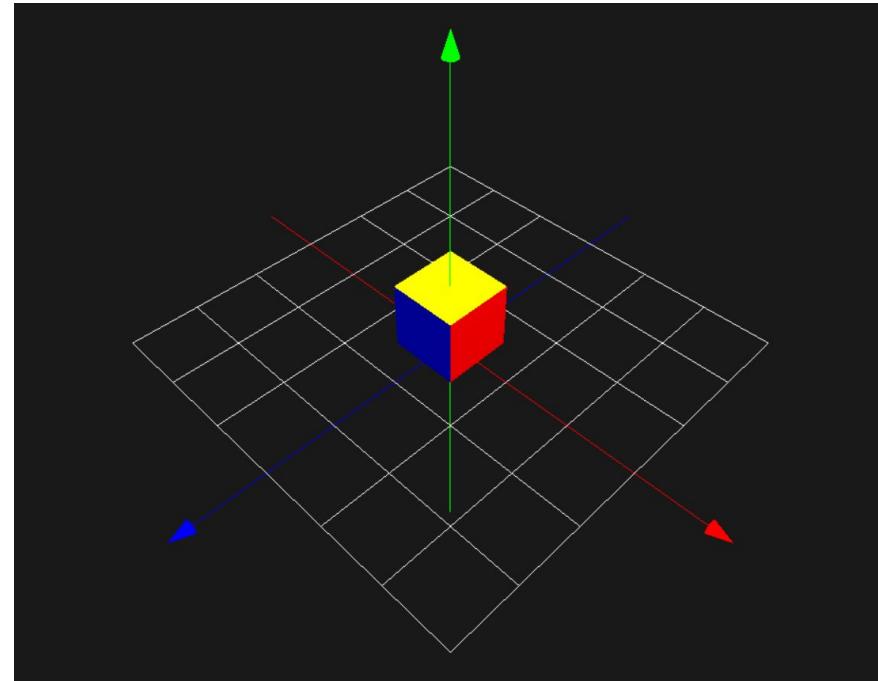


Renderizado sin iluminación vs renderizado con iluminación local (Phong)

`main_color.py`

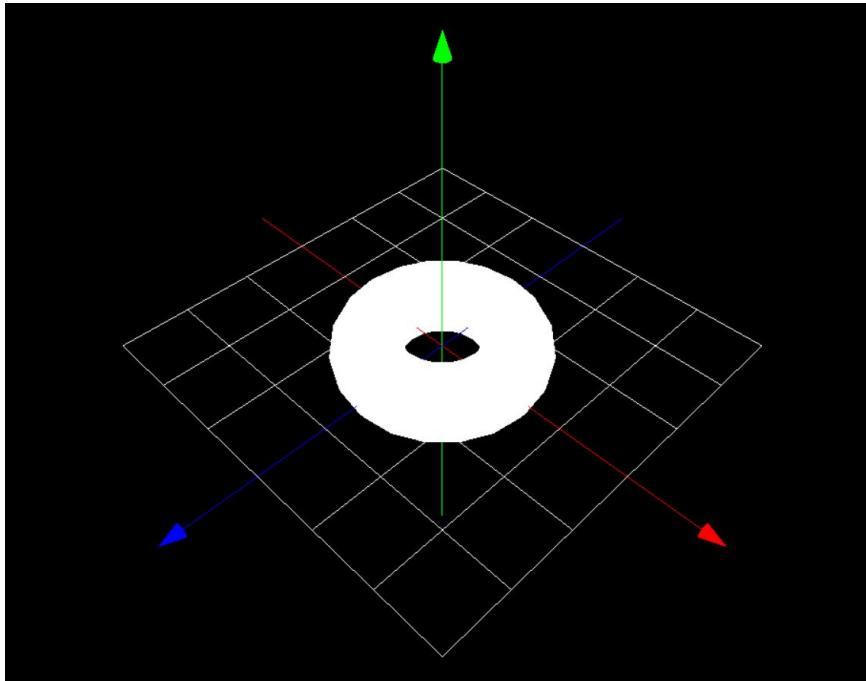


`main_color_phong.py`

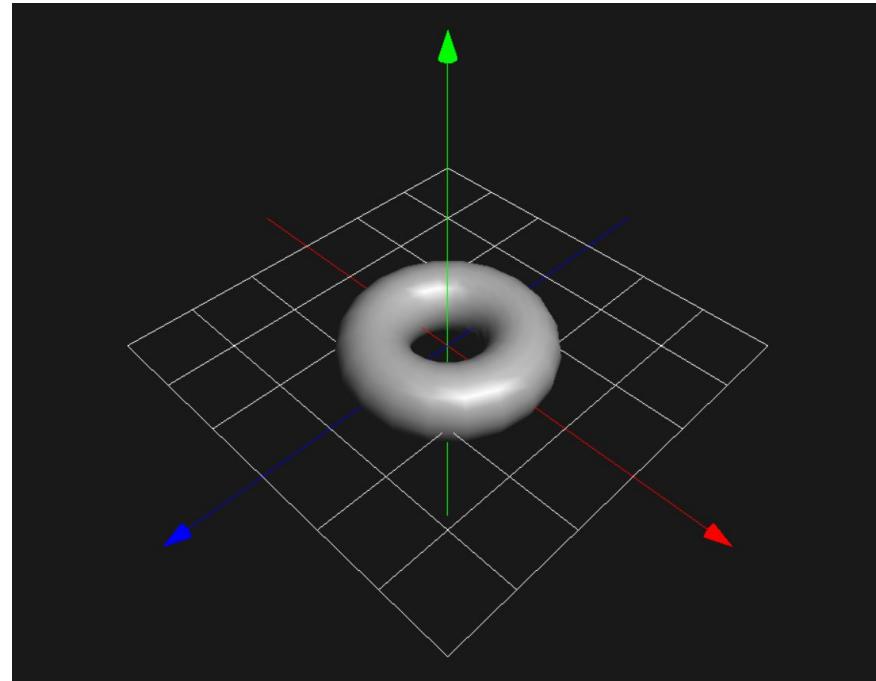


Renderizado sin iluminación vs renderizado con iluminación local (Phong)

`main_obj.py`

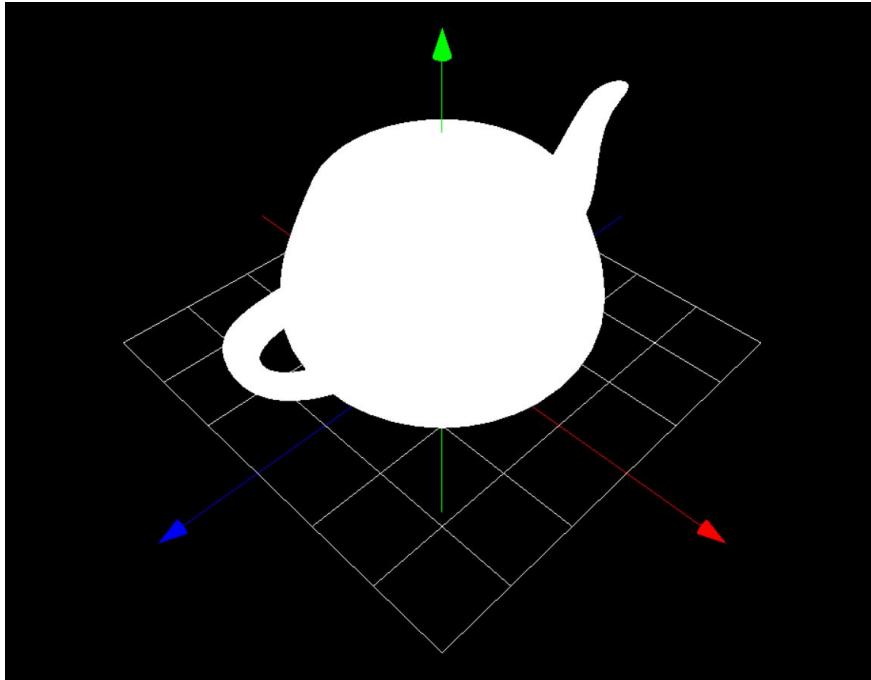


`main_obj_phong.py`

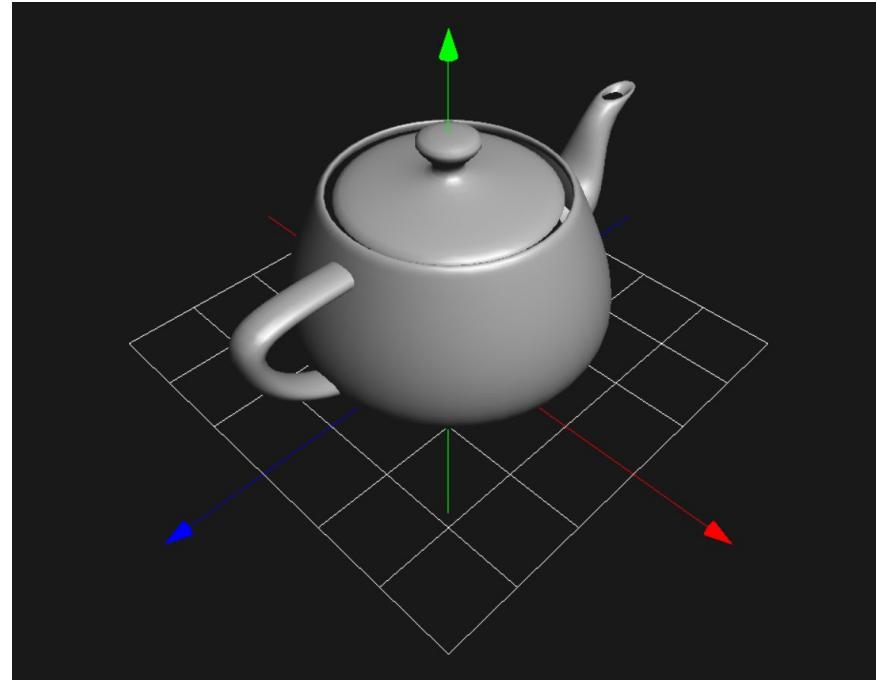


Renderizado sin iluminación vs renderizado con iluminación local (Phong)

`main_obj.py`

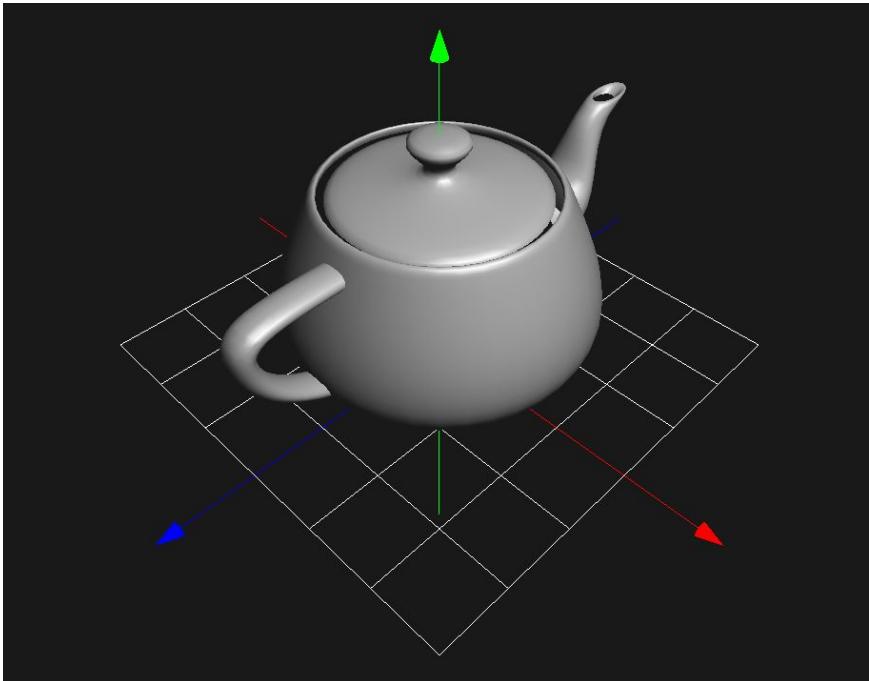


`main_obj_phong.py`

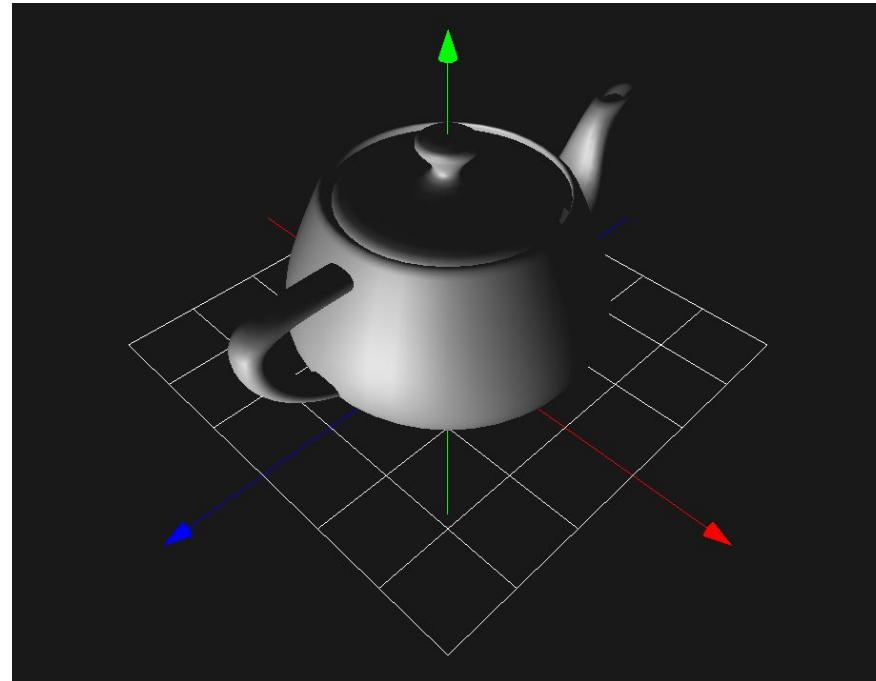


Variación de la posición de la luz

POSICION_LUZ = [0.5, 0.5, 0.5, 1.0]

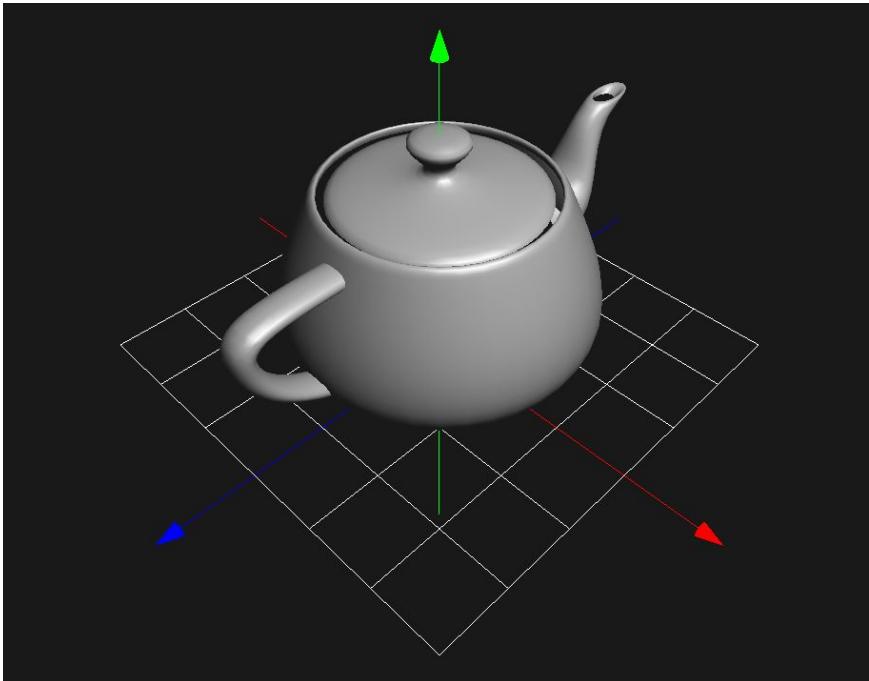


POSICION_LUZ= [-0.5, -0.5, -0.5, 1.0]

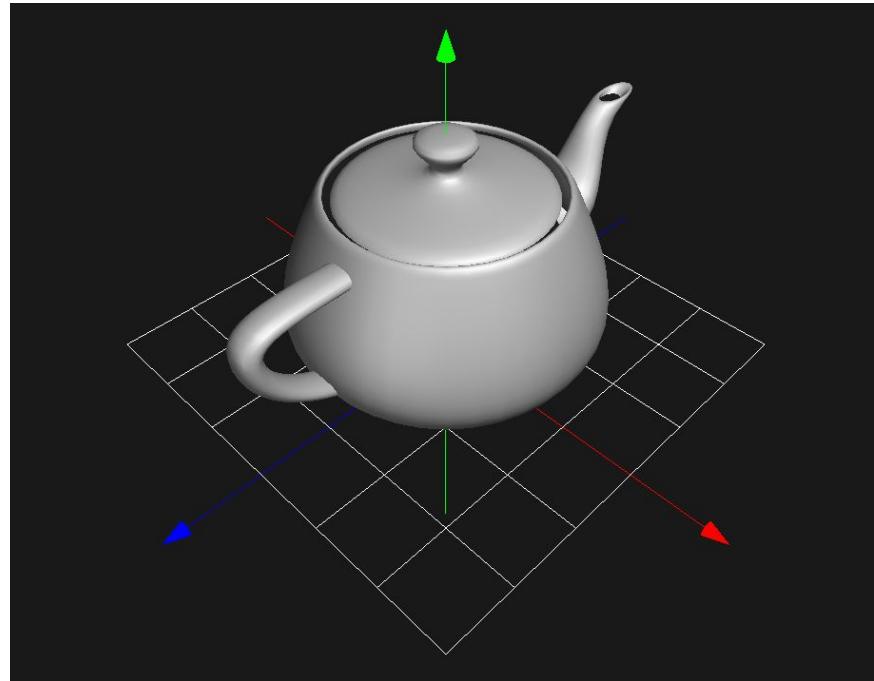


Variación del número de luces

Una luz

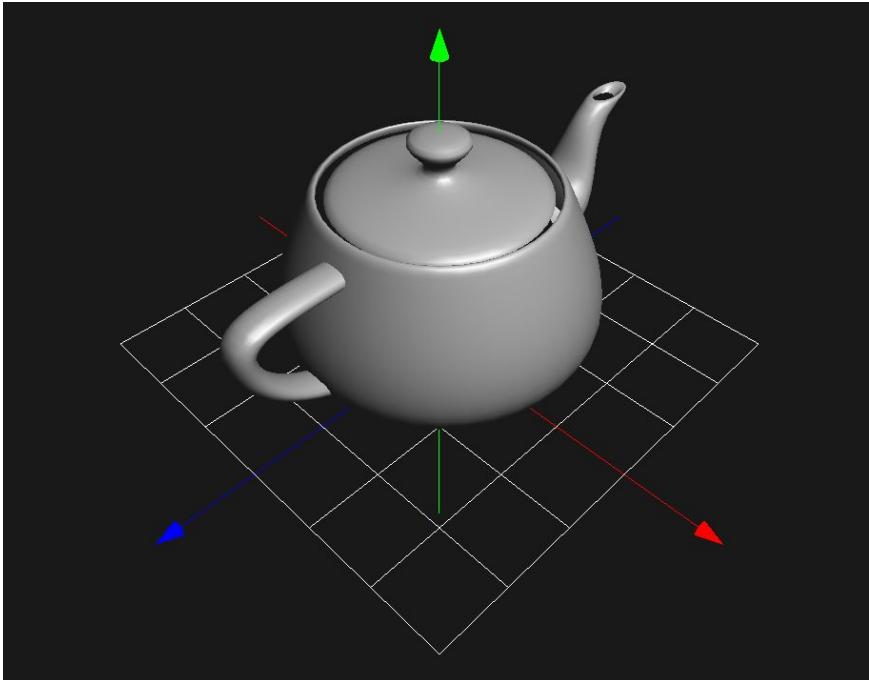


Dos luces

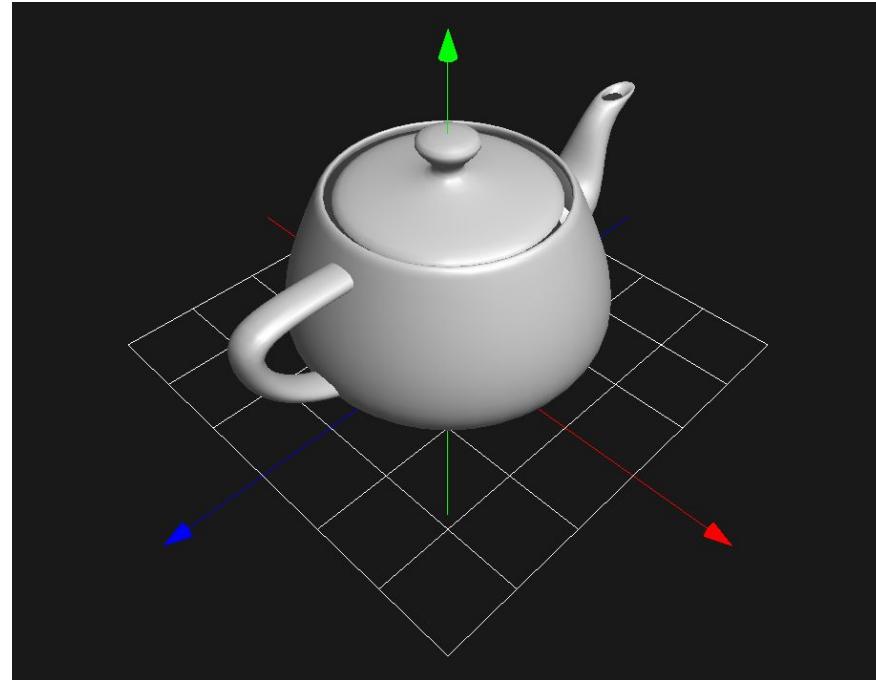


Variación de la intensidad de la luz ambiental

LUZ_AMBIENTE = [0.3, 0.3, 0.3, 1.0]

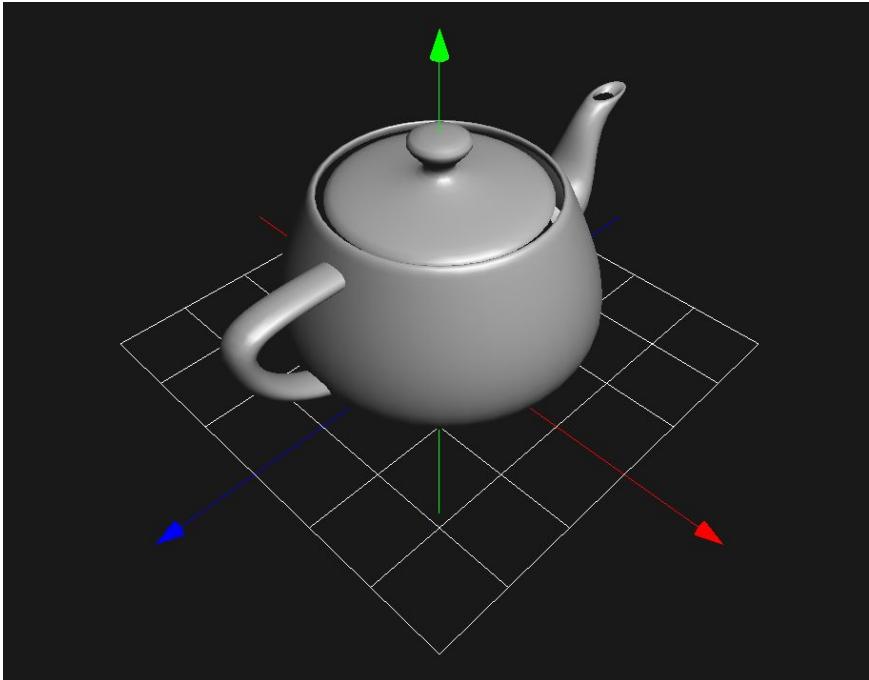


LUZ_AMBIENTE = [1.0, 1.0, 1.0, 1.0]

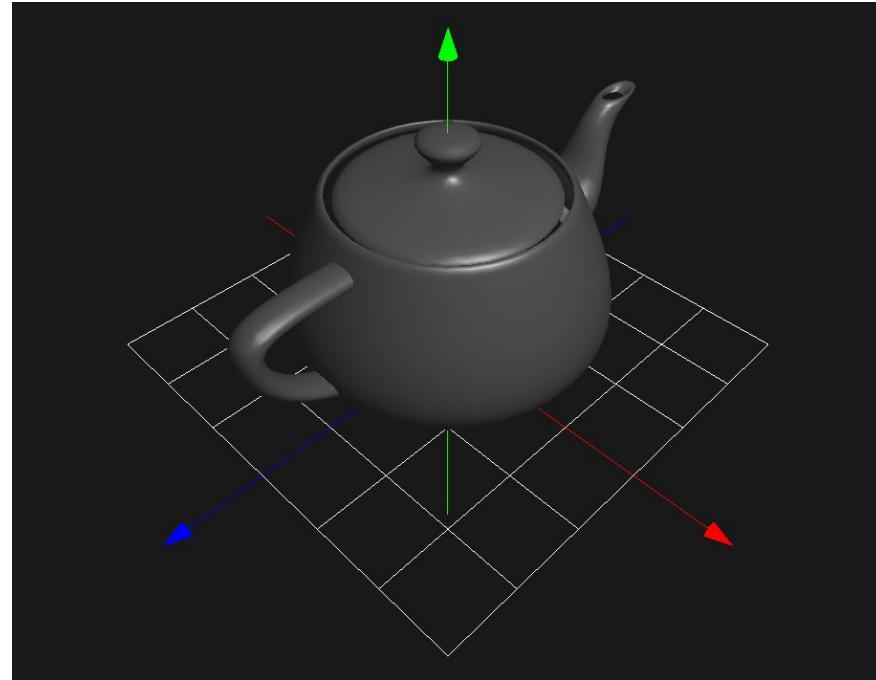


Variación de la intensidad de la luz difusa

LUZ_DIFUSA = [0.8, 0.8, 0.8, 1.0]

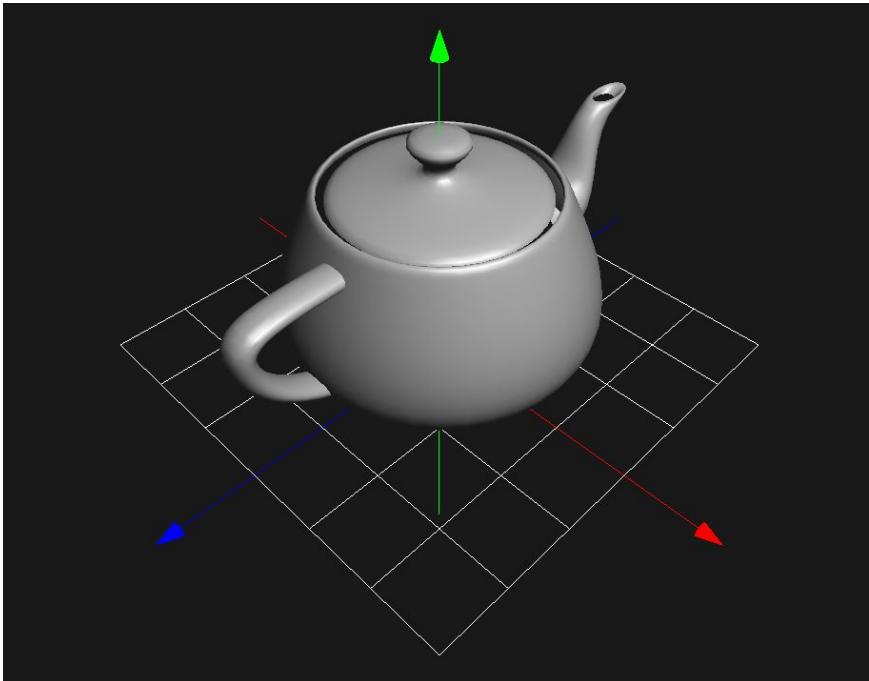


LUZ_DIFUSA = [0.3, 0.3, 0.3, 1.0]

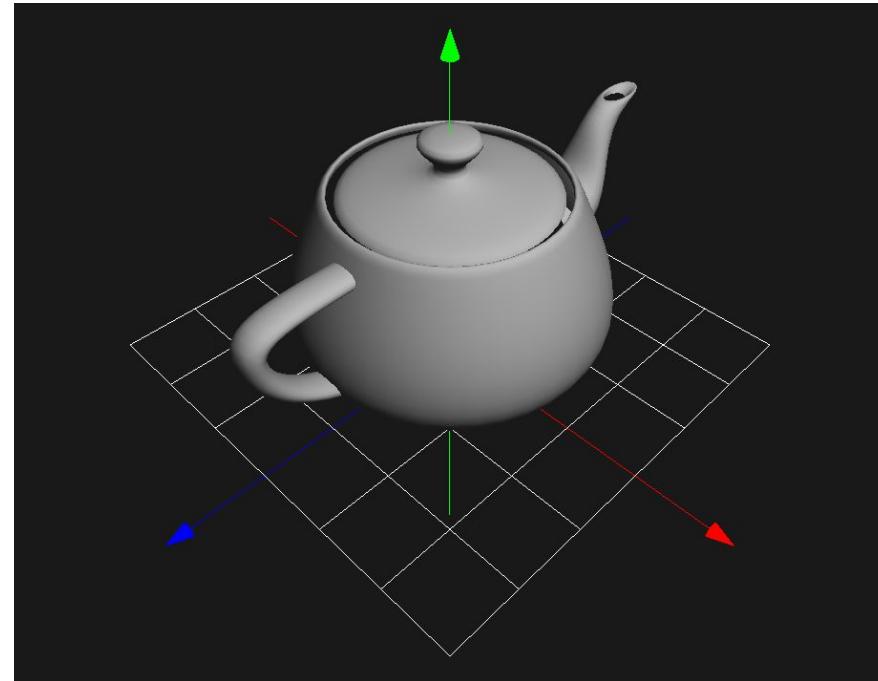


Variación del coeficiente especular

COEFICIENTE_ESPECULAR = [0.5, 0.5, 0.5, 1.0]

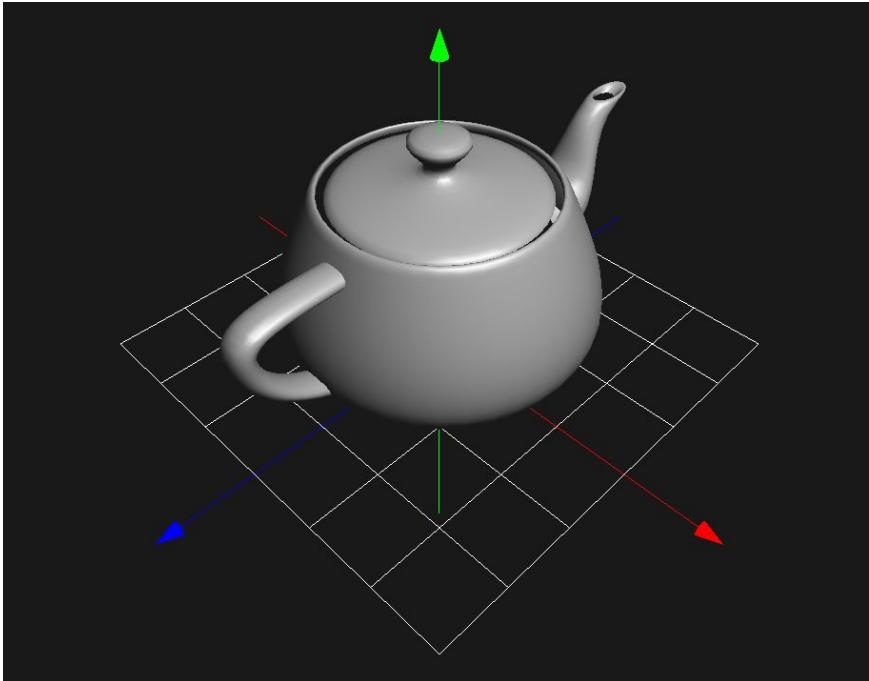


COEFICIENTE_ESPECULAR = [0.0, 0.0, 0.0, 1.0]

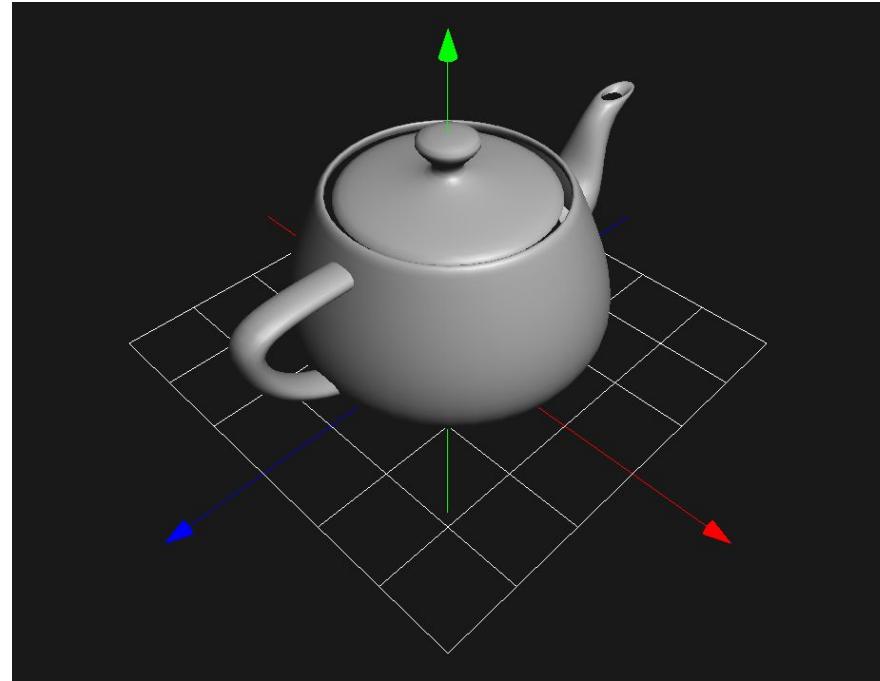


Variación del exponente de brillo

EXPONENTE_BRILLO = 10.0



EXPONENTE_BRILLO = 50.0

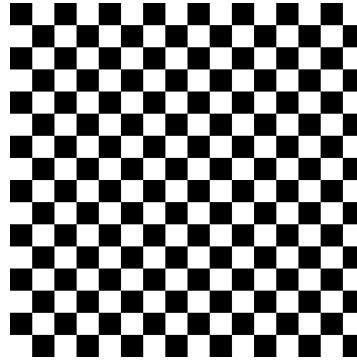


Introducción a las texturas

Texturas

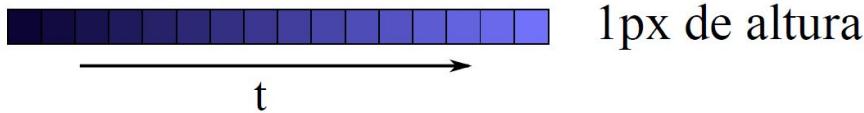
Hasta ahora, hemos asumido que todas las superficies reflejan la luz de manera uniforme en cada punto. Sin embargo, en el mundo real, esto es una rareza.

- Las superficies reales presentan **variaciones reflectivas** en cada punto y generalmente presentan **imperfecciones**.
- La **representación geométrica** de estos detalles es **compleja y computacionalmente costosa**.
- El **mapeo de texturas** permite representar las propiedades reflectivas en cada punto **sin necesidad de utilizar geometría adicional**, mejorando el realismo en tiempo real.

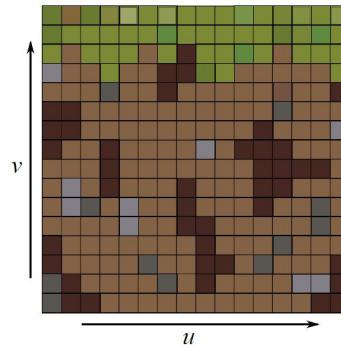


Mapeo de texturas

Mapa de texturas 1D



Mapa de texturas 2D



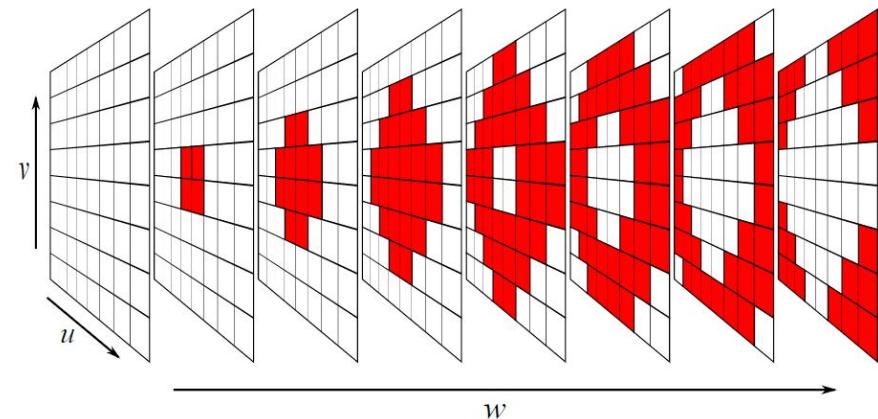
El **mapeo de texturas** o *texture mapping* es una técnica que permite modelar las variaciones de material y acabados de una superficie mediante la especificación de las propiedades reflectivas y otros atributos de cada punto de la superficie en mapas de textura, o simplemente texturas.

- Los mapas de textura son estructuras de datos de **1, 2, 3** dimensiones donde se almacena la información que deseamos mapear a una superficie.
- Por ejemplo, si deseamos colorear un terreno en función de la altitud del terreno en cada punto, podemos utilizar un mapa de texturas **1D**.
- Los mapas de textura más utilizados son las texturas **2D**, que consisten en imágenes rasterizadas que representan una matriz bidimensional de píxeles con las que podemos “envolver” nuestro objeto.

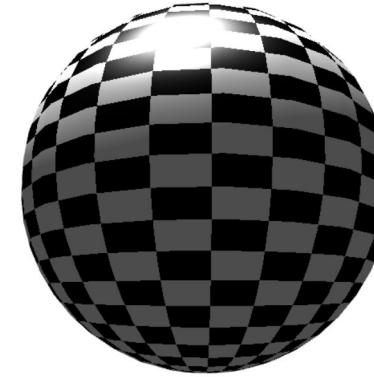
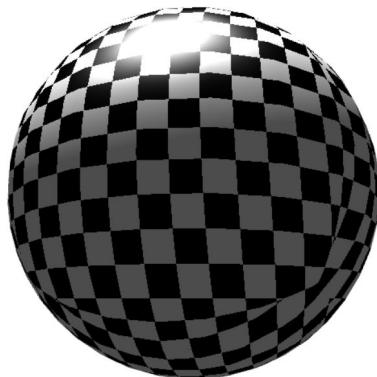
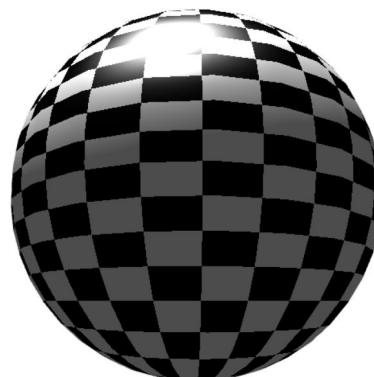
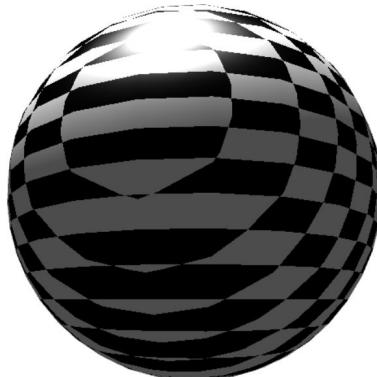
Texturas 3D

Las texturas **3D**, o texturas volumétricas, son matrices tridimensionales de píxeles, en las que cada punto de textura se define a través de sus coordenadas **x, y, z**.

Si bien este tipo de texturas puede entenderse como un conjunto de texturas **2D** apiladas, requieren una cantidad de almacenamiento significativo, por lo que en vez de imágenes se suelen utilizar funciones que permiten para mapear cada coordenada a un color específico.



Función de proyección



El proceso de renderizado de una textura comienza asignando a cada punto de la superficie un punto correspondiente en la textura mediante una **función de proyección**.

Una función de proyección es una función matemática que transforma puntos de una superficie tridimensional en coordenadas bidimensionales de textura en el rango **[0,0]x [1,1]**, conocido como espacio **UV** o **ST**.

Función de proyección

Las principales funciones de proyección que podemos aplicar a las texturas son:



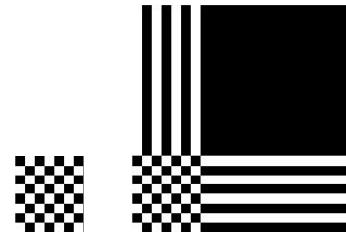
- **Proyección planar:** proyecta la textura sobre el objeto a lo largo de un plano, similar a proyectar una diapositiva sobre él. Es ideal para superficies planas o relativamente planas.
- **Proyección cilíndrica:** envuelve la textura alrededor del objeto siguiendo una forma cilíndrica. Es útil para objetos con simetría radial, como columnas o botellas.
- **Proyección esférica:** proyecta la textura de manera esférica, adecuada para objetos redondeados como globos o planetas.
- **Proyección cúbica:** utiliza seis proyecciones planas correspondientes a las caras de un cubo, comúnmente usada en mapas de entorno para simular reflejos.

Modos de envoltura de texturas

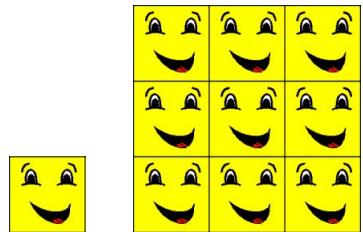
Dado que el espacio de textura **2D** es un espacio rectangular definido dentro del rango $[0,0] \times [1,1]$, debemos especificar qué sucede cuando nos salimos de este rango mediante los llamados **modos de envoltura de texturas** (*texture wrapping modes*):

- **Clamp:** las coordenadas de textura que están fuera del rango se limitan al valor más cercano dentro de este.
- **Repeat:** la textura se repite continuamente más allá de sus bordes originales.
- **Mirror:** la textura se repite pero de forma espejada en cada repetición.
- **Border:** las coordenadas de textura fuera del rango se asignan a un color de borde específico.

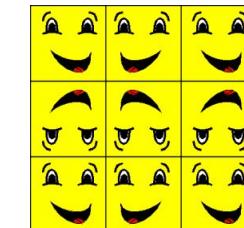
CLAMP



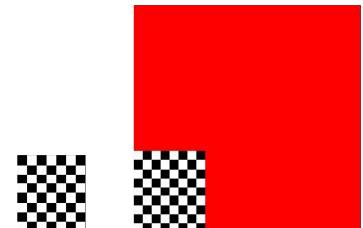
REPEAT



MIRROR



BORDER



Implementación de texturas de imagen en OpenGL

La aplicación **3D** desarrollada para la asignatura implementa texturas **2D** o texturas de imagen en tres pasos:

1. Carga de la textura desde un archivo de imagen.
2. Aplicación de la textura al modelo.
3. Renderizado de la escena.

Carga de la textura.

La función responsable de cargar la textura es **cargar_textura()** en el archivo **texturas.py**:

- **Carga de imagen:** utiliza la función **pygame.image.load()** para cargar la imagen de textura a partir de un archivo.
- **Conversión de datos:** convierte la imagen en una secuencia de bytes adecuada para OpenGL con **pygame.image.tostring()**.
- **ID de textura:** genera un identificador de textura única único mediante **glGenTextures()**.
- **Configuración de OpenGL:** enlaza la textura con **glBindTexture()**, especifica los parámetros de la textura con **glTexImage2D()**, y define los modos de envoltura vertical y horizontal y los filtros de magnificación y minificación

Carga de la textura.

```
7 def cargar_textura(ruta_textura): 2 usages
8
9     # Carga la imagen desde el archivo usando Pygame.
10    textura_surface = pygame.image.load(ruta_textura)
11
12    # Convierte la imagen en una cadena de bytes en formato RGB para su uso en OpenGL.
13    textura_data = pygame.image.tostring(textura_surface, format: "RGB", flipped: 1)
14
15    # Genera un identificador único para la textura en OpenGL.
16    textura_id = glGenTextures(1)
17
18    # Configura y enlaza la textura en OpenGL usando el ID generado.
19    glBindTexture(GL_TEXTURE_2D, textura_id)
20
21    # Carga la textura en OpenGL.
22    glTexImage2D(GL_TEXTURE_2D, level: 0, GL_RGB, textura_surface.get_width(),
23                 textura_surface.get_height(), border: 0, GL_RGB,
24                 GL_UNSIGNED_BYTE, textura_data)
25
26    # Configura los parámetros de la textura en OpenGL.
27    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
28    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)
29    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
30    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
31
32    # Devuelve el identificador de textura (textura_id) para que pueda ser utilizado en otras partes de la aplicación
33    return textura_id
```

Aplicación de la textura al modelo.

La gestión del modelo 3D se lleva a cabo en la clase **Modelo**, definida en el archivo `modelo_phong_texturas.py`:

- **Lectura de coordenadas de textura:** el método `cargar_modelo()` lee las coordenadas de textura (indicadas por líneas `vt` en el archivo), y las almacena en la lista `coordenadas_textura`.
- **Renderizado del modelo con texturas:** en el método `_dibujar_objeto()` se aplica la textura a cada vértice mediante la función `glTexCoord2fv()`, que usa las coordenadas almacenadas en la lista `coordenadas_textura`.
- **Activación y desactivación de la textura:** en el método `_dibujar_objeto()` se activa el modo de texturas en **OpenGL** con `glEnable(GL_TEXTURE_2D)` y se asigna el identificador de textura `textura_id` con `glBindTexture()`. Al finalizar el renderizado del objeto se desactiva el modo de textura con `glDisable(GL_TEXTURE_2D)` para evitar que se aplique a otros elementos no texturizados.

Aplicación de la textura al modelo.

```
57     def _dibujar_objeto(self, textura_id): 1 usage
58         """Dibuja cada triángulo del modelo con texturas."""
59         glEnable(GL_NORMALIZE)
60         glEnable(GL_TEXTURE_2D)
61         glBindTexture(GL_TEXTURE_2D, textura_id)
62
63         for vert_indices, norm_indices, tex_indices in self.triangles:
64             glBegin(self.draw_type)
65             for vi, ni, ti in zip(vert_indices, norm_indices, tex_indices):
66                 glNormal3fv(self.normales[ni]) # Configura la normal del vértice
67                 glTexCoord2fv(self.coordenadas_textura[ti]) # Configura la coordenada de textura
68                 glVertex3fv(self.vertices[vi]) # Configura el vértice
69             glEnd()
70
71         glDisable(GL_TEXTURE_2D)
```

Renderizado de la escena

El renderizado de la escena completa es llevado a cabo por la función **renderizar()** del fichero **main_obj_phong_texturas.py**:

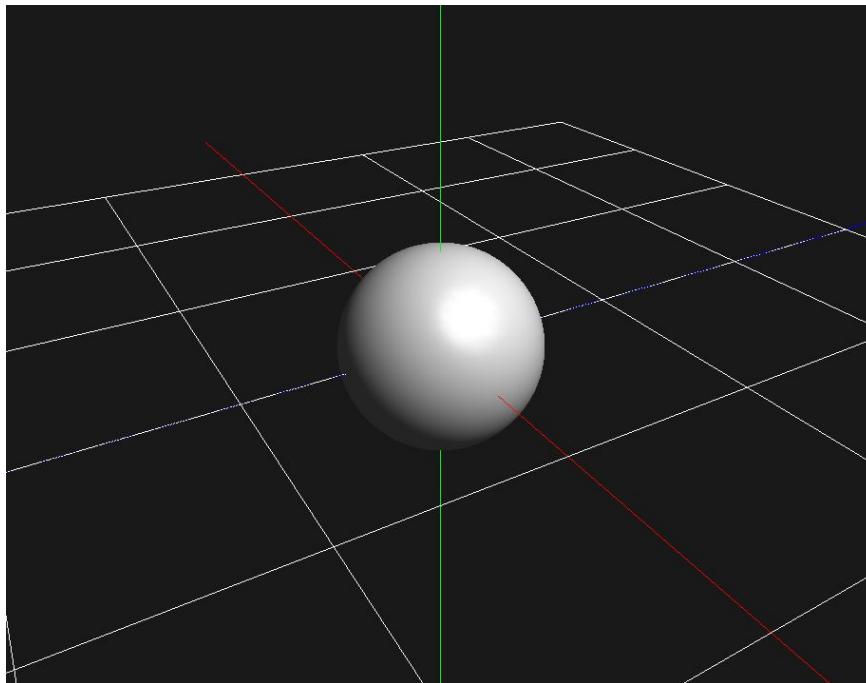
- **Borra los búfers de color y profundidad** con **glClear()**.
- **Orienta la cámara** con **glRotatef()** y **glLookAt()**.
- **Dibuja los elementos auxiliares** previa desactivación del sombreado llamando a la **funcion dibujar_elementos_auxiliares()** del fichero utilidades.
- **Dibuja el modelo** aplicando las transformaciones deseadas llamando al método **dibujar()** de la clase **Modelo**.

Renderizado de la escena

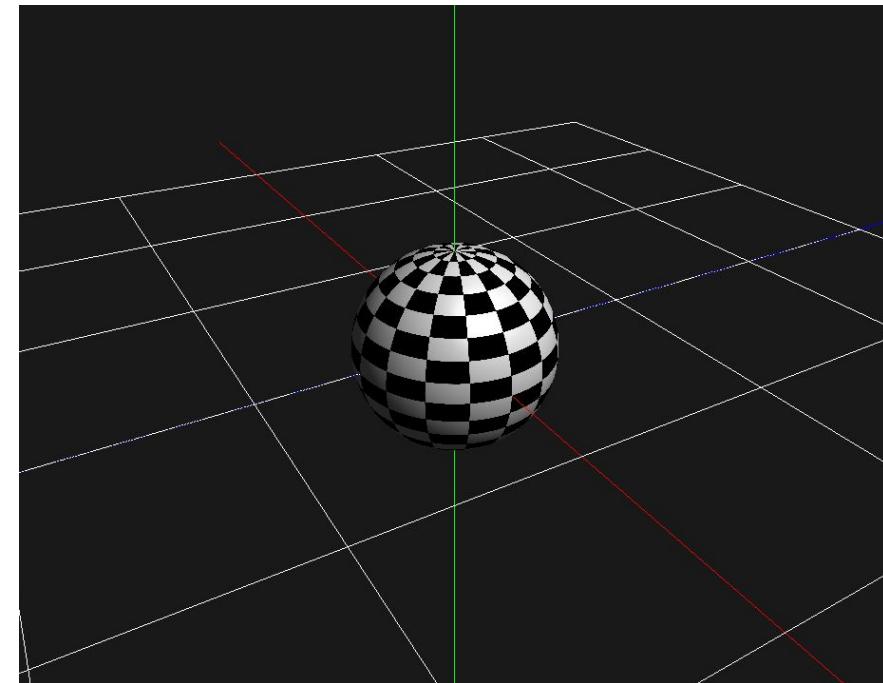
```
48     # Renderiza la escena
49     def renderizar():
50         """Renderiza los elementos de la escena, incluyendo la cámara, elementos auxiliares y el modelo 3D."""
51         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
52         glLoadIdentity()
53
54         glRotatef(camara.roll, 0, 0, 1)
55         cam_x, cam_y, cam_z = camara.obtener_posicion()
56         gluLookAt(cam_x, cam_y, cam_z, 0, 0, 0, 0, 1, 0)
57
58         glDisable(GL_LIGHTING)
59         dibujar_elementos_auxiliares(ejes=True, rejilla=True)
60         glEnable(GL_LIGHTING)
61
62     # Dibujar el modelo con la textura aplicada
63     modelo.dibujar(textura_id = textura_id, t_x=0, t_y=0, t_z=0,
64                     angulo=0, eje_x=0, eje_y=0, eje_z=0, sx=1, sy=1, sz=1)
```

Renderizado sin textura vs renderizado con textura

Sin textura

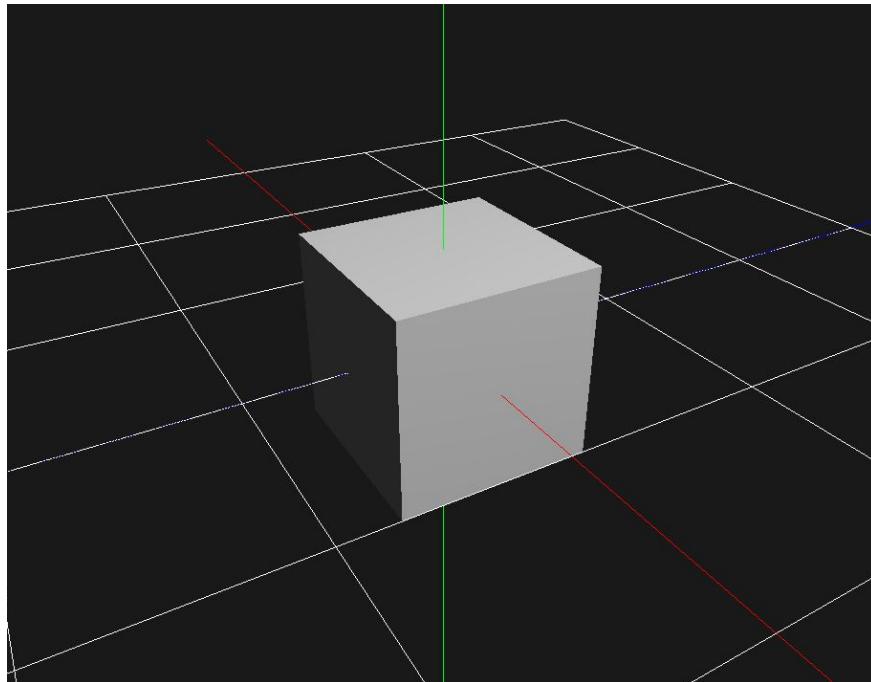


Textura de ajedrez

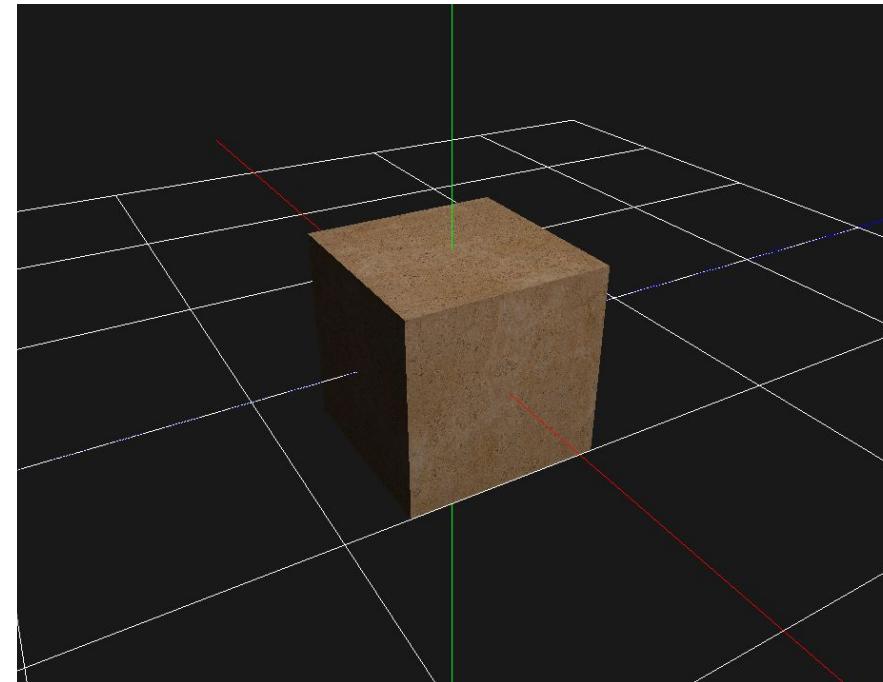


Renderizado sin textura vs renderizado con textura

Sin textura

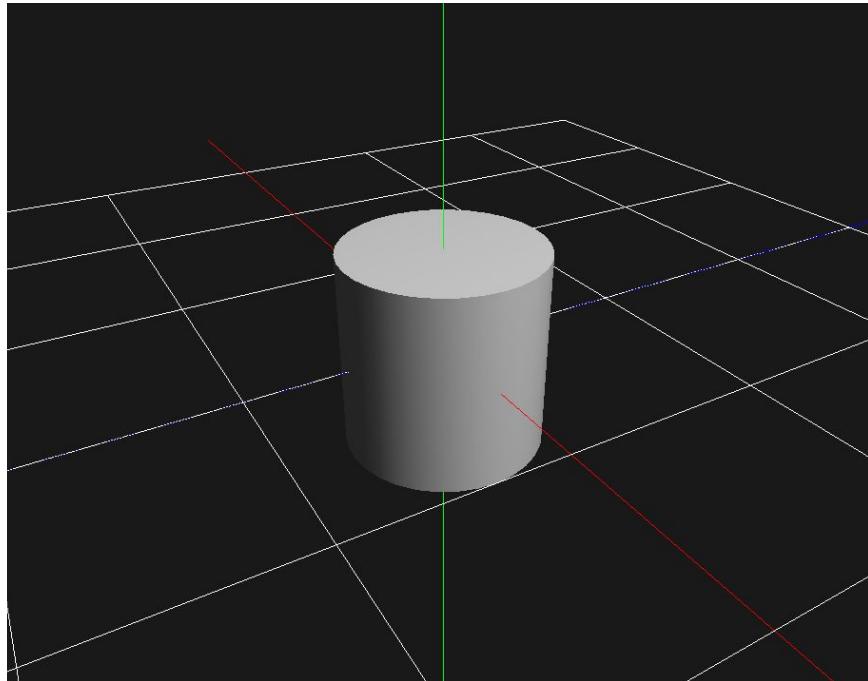


Textura de piedra

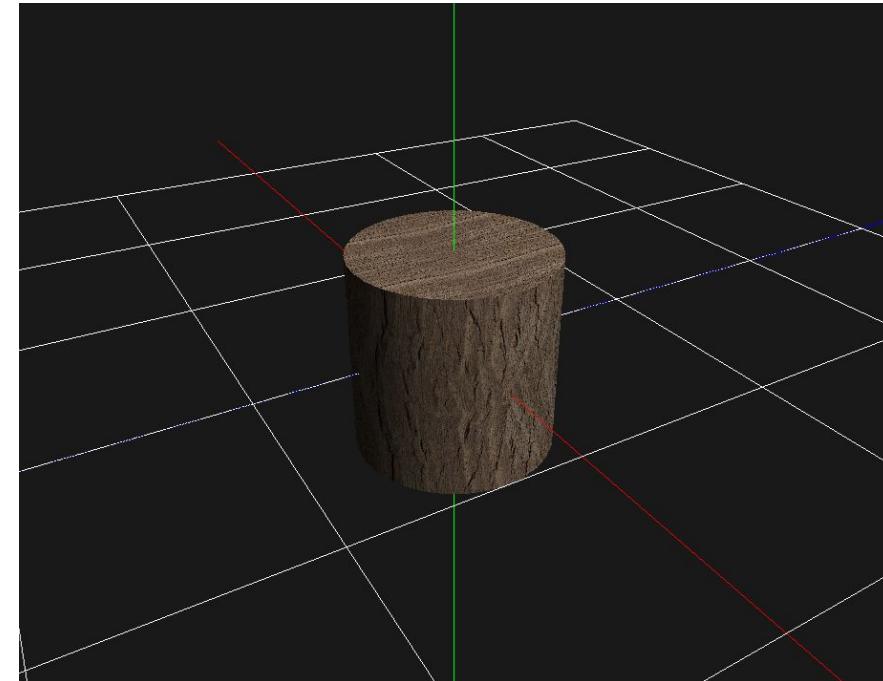


Renderizado sin textura vs renderizado con textura

Sin textura

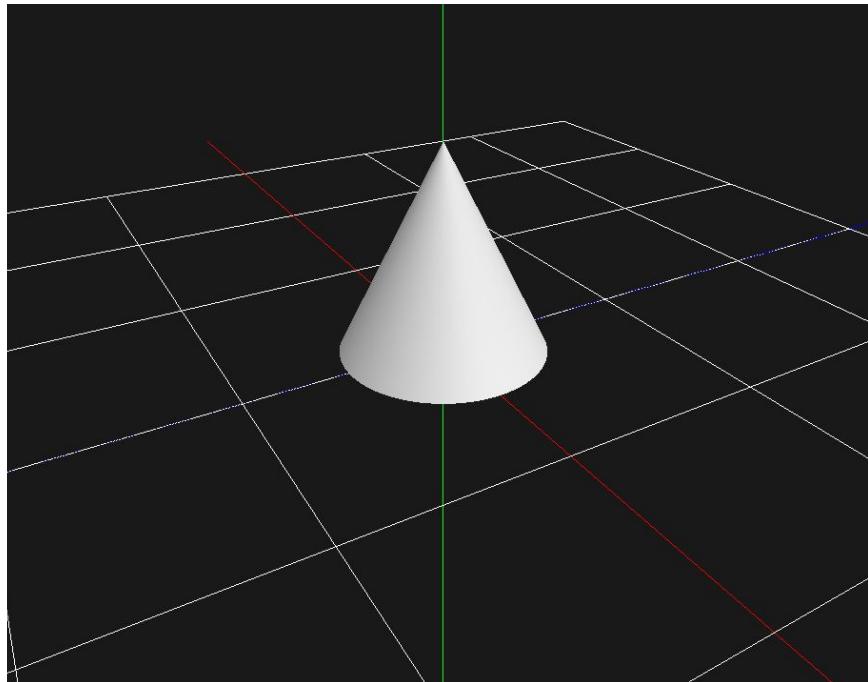


Textura de madera

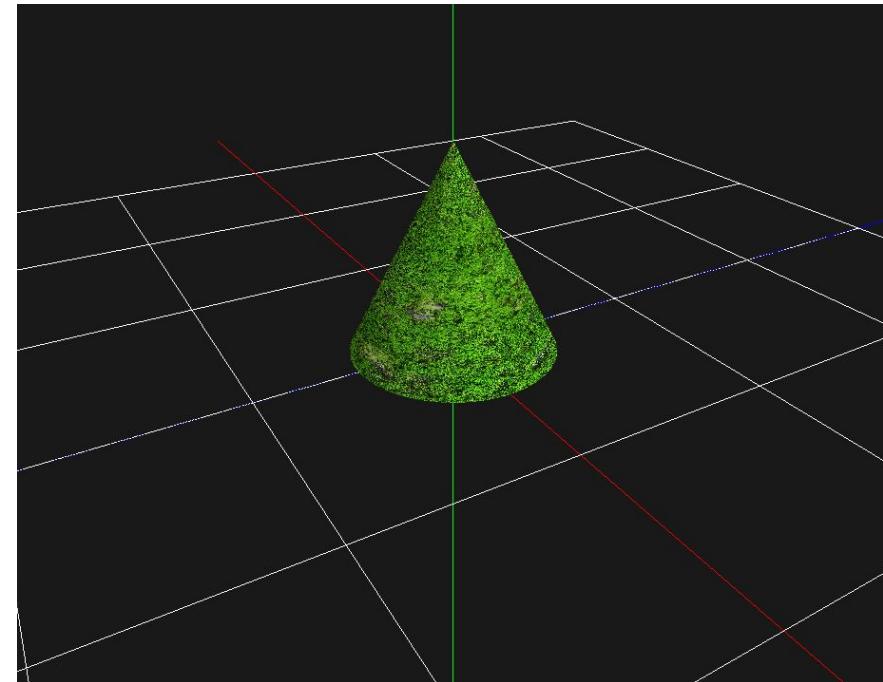


Renderizado sin textura vs renderizado con textura

Sin textura



Textura de musgo



Referencias bibliográficas

- Foley, J.D., Van Dam, A., Feiner, S.K., Hughes, J.F., Phillips, R.L. (1993). *Introduction to Computer Graphics*. Addison-Wesley Publishing Company.
- Méndez, M. (2022). *Introducción a la graficación por computadora*.
<https://proyectodescartes.org/iCartesiLibri/PDF/GraficacionComputadora.pdf>