Línea 1: import numpy as np

- Importa la biblioteca `numpy` y le asigna el alias `np`. NumPy es esencial para operaciones numéricas eficientes, especialmente con arrays multidimensionales, que se utilizarán para representar la imagen, coordenadas, vectores, etc.

Línea 2: import matplotlib.pyplot as plt

- Importa el submódulo `pyplot` de la biblioteca `matplotlib` y le asigna el alias `plt`. Este módulo se utilizará para mostrar la imagen renderizada de la esfera.

Línea 3: (En blanco)

- Línea en blanco: Separa la sección de importaciones de la definición de parámetros iniciales, mejorando la legibilidad.

Línea 4: # --- Parámetros Iniciales ---

- Comentario: Indica el inicio de una sección donde se definirán varias constantes y parámetros que controlarán el renderizado.

Línea 5: (En blanco)

- Línea en blanco: Separa el comentario de la sección de los primeros parámetros.

Línea 6: # Dimensiones de la imagen y parámetros de la esfera

- Comentario: Especifica que los siguientes parámetros se refieren al tamaño de la imagen a generar y a las propiedades geométricas de la esfera.

Línea 7: img_size = 400

- Define una variable `img_size` y le asigna el valor entero `400`. Esto determinará que la imagen generada será de 400x400 píxeles.

Línea 8: radius = 1.0

- Define una variable `radius` y le asigna el valor flotante `1.0`. Este es el radio de la esfera que se va a renderizar.

Línea 9: center = np.array([0.0, 0.0, 0.0]) # Asumiendo centro en el origen para simplificar normales

- Define una variable `center` y le asigna un array NumPy que representa las coordenadas (x, y, z) del centro de la esfera. Se establece en `[0.0, 0.0, 0.0]`, es decir, el origen.

- El comentario `# Asumiendo centro en el origen para simplificar normales` explica que centrar la esfera en el origen simplifica el cálculo de los vectores normales a su superficie (el vector normal en un punto de una esfera centrada en el origen es simplemente el vector desde el origen hasta ese punto, normalizado).

Línea 10: (En blanco)

- Línea en blanco: Separa los parámetros de la esfera de los parámetros de iluminación.

Línea 11: # Parámetros de iluminación

- Comentario: Indica que los siguientes parámetros definen las propiedades de la luz y del material en la escena.

Línea 12: ambient_intensity = 0.2 # Componente ambiental global

- Define una variable `ambient_intensity` y le asigna el valor flotante `0.2`.

- El comentario `# Componente ambiental global` explica que este valor representa la intensidad de la luz ambiental, que ilumina todos los puntos de la escena de manera uniforme, independientemente de la posición de la fuente de luz o de la orientación de la superficie.

Línea 13: kd = 0.7 # Coeficiente difuso del material

- Define una variable `kd` y le asigna el valor flotante `0.7`.

- El comentario `# Coeficiente difuso del material` explica que `kd` es el coeficiente de reflexión difusa del material de la esfera. Este coeficiente determina qué fracción de la luz incidente se dispersa de manera uniforme en todas las direcciones.

Línea 14: ks = 0.5 # Coeficiente especular del material

- Define una variable `ks` y le asigna el valor flotante `0.5`.

- El comentario `# Coeficiente especular del material` explica que `ks` es el coeficiente de reflexión especular del material. Este coeficiente determina la intensidad del brillo especular (highlight) que aparece en la superficie.

Línea 15: specular_exponent = 32 # Exponente para el brillo especular (shininess)

- Define una variable `specular_exponent` y le asigna el valor entero `32`.

- El comentario `# Exponente para el brillo especular (shininess)` explica que este valor, también conocido como "shininess" o "índice de Phong", controla el tamaño y la nitidez del brillo especular. Valores más altos producen brillos más pequeños y concentrados.

Línea 16: (En blanco)

- Línea en blanco: Separa los coeficientes de material de las posiciones de luz y observador.

Línea 17: # Posición de la fuente de luz y del observador (cámara)

- Comentario: Indica que los siguientes parámetros definen la ubicación espacial de la fuente de luz y del punto desde el cual se observa la escena (la cámara o el ojo).

Línea 18: light_pos = np.array([2.0, 2.0, 2.0])

- Define una variable `light_pos` y le asigna un array NumPy que representa las coordenadas (x, y, z) de la fuente de luz en la escena: `[2.0, 2.0, 2.0]`.

Línea 19: eye_pos = np.array([0.0, 0.0, 5.0]) # Observador en el eje Z positivo

- Define una variable `eye_pos` y le asigna un array NumPy que representa las coordenadas (x, y, z) de la posición del observador (cámara): `[0.0, 0.0, 5.0]`.

- El comentario `# Observador en el eje Z positivo` aclara que la cámara está situada a lo largo del eje Z, mirando hacia el origen.

Línea 20: (En blanco)

- Línea en blanco: Separa las posiciones de los colores.

Línea 21: # Color del objeto (AHORA AZUL) y del highlight especular (blanco)

- Comentario: Indica que los siguientes parámetros definen los colores base del objeto y del brillo especular. Destaca un cambio en el color del objeto.

Línea 22: object_color = np.array([0.0, 0.0, 1.0]) # <--- CAMBIO AQUÍ: Color Azul

- Define una variable `object_color` y le asigna un array NumPy que representa el color RGB del objeto (la esfera): `[0.0, 0.0, 1.0]`. Esto corresponde a azul puro (R=0, G=0, B=1).

- El comentario `# <--- CAMBIO AQUÍ: Color Azul` enfatiza que este es el nuevo color asignado al objeto.

Línea 23: specular_color = np.array([1.0, 1.0, 1.0]) # Ks implícito en este color

- Define una variable `specular_color` y le asigna un array NumPy que representa el color RGB del brillo especular: `[1.0, 1.0, 1.0]`. Esto corresponde a blanco puro.

- El comentario `# Ks implícito en este color` sugiere que, aunque `ks` es un coeficiente escalar, el color del brillo especular también puede influir en su apariencia final. En muchos modelos, el color especular modula la intensidad del

componente especular, de forma similar a como el color del objeto modula el componente difuso. Aquí, parece que `specular_color` define el color del reflejo mismo, y `ks` su intensidad.

Línea 24: (En blanco)

- Línea en blanco: Separa la sección de parámetros iniciales de la sección de preparación de geometría e imagen.

Línea 25: # --- Preparación de la Geometría y la Imagen ---

- Comentario: Indica el inicio de una sección donde se configurarán las estructuras de datos para la imagen y se calcularán las coordenadas 3D de la superficie visible de la esfera.

Línea 26: (En blanco)

- Línea en blanco: Separa el comentario de la sección de las primeras operaciones de preparación.

Línea 27: # Crear una cuadrícula 2D representando la proyección en el plano XY

- Comentario: Explica que las siguientes líneas generarán una malla de coordenadas (x, y) que servirá como base para proyectar la esfera y determinar qué píxeles le corresponden.

Línea 28: x = np.linspace(-radius, radius, img_size)

- Crea un array NumPy llamado `x` utilizando `np.linspace`. Este array contendrá `img_size` (400) valores espaciados uniformemente entre `-radius` (-1.0) y `radius` (1.0). Estos representan las coordenadas x de los píxeles en el espacio del objeto.

Línea 29: y = np.linspace(-radius, radius, img_size)

- Crea un array NumPy llamado `y` de manera similar a `x`, también con `img_size` valores entre `-radius` y `radius`. Estos representan las coordenadas y de los píxeles.

Línea 30: X, Y = np.meshgrid(x, y)

- Utiliza `np.meshgrid` para crear dos arrays 2D, `X` e `Y`, a partir de los arrays 1D `x` e `y`.

- `X` será un array de `img_size` x `img_size` donde cada fila es una copia de `x`.

- `Y` será un array de `img_size` x `img_size` donde cada columna es una copia de `y`.

- Juntos, `X[i, j]` e `Y[i, j]` proporcionan las coordenadas (x, y) del píxel en la posición (fila i, columna j) de la imagen.

Línea 31: (En blanco)

- Línea en blanco: Separa la creación de la cuadrícula de la inicialización de la imagen.

Línea 32: # Inicializar la imagen (RGB)

- Comentario: Indica que la siguiente línea crea el array que almacenará los valores de color de cada píxel de la imagen resultante.

Línea 33: image = np.zeros((img_size, img_size, 3))

- Crea un array NumPy tridimensional llamado `image` e inicializa todos sus elementos a cero.

- Las dimensiones son `(img_size, img_size, 3)`: `img_size` filas, `img_size` columnas, y 3 canales de color (R, G, B).

- Esto prepara un lienzo negro sobre el cual se dibujará la esfera iluminada.

Línea 34: (En blanco)

- Línea en blanco: Separa la inicialización de la imagen del cálculo de la coordenada Z.

Línea 35: # Calcular la coordenada z de la esfera

- Comentario: Explica que las siguientes líneas calcularán la coordenada Z para cada punto (X, Y) de la cuadrícula que se encuentre sobre la superficie de la esfera.

Línea 36: z_squared = radius**2 - X**2 - Y**2

- Calcula `z_squared` para cada punto (X, Y) de la cuadrícula. La ecuación de una esfera centrada en el origen es `x^2 + y^2 + z^2 = radius^2`. Despejando `z^2`, obtenemos `z^2 = radius^2 - x^2 - y^2`.

- `X**2` y `Y**2` calculan el cuadrado de cada elemento en los arrays `X` e `Y` respectivamente. Las operaciones se realizan elemento a elemento.

- `z_squared` será un array 2D del mismo tamaño que `X` e `Y`.

Línea 37: Z = np.sqrt(np.maximum(0, z_squared))

- Calcula la coordenada `Z` tomando la raíz cuadrada de `z_squared`.

- `np.maximum(0, z_squared)`: Antes de tomar la raíz cuadrada, se asegura de que todos los valores en `z_squared` sean no negativos. Si `X**2 + Y**2 > radius**2` (puntos fuera de la proyección del disco de la esfera), `z_squared` sería negativo, y su raíz cuadrada indefinida (o compleja). `np.maximum` reemplaza estos valores negativos con 0. Esto efectivamente considera solo la mitad frontal de la esfera (Z >= 0).

- `np.sqrt(...)` calcula la raíz cuadrada elemento a elemento.

- `Z` es un array 2D que contiene la coordenada z de la superficie de la esfera para cada par (X, Y) (o 0 si el punto (X,Y) está fuera del disco de la esfera).

Línea 38: (En blanco)

- Línea en blanco: Separa el cálculo de Z de la creación de la máscara.

Línea 39: # Crear una máscara para los puntos que pertenecen a la esfera

- Comentario: Indica que la siguiente línea creará un array booleano para identificar qué píxeles de la cuadrícula (X, Y) corresponden realmente a la proyección de la esfera.

Línea 40: mask = X**2 + Y**2 <= radius**2

- Crea un array booleano 2D llamado `mask` del mismo tamaño que `X` e `Y`.

- Para cada píxel (i, j), `mask[i, j]` será `True` si `X[i, j]^2 + Y[i, j]^2 <= radius^2` (es decir, si el punto (X, Y) está dentro o sobre el borde del círculo que es la proyección de la esfera en el plano XY).

- `mask[i, j]` será `False` si el punto está fuera de este círculo. Esta máscara es crucial para aplicar cálculos solo a los píxeles que representan la esfera.

Línea 41: (En blanco)

- Línea en blanco: Separa la creación de la máscara de la creación del array de puntos 3D.

Línea 42: # Posición de cada punto en la superficie visible de la esfera

- Comentario: Indica que la siguiente línea combinará las coordenadas X, Y, y Z calculadas en un único array que representa los puntos 3D en la superficie de la esfera.

Línea 43: points = np.stack([X, Y, Z], axis=-1)

- Crea un array NumPy tridimensional llamado `points`.

- `np.stack([X, Y, Z], axis=-1)` apila los arrays 2D `X`, `Y`, y `Z` a lo largo de un nuevo último eje (el eje -1 o 2).

- El resultado `points` tendrá dimensiones `(img_size, img_size, 3)`. Para cada píxel `(i, j)`, `points[i, j]` será un array de 3 elementos `[X[i, j], Y[i, j], Z[i, j]]`,

representando la coordenada 3D del punto en la superficie de la esfera correspondiente a ese píxel. Si el píxel está fuera de la esfera (según la máscara), `Z[i,j]` será 0, y este punto no corresponderá a la superficie esférica real si X e Y son grandes. La `mask` se usará más adelante para seleccionar solo los puntos relevantes.

Línea 44: (En blanco)

- Línea en blanco: Separa la preparación de la geometría de la sección de cálculo de normales.

Línea 45: # --- Cálculo de Normales ---

- Comentario: Indica el inicio de una sección dedicada al cálculo de los vectores normales a la superficie de la esfera en cada punto visible.

Línea 46: (En blanco)

- Línea en blanco: Separa el comentario de la sección de las operaciones de cálculo de normales.

Línea 47: # Normal en cada punto de la esfera (vector unitario)

- Comentario: Explica que el objetivo es obtener vectores normales unitarios (longitud 1) para cada punto en la superficie.

Línea 48: normals = np.zeros_like(points)

- Crea un array NumPy llamado `normals` con la misma forma y tipo que el array `points` (es decir, `img_size` x `img_size` x 3), e inicializa todos sus elementos a cero. Este array almacenará los vectores normales.

Línea 49: # Calcular normales solo para los puntos dentro de la máscara

- Comentario: Indica que el cálculo de las normales solo se realizará para aquellos puntos que efectivamente pertenecen a la esfera, según la `mask` previamente definida.

Línea 50: normals[mask] = points[mask] / radius

- Calcula los vectores normales.

- `points[mask]`: Esta es una indexación booleana. Selecciona solo los elementos de `points` donde la `mask` es `True`. El resultado es un array 2D donde cada fila es un punto `[x, y, z]` en la superficie de la esfera. El número de filas es `mask.sum()`.

- `points[mask] / radius`: Para una esfera centrada en el origen, el vector normal en un punto de la superficie es simplemente el vector desde el centro (origen) hasta ese punto, normalizado. Dividir las coordenadas del punto `(x, y, z)` por el `radius` normaliza este vector, dándole longitud 1.

- `normals[mask] = ...`: Asigna estos vectores normales calculados a las posiciones correspondientes en el array `normals`. Los puntos fuera de la máscara en `normals` permanecerán como `[0, 0, 0]`.

Línea 51: (En blanco)

- Línea en blanco: Separa la sección de cálculo de normales de la definición de la función auxiliar.

Línea 52: # --- Función Auxiliar ---

- Comentario: Indica el inicio de una sección donde se define una función de utilidad.

Línea 53: (En blanco)

- Línea en blanco: Separa el comentario de la sección de la definición de la función.

Línea 54: # Función para normalizar vectores

- Comentario: Describe el propósito de la función que se definirá a continuación: normalizar un conjunto de vectores.

Línea 55: def normalize(v):

- Define una función llamada `normalize` que toma un argumento `v`. Se espera que `v` sea un array NumPy de vectores (por ejemplo, un array de NxD, donde N es el número de vectores y D su dimensionalidad).

Línea 56: norm = np.linalg.norm(v, axis=-1, keepdims=True)

- Dentro de la función `normalize`: Calcula la norma (magnitud o longitud) de cada vector en `v`.

- `np.linalg.norm(v, axis=-1, ...)`: Calcula la norma L2 a lo largo del último eje (`axis=-1`). Si `v` es un array de `N` vectores de 3 dimensiones (`Nx3`), esto calculará la norma de cada uno de los `N` vectores.

- `keepdims=True`: Asegura que el resultado `norm` mantenga la dimensionalidad del eje sobre el cual se calculó la norma, facilitando la división posterior (broadcasting). Si `v` es `Nx3`, `norm` será `Nx1`.

Línea 57: # Usar np.finfo para un epsilon pequeño y evitar división por cero

- Dentro de la función `normalize`: Comentario que explica la precaución tomada en la siguiente línea para evitar errores de división por cero si algún vector tiene norma cero.

Línea 58: norm = np.maximum(norm, np.finfo(float).eps)

- Dentro de la función `normalize`: Compara cada norma calculada con `np.finfo(float).eps`.

- `np.finfo(float).eps` es un valor muy pequeño (epsilon de máquina para números de punto flotante), la diferencia más pequeña representable entre 1.0 y el siguiente valor flotante mayor.

- `np.maximum(norm, ...)`: Reemplaza cualquier norma que sea menor que epsilon (incluyendo cero) con epsilon. Esto evita la división por cero o por un número extremadamente pequeño que podría llevar a valores infinitos o NaN (Not a Number).

Línea 59: return v / norm

- Dentro de la función `normalize`: Divide cada vector en `v` por su norma (ajustada). Debido al `keepdims=True` y las reglas de broadcasting de NumPy, cada componente de cada vector en `v` es dividido por la norma correspondiente de ese vector. El resultado es un array de vectores con la misma forma que `v`, pero donde cada vector tiene longitud 1 (o muy cercana a 1).

Línea 60: (En blanco)

- Línea en blanco: Separa la definición de la función auxiliar de la sección de cálculos de iluminación.

Línea 61: # --- Cálculos de Iluminación (Vectorizados donde sea posible) ---

- Comentario: Indica el inicio de la sección principal donde se calcularán los componentes de iluminación (ambiental, difusa, especular) para cada punto de la esfera. Destaca que se intentará usar operaciones vectorizadas de NumPy para eficiencia.

Línea 62: (En blanco)

- Línea en blanco: Separa el comentario de la sección de las primeras operaciones de cálculo de iluminación.

Línea 63: # Calcular vectores de luz (L) y vista (V) para cada punto de la esfera

- Comentario: Explica que se calcularán los vectores `L` (hacia la fuente de luz) y `V` (hacia el observador) para cada punto en la superficie de la esfera.

Línea 64: # Aplicar solo a los puntos válidos usando la máscara

- Comentario: Reitera que estos cálculos se restringirán a los puntos que pertenecen a la esfera, utilizando la `mask`.

Línea 65: (En blanco)

- Línea en blanco: Separa los comentarios de las operaciones.

Línea 66: # Vector de luz: desde el punto hasta la fuente de luz

- Comentario: Especifica cómo se define el vector de luz `L`.

Línea 67: L = light_pos - points[mask]

- Calcula los vectores de luz `L`.

- `points[mask]`: Selecciona los puntos 3D (`[x,y,z]`) que están en la superficie de la esfera. Es un array de forma `(num_mask_points, 3)`.

- `light_pos`: Es la posición de la fuente de luz (un array de forma `(3,)`).

- La resta se realiza mediante broadcasting: `light_pos` se resta de cada fila (punto) en `points[mask]`. El resultado `L` es un array de forma `(num_mask_points, 3)`, donde cada fila es el vector que va desde un punto en la superficie de la esfera hacia la fuente de luz.

Línea 68: L = normalize(L)

- Normaliza los vectores de luz `L` llamando a la función `normalize` definida anteriormente. Cada vector en `L` ahora tendrá longitud 1.

Línea 69: (En blanco)

- Línea en blanco: Separa el cálculo de L del cálculo de V.

Línea 70: # Vector de vista: desde el punto hasta la posición del observador

- Comentario: Especifica cómo se define el vector de vista `V`.

Línea 71: V = eye_pos - points[mask]

- Calcula los vectores de vista `V`.

- `eye_pos`: Es la posición del observador (un array de forma `(3,)`).

- Similar al cálculo de `L`, `eye_pos` se resta de cada punto en `points[mask]` (mediante broadcasting). El resultado `V` es un array de forma `(num_mask_points, 3)`, donde cada fila es el vector que va desde un punto en la superficie de la esfera hacia el observador.

Línea 72: V = normalize(V)

- Normaliza los vectores de vista `V` llamando a la función `normalize`. Cada vector en `V` ahora tendrá longitud 1.

Línea 73: (En blanco)

- Línea en blanco: Separa el cálculo de V del acceso a las normales N.

Línea 74: # Acceder a las normales solo para los puntos válidos

- Comentario: Indica que se van a extraer los vectores normales correspondientes a los puntos de la esfera.

Línea 75: N = normals[mask] # Ya están normalizadas

- Extrae los vectores normales `N` para los puntos de la esfera.

- `normals[mask]`: Utiliza la indexación booleana para seleccionar solo las normales de los puntos que están en la superficie de la esfera. `N` será un array de forma `(num_mask_points, 3)`.

- El comentario `# Ya están normalizadas` recuerda que los vectores en `normals` (para los puntos de la máscara) fueron calculados para tener longitud 1 en la línea 50.

Línea 76: (En blanco)

- Línea en blanco: Separa la obtención de N del cálculo del vector de reflexión R.

Línea 77: # Calcular la reflexión para el modelo Phong: R = 2*(N·L)*N - L

- Comentario: Introduce el cálculo del vector de reflexión `R`, que es un componente clave para el brillo especular en el modelo de iluminación de Phong. Muestra la fórmula matemática.

Línea 78: dot_NL = np.sum(N * L, axis=-1, keepdims=True)

- Calcula el producto punto (dot product) entre cada vector normal `N` y su correspondiente vector de luz `L`.

- `N * L`: Realiza una multiplicación elemento a elemento entre los arrays `N` y `L` (ambos de forma `(num_mask_points, 3)`). El resultado es un array de forma `(num_mask_points, 3)`.

- `np.sum(..., axis=-1, keepdims=True)`: Suma los resultados de la multiplicación a lo largo del último eje (el eje de las componentes x, y, z). Esto efectúa el producto punto `N·L` para cada par de vectores.

- `keepdims=True`: Mantiene la dimensionalidad, por lo que `dot_NL` tendrá forma `(num_mask_points, 1)`. Este valor representa `cos(theta)`, donde `theta` es el ángulo entre `N` y `L`.

Línea 79: # Clampear N·L >= 0 *antes* de calcular R para evitar reflejos extraños de luz "trasera"

- Comentario: Explica una importante consideración: el producto `N·L` debe ser no negativo para la reflexión. Si `N·L` es negativo, significa que la luz incide desde "detrás" de la superficie (relativo a la normal), lo cual no debería contribuir a la reflexión especular de la forma estándar.

Línea 80: dot_NL_clamped = np.maximum(0.0, dot_NL)

- Limita (clamping) los valores de `dot_NL` para que sean mayores o iguales a cero.

- `np.maximum(0.0, dot_NL)`: Para cada elemento en `dot_NL`, si es negativo, se reemplaza por `0.0`. Si es positivo o cero, se mantiene su valor. El resultado `dot_NL_clamped` tiene la misma forma que `dot_NL` (`(num_mask_points, 1)`).

Línea 81: R = 2 * dot_NL_clamped * N - L

- Calcula los vectores de reflexión `R` utilizando la fórmula de Phong.

- `2 * dot_NL_clamped * N`: `dot_NL_clamped` (forma `(num_mask_points, 1)`) se multiplica por cada componente de cada vector en `N` (forma `(num_mask_points, 3)`) mediante broadcasting, y luego se multiplica por 2.

- `... - L`: Se resta el vector de luz `L` (forma `(num_mask_points, 3)`) del resultado anterior.

- `R` es un array de forma `(num_mask_points, 3)` donde cada fila es el vector de reflexión para un punto en la esfera.

Línea 82: R = normalize(R)

- Normaliza los vectores de reflexión `R` para asegurar que tengan longitud 1, lo cual es importante para el cálculo del componente especular.

Línea 83: (En blanco)

- Línea en blanco: Separa el cálculo de R del cálculo de los componentes individuales de la iluminación.

Línea 84: # Calcular cada componente de la iluminación para los puntos en la máscara

- Comentario: Indica que las siguientes líneas calcularán las intensidades de los componentes ambiental, difuso y especular de la luz.

Línea 85: (En blanco)

- Línea en blanco: Separa el comentario de la sección del cálculo del componente ambiental.

Línea 86: # Componente ambiental

- Comentario: Identifica el cálculo del componente ambiental.

Línea 87: ambient = ambient_intensity

- Asigna el valor de `ambient_intensity` (definido previamente como 0.2) a la variable `ambient`. Esta es una intensidad escalar que se aplicará a todos los puntos visibles de la esfera.

Línea 88: (En blanco)

- Línea en blanco: Separa el componente ambiental del componente difuso.

Línea 89: # Componente difusa (según Lambert)

- Comentario: Identifica el cálculo del componente difuso, mencionando la ley de Lambert (la intensidad difusa es proporcional al coseno del ángulo entre la normal y la dirección de la luz).

Línea 90: # kd * max(0, N · L)

- Comentario: Muestra la fórmula matemática para el componente difuso: `kd` (coeficiente difuso) multiplicado por el máximo entre 0 y el producto punto de `N` y `L`.

Línea 91: # Usamos el dot_NL_clamped calculado antes

- Comentario: Recuerda que `max(0, N · L)` ya se calculó y almacenó en `dot_NL_clamped`.

Línea 92: diffuse_intensity = kd * dot_NL_clamped.squeeze() # .squeeze() para quitar la dimensión extra

- Calcula la intensidad de la luz difusa para cada punto.

- `dot_NL_clamped.squeeze()`: `dot_NL_clamped` tiene forma `(num_mask_points, 1)`. El método `.squeeze()` elimina las dimensiones de tamaño 1, por lo que el resultado es un array 1D de forma `(num_mask_points,)`.

- `kd * ...`: Multiplica el coeficiente difuso `kd` (0.7) por cada valor en el array `dot_NL_clamped` (aplastado).

- `diffuse_intensity` es un array 1D donde cada elemento es la intensidad difusa para un punto en la esfera.

Línea 93: diffuse = diffuse_intensity # Renombrado para claridad, ahora es escalar por punto

- Asigna `diffuse_intensity` a una nueva variable `diffuse`.

- El comentario `# Renombrado para claridad, ahora es escalar por punto` explica que, aunque el nombre anterior era descriptivo, `diffuse` es más conciso y aclara que representa una intensidad escalar para cada punto (no un vector de color todavía).

Línea 94: (En blanco)

- Línea en blanco: Separa el componente difuso del componente especular.

Línea 95: # Componente especular (modelo Phong)

- Comentario: Identifica el cálculo del componente especular, basado en el modelo de Phong.

Línea 96: # ks * max(0, R · V) ^ specular_exponent

- Comentario: Muestra la fórmula matemática para el componente especular: `ks` (coeficiente especular) multiplicado por el máximo entre 0 y el producto punto de `R` (vector de reflexión) y `V` (vector de vista), todo ello elevado al `specular_exponent`.

Línea 97: dot_RV = np.sum(R * V, axis=-1) # R·V por punto

- Calcula el producto punto entre cada vector de reflexión `R` y su correspondiente vector de vista `V`.

- `R * V`: Multiplicación elemento a elemento (ambos son `(num_mask_points, 3)`).

- `np.sum(..., axis=-1)`: Suma a lo largo del último eje para obtener el producto punto. El resultado `dot_RV` es un array 1D de forma `(num_mask_points,)`, donde cada elemento es `R·V` para un punto.

Línea 98: specular_intensity = ks * (np.maximum(0.0, dot_RV) ** specular_exponent)

- Calcula la intensidad de la luz especular para cada punto.

- `np.maximum(0.0, dot_RV)`: Limita los valores de `dot_RV` para que sean no negativos. Solo si el observador está "viendo" la reflexión de la luz (`R·V > 0`), habrá brillo especular.

- `... ** specular_exponent`: Eleva cada valor no negativo de `R·V` a la potencia `specular_exponent` (32). Esto hace que el brillo sea más concentrado alrededor del punto donde `R` y `V` están perfectamente alineados.

- `ks * ...`: Multiplica el resultado por el coeficiente especular `ks` (0.5).

- `specular_intensity` es un array 1D donde cada elemento es la intensidad especular para un punto en la esfera.

Línea 99: specular = specular_intensity # Renombrado para claridad, ahora es escalar por punto

- Asigna `specular_intensity` a una nueva variable `specular`.

- El comentario `# Renombrado para claridad, ahora es escalar por punto` sirve para el mismo propósito que en el caso de `diffuse`.

Línea 100: (En blanco)

- Línea en blanco: Separa los cálculos de intensidad de la combinación de componentes y asignación de colores.

Línea 101: # --- Combinar Componentes y Asignar Colores ---

- Comentario: Indica el inicio de la sección donde las intensidades calculadas (ambiental, difusa, especular) se combinarán con los colores del objeto y del brillo especular para determinar el color final de cada píxel.

Línea 102: (En blanco)

- Línea en blanco: Separa el comentario de la sección de la inicialización del array `shading`.

Línea 103: # Combinar componentes para obtener el color final por píxel

- Comentario: Explica el propósito de las siguientes líneas.

Línea 104: shading = np.zeros((mask.sum(), 3)) # Array para guardar el color calculado

- Crea un array NumPy 2D llamado `shading` e inicializa todos sus elementos a cero.

- `mask.sum()`: Calcula el número total de píxeles que pertenecen a la esfera (donde `mask` es `True`).

- La forma del array es `(num_mask_points, 3)`. Cada fila corresponderá a un punto en la esfera, y las 3 columnas almacenarán los componentes R, G, B del color calculado para ese punto.

- El comentario `# Array para guardar el color calculado` aclara su función.

Línea 105: (En blanco)

- Línea en blanco: Separa la inicialización de `shading` del bucle de canales de color.

Línea 106: # Para cada canal (R, G, B)

- Comentario: Indica que el siguiente bucle iterará sobre los tres canales de color.

Línea 107: for c in range(3):

- Inicia un bucle `for` que iterará tres veces, con la variable `c` tomando los valores 0, 1 y 2, correspondientes a los índices de los canales Rojo, Verde y Azul respectivamente.

Línea 108: # Color = Amb*ObjCol + Diff*ObjCol + Spec*SpecCol

- Dentro del bucle `for c`: Comentario que muestra la fórmula general para el modelo de iluminación de Phong para un canal de color:

  - `Amb*ObjCol`: Componente ambiental (intensidad ambiental * color del objeto para ese canal).

  - `Diff*ObjCol`: Componente difuso (intensidad difusa * color del objeto para ese canal).

  - `Spec*SpecCol`: Componente especular (intensidad especular * color del brillo especular para ese canal).

Línea 109: # Asegurarse que diffuse y specular tengan la forma correcta para broadcasting si es necesario

- Dentro del bucle `for c`: Comentario que sirve como recordatorio sobre la importancia de las formas de los arrays para que el broadcasting de NumPy funcione como se espera.

Línea 110: # (aunque con squeeze y axis=-1 deberían ser 1D arrays de tamaño mask.sum())

- Dentro del bucle `for c`: Comentario adicional que confirma que `diffuse` y `specular` ya son arrays 1D con `mask.sum()` elementos, lo que simplifica las operaciones.

Línea 111: shading[:, c] = ambient * object_color[c] + diffuse * object_color[c] + specular * specular_color[c]

- Dentro del bucle `for c`: Calcula el valor del canal de color `c` para todos los puntos de la esfera.

- `shading[:, c]`: Selecciona la columna `c` del array `shading` (es decir, el canal R, G o B para todos los puntos de la máscara).

- `ambient * object_color[c]`: El escalar `ambient` (intensidad ambiental) se multiplica por el componente `c` del `object_color` (ej. `object_color[0]` para el rojo). Esto es un escalar.

- `diffuse * object_color[c]`: El array 1D `diffuse` (intensidades difusas por punto) se multiplica (elemento a elemento) por el componente `c` del `object_color`. El resultado es un array 1D de valores difusos para el canal `c` en cada punto.

- `specular * specular_color[c]`: El array 1D `specular` (intensidades especulares por punto) se multiplica (elemento a elemento) por el componente `c` del `specular_color`. El resultado es un array 1D de valores especulares para el canal `c` en cada punto.

- Los tres términos se suman (aprovechando el broadcasting del término ambiental escalar con los arrays 1D). El resultado final, un array 1D, se asigna a la columna `c` de `shading`.

Línea 112: (En blanco)

- Línea en blanco: Separa el bucle de la asignación de colores a la imagen final.

Línea 113: # Solo asignar color a los píxeles que pertenecen a la esfera

- Comentario: Indica que los colores calculados en `shading` ahora se transferirán a la imagen principal, pero solo a los píxeles correspondientes a la esfera.

Línea 114: final_image = np.zeros_like(image) # Imagen final inicializada a negro

- Crea un nuevo array NumPy llamado `final_image` con la misma forma y tipo que `image` (la imagen original de 400x400x3 inicializada con ceros) y lo inicializa a ceros (negro). Esto es para asegurar que los píxeles fuera de la esfera permanezcan negros.

Línea 115: final_image[mask] = shading # Asignar los colores calculados a los píxeles de la máscara

- Asigna los colores calculados a la imagen final.

- `final_image[mask]`: Utiliza la indexación booleana con `mask` (que es 400x400). Esto selecciona las "ranuras" en `final_image` correspondientes a los píxeles donde la esfera es visible. Estas ranuras tienen la forma `(mask.sum(), 3)`.

- `shading`: Es el array de colores calculados, también de forma `(mask.sum(), 3)`.

- La asignación copia los valores de color RGB de `shading` a las ubicaciones apropiadas en `final_image`. Los píxeles donde `mask` es `False` no se modifican y permanecen negros.

- El comentario `# Asignar los colores calculados a los píxeles de la máscara` describe la operación.

Línea 116: (En blanco)

- Línea en blanco: Separa la asignación de colores del proceso de recorte (clipping).

Línea 117: # Asegurarse que los valores estén en el rango [0,1]

- Comentario: Indica que los valores de color resultantes deben ser limitados al rango válido para visualización (típicamente [0, 1] para flotantes o [0, 255] para enteros).

Línea 118: final_image = np.clip(final_image, 0, 1)

- Limita (recorta) todos los valores en `final_image` para que estén dentro del rango [0, 1].

- `np.clip(array, min_val, max_val)`: Para cada elemento en `array`, si es menor que `min_val`, se reemplaza por `min_val`. Si es mayor que `max_val`, se reemplaza por `max_val`. Si está dentro del rango, no cambia.

- Esto es importante porque la suma de los componentes de iluminación podría, en teoría, exceder 1.0, lo que Matplotlib podría interpretar incorrectamente al mostrar la imagen.

Línea 119: (En blanco)

- Línea en blanco: Separa la preparación de la imagen de la sección de visualización.

Línea 120: # --- Mostrar el resultado ---

- Comentario: Indica el inicio de la sección donde se utilizará Matplotlib para mostrar la imagen generada.

Línea 121: plt.figure(figsize=(6, 6))

- Crea una nueva figura de Matplotlib.

- `figsize=(6, 6)`: Especifica el tamaño de la figura en pulgadas (6 pulgadas de ancho por 6 pulgadas de alto).

Línea 122: plt.imshow(final_image)

- Muestra la imagen `final_image` en la figura actual. `plt.imshow` es la función estándar de Matplotlib para mostrar datos de array como una imagen. Espera que los valores de color estén en el rango [0,1] para flotantes RGB.

Línea 123: plt.axis('off')

- Desactiva los ejes (números, marcas y líneas de los ejes X e Y) en el gráfico, ya que no son relevantes para mostrar una imagen renderizada.

Línea 124: plt.title('Iluminación Combinada: Ambiental, Difusa y Especular (Azul)')

- Establece el título del gráfico como la cadena "Iluminación Combinada: Ambiental, Difusa y Especular (Azul)".

Línea 125: plt.show()

- Muestra la figura de Matplotlib con la imagen renderizada y el título. Esta función también inicia el bucle de eventos de la GUI de Matplotlib, si es necesario, y el script generalmente se pausará aquí hasta que se cierre la ventana del gráfico.

Resumen del Código

Este script de Python utiliza numpy para cálculos numéricos y matplotlib para la visualización, con el objetivo de renderizar una imagen 2D de una esfera 3D iluminada. El modelo de iluminación empleado es el de Phong, que combina tres componentes: ambiental, difuso y especular.

1. **Configuración Inicial:**
    - Se definen parámetros clave: el tamaño de la imagen de salida (img_size), el radio de la esfera (radius), y su centro (en el origen).
    - Se establecen los parámetros de iluminación: intensidad de la luz ambiental (ambient_intensity), coeficientes de reflexión difusa (kd) y especular (ks) del material, y el exponente especular (specular_exponent) que controla el brillo.
    - Se especifican las posiciones 3D de la fuente de luz (light_pos) y del observador/cámara (eye_pos).
    - Se define el color base del objeto (la esfera) como azul (object_color) y el color del brillo especular como blanco (specular_color).
2. **Preparación de la Geometría:**
    - Se crea una cuadrícula 2D de coordenadas (X, Y) que representan los píxeles de la imagen en el espacio del objeto.
    - Para cada punto (X, Y) en esta cuadrícula, se calcula la coordenada Z correspondiente en la superficie de la esfera (asumiendo que se mira la mitad frontal, Z >= 0). Los puntos fuera de la proyección de la esfera obtienen Z=0.
    - Se genera una mask booleana que identifica qué píxeles (X, Y) caen dentro de la proyección circular de la esfera.
    - Se combinan X, Y, y Z en un array points que contiene las coordenadas 3D de cada punto en la cuadrícula.

3. **Cálculo de Normales:**
    - Para cada punto en la superficie de la esfera (identificado por la mask), se calcula el vector normal. Dado que la esfera está centrada en el origen, la normal en un punto es simplemente el vector desde el origen a ese punto, dividido por el radio para normalizarlo (hacerlo unitario).
4. **Cálculos de Iluminación (Vectorizados):**
    - Se define una función auxiliar normalize para convertir vectores a longitud unitaria, manejando posibles divisiones por cero.
    - Para cada punto en la superficie de la esfera (usando la mask):
        - **Vector de Luz (L):** Se calcula el vector desde el punto hacia la fuente de luz y se normaliza.
        - **Vector de Vista (V):** Se calcula el vector desde el punto hacia el observador y se normaliza.
        - **Vector de Reflexión (R):** Se calcula el vector de reflexión de la luz L sobre la superficie (dada por la normal N) usando la fórmula R = 2*(N·L)*N - L, y se normaliza. Se asegura que N·L sea no negativo.
        - **Componente Ambiental:** Es una intensidad constante (ambient_intensity).
        - **Componente Difuso:** Se calcula como kd * max(0, N·L). Esta intensidad depende del ángulo entre la normal y la luz.
        - **Componente Especular:** Se calcula como ks * max(0, R·V) ^ specular_exponent. Esta intensidad depende del ángulo entre el vector de reflexión y el vector de vista, creando el "brillo".
5. **Combinación de Componentes y Asignación de Color:**
    - Se itera sobre los tres canales de color (Rojo, Verde, Azul).
    - Para cada canal y cada punto en la esfera, el color final se calcula sumando las contribuciones de los tres componentes de iluminación: Color_canal = (ambiental * objeto_color_canal) + (difuso * objeto_color_canal) + (especular * especular_color_canal)
    - Los colores calculados se almacenan en un array shading.

- Estos colores se asignan a los píxeles correspondientes en una imagen final (final_image), utilizando la mask para asegurar que solo se coloreen los píxeles de la esfera.
- Los valores de color en final_image se recortan (clipping) al rango [0, 1] para asegurar que sean válidos para la visualización.

6. **Visualización:**
    - Se utiliza matplotlib.pyplot para mostrar la final_image resultante.
    - Se desactiva la visualización de los ejes y se añade un título descriptivo a la imagen.

El resultado es una imagen de una esfera azul con sombreado suave (difuso), un brillo especular blanco, y una iluminación ambiental general, simulando cómo se vería bajo una fuente de luz específica desde una perspectiva particular.

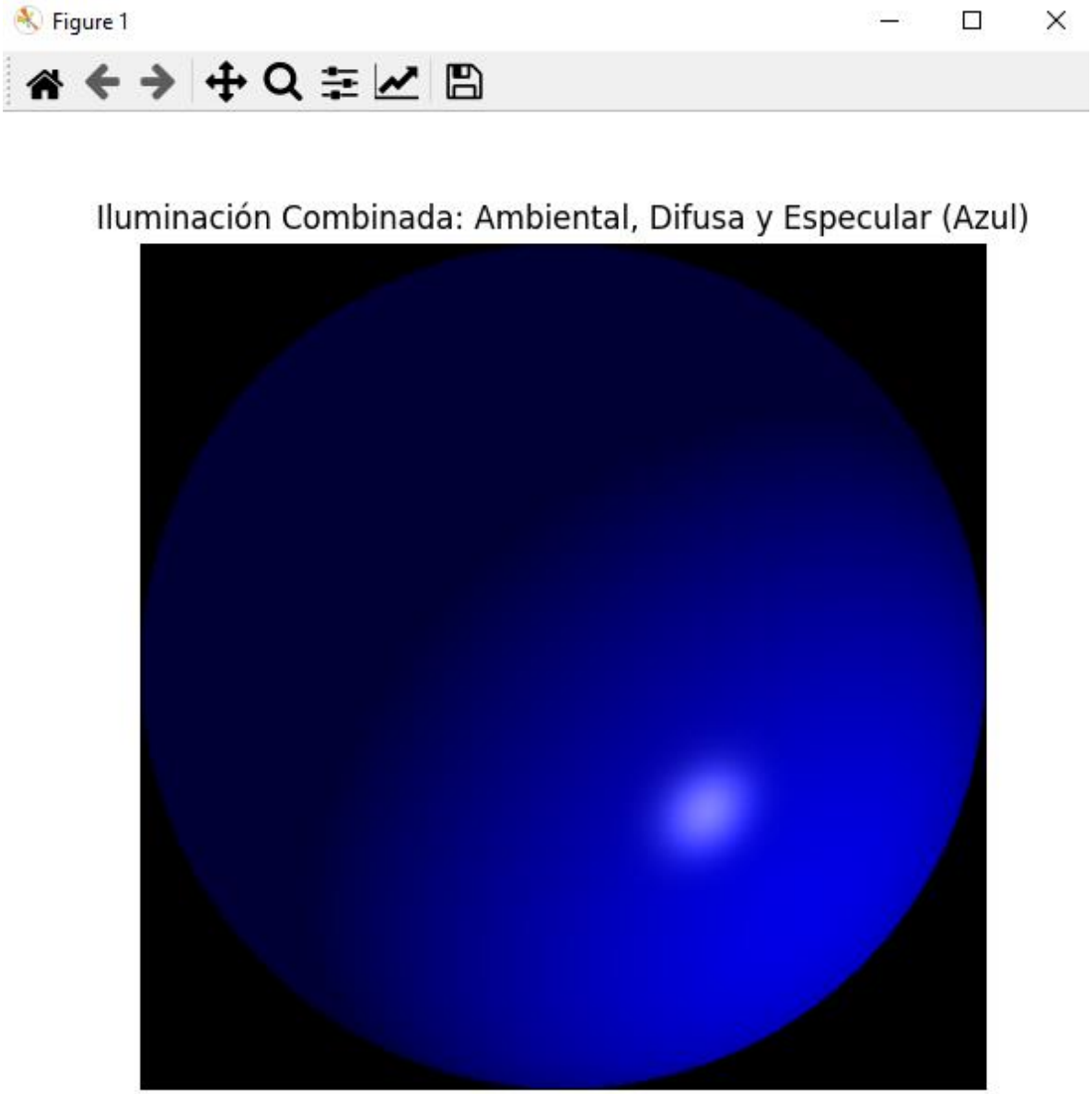
Figure 1
Iluminación Combinada: Ambiental, Difusa y Especular (Azul)