UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

CARRERA PROFESIONAL DE INGENIERIA INFORMATICA Y DE SISTEMAS

COMPUTACION GRAFICA 2

DOCENTE: M.SC. HECTOR E. UGARTE R.

HOJA DE EJERCICIOS 6: ILUMINACION, SOMBRAS Y REFLEXIONES

1. COMPETENCIAS

- El estudiante conoce el concepto de cómo se iluminan objetos 3D.
- El estudiante entiende cómo aplicar reflexiones y sombras a objetos 3D.

2. MARCO TEORICO

Iluminación

La iluminación de gráficos por computadora es la colección de técnicas utilizadas para simular la luz en escenas de gráficos por computadora. Mientras que las técnicas de iluminación ofrecen flexibilidad en el nivel de detalle y funcionalidad disponible, también operan en diferentes niveles de complejidad y demanda computacional.

El efecto general de una fuente de luz sobre un objeto está determinado por la combinación de las interacciones del objeto con él, generalmente descritas por al menos tres componentes principales. Los tres componentes principales de iluminación (y los tipos de interacción subsiguientes) son difusos, ambientales y especulares.

Difuso

La iluminación difusa (o reflexión difusa) es la iluminación directa de un objeto por una cantidad uniforme de luz que interactúa con una superficie que dispersa la luz. Después de que la luz incide en un objeto, se refleja en función de las propiedades de la superficie del objeto, así como del ángulo de la luz entrante. Esta interacción es el factor principal que contribuye al brillo del objeto y constituye la base de su color.

Ambiente

Como la luz ambiental no tiene dirección, interactúa de manera uniforme en todas las superficies, con su intensidad determinada por la fuerza de las fuentes de luz ambiental y las propiedades de los materiales de la superficie de los objetos, es decir, sus coeficientes de reflexión ambiental.

Especular

El componente de iluminación especular da brillo y realce a los objetos. Esto es distinto de los efectos de espejo porque otros objetos en el entorno no son visibles en estos reflejos. En cambio, la iluminación especular crea puntos brillantes en los objetos según la intensidad del componente de iluminación especular y el coeficiente de reflexión especular de la superficie.

El **modelo Blinn-Phong** es una aproximación al modelo Phong que es menos intensivo computacionalmente. Entonces todo objeto de la escena debe tener estas 4 propiedades (ambiente, difuso, especular, brillo). el modelo Blinn-Phong calcula la iluminación de un punto de la siguiente manera:

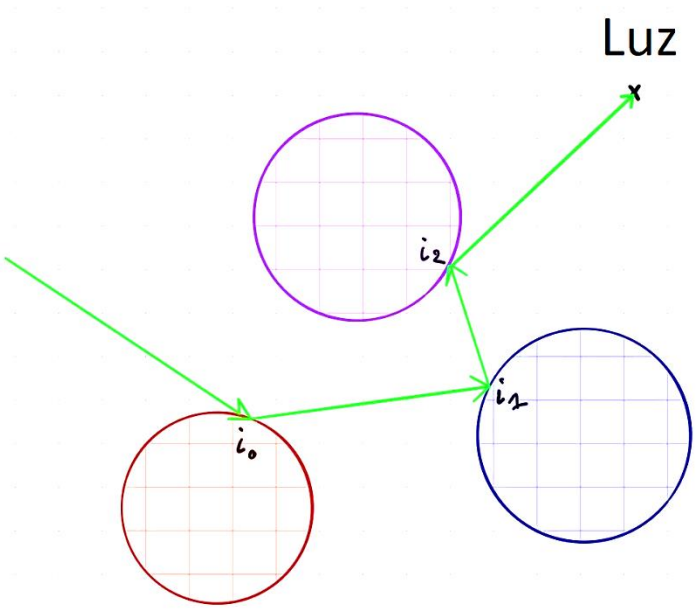
$$I_p = k_a * i_a + k_d * i_d * L \cdot N + k_s * i_s * (N \cdot \frac{L+V}{\|L+V\|})^{\frac{\alpha}{4}}$$

Donde:

- ka, kd, ks son las propiedades ambientales, difusas y especulares del objeto;
- ia, id, is son las propiedades ambientales, difusas y especulares de la luz;
- L es un vector unitario de dirección desde el punto de intersección hacia la luz;
- N es el vector unitario normal a la superficie del objeto en el punto de intersección;
- V es un vector unitario de dirección desde el punto de intersección hacia la cámara;
- α es el brillo del objeto;

¿Qué pasa si el rayo golpea múltiples objetos antes de golpear la cámara?
Esto es reflexión. El rayo acumulará diferentes colores y cuando incida en la cámara verás reflejos. Cada objeto tiene un coeficiente de reflexión en el rango de 0 a 1. "0" significa que el objeto es mate, "1" significa que el objeto es como un espejo. Actualmente, calculamos un rayo que comienza en la cámara y se dirige hacia un píxel, luego rastreamos ese rayo en la escena, buscamos la intersección más cercana y calculamos el color del punto de intersección.

Para incluir reflejos, debemos rastrear el rayo reflejado después de que ocurra una intersección e incluir la contribución de color de cada punto de intersección. Repetimos ese proceso un cierto número de veces.



Cálculo de color

Para obtener el color de un píxel, debemos sumar la contribución de cada punto intersectado por el rayo.

$$c_p = i_0 + r_0 i_1 + r_0 r_1 i_2 + r_0 r_1 r_2 i_3 + \dots$$

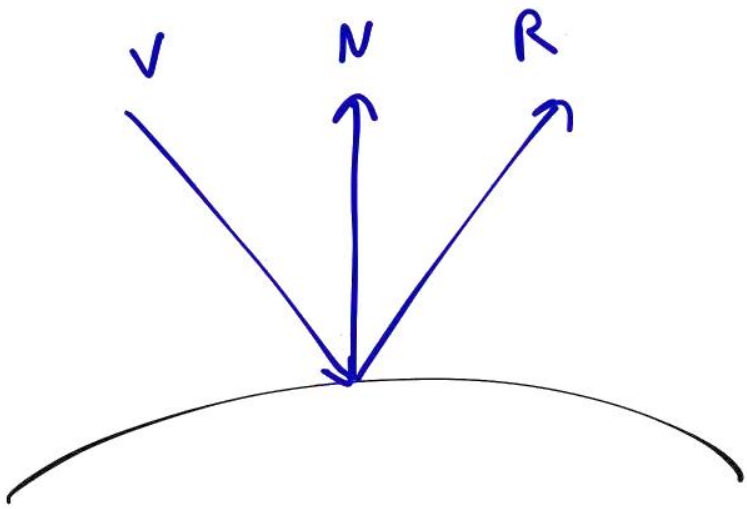
dónde,

- c es el color (final) de un píxel;
- i es la iluminación calculada por el modelo Blinn-Phong del número de puntos de intersección índice;
- r es el reflejo del número de objetos de índice cruzados;

Rayo reflejado

Podemos calcular un rayo reflejado de la siguiente manera:

$$R = V - 2(V \cdot N)N$$



Donde:

- R es el rayo reflejado normalizado;
- V es un vector unitario de dirección del rayo a reflejar;
- N es el vector unitario de dirección normal a la superficie del trazo del rayo;

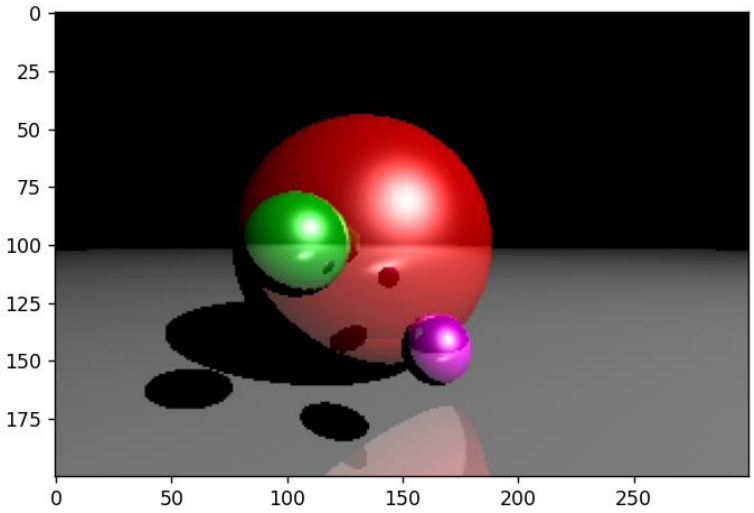
3. PRACTICA

A. El siguiente código extiende el ejemplo visto en la practica inicial de raytracing. Ahora las esferas muestran sombras y reflexiones. Entiende, prueba y comenta el código.

```
import numpy as np
import matplotlib.pyplot as plt

def normalizar(vector):
    return vector / np.linalg.norm(vector)

def reflejado(vector, eje):
    return vector - 2 * np.dot(vector, eje) * eje

def interseccion_esfera(centro, radio, origen_rayo, direccion_rayo):
    b = 2 * np.dot(direccion_rayo, origen_rayo - centro)
    c = np.linalg.norm(origen_rayo - centro) ** 2 - radio ** 2
    delta = b ** 2 - 4 * c
    if delta > 0:
        t1 = (-b + np.sqrt(delta)) / 2
        t2 = (-b - np.sqrt(delta)) / 2
        if t1 > 0 and t2 > 0:
            return min(t1, t2)
    return None

def objeto_intersectado_mas_cercano(objetos, origen_rayo, direccion_rayo):
    distancias = [interseccion_esfera(obj['centro'], obj['radio'], origen_rayo,
direccion_rayo) for obj in objetos]
    objeto_mas_cercano = None
    distancia_minima = np.inf
    for indice, distancia in enumerate(distancias):
        if distancia and distancia < distancia_minima:
            distancia_minima = distancia
            objeto_mas_cercano = objetos[indice]
    return objeto_mas_cercano, distancia_minima

ancho = 300
altura = 200

profundidad_maxima = 3

camara = np.array([0, 0, 1])
ratio = float(ancho) / altura
pantalla = (-1, 1 / ratio, 1, -1 / ratio) # izquiera, arriba, derecha, abajo

luz = { 'posicion': np.array([5, 5, 5]), 'ambiente': np.array([1, 1, 1]), 'difuso':
np.array([1, 1, 1]), 'especular': np.array([1, 1, 1]) }

objetos = [
    { 'centro': np.array([-0.2, 0, -1]), 'radio': 0.7, 'ambiente': np.array([0.1, 0, 0]),
'difuso': np.array([0.7, 0, 0]), 'especular': np.array([1, 1, 1]), 'brillo': 100,
'reflexion': 0.5 },
    { 'centro': np.array([0.1, -0.3, 0]), 'radio': 0.1, 'ambiente': np.array([0.1, 0,
0.1]), 'difuso': np.array([0.7, 0, 0.7]), 'especular': np.array([1, 1, 1]), 'brillo': 100,
'reflexion': 0.5 },
    { 'centro': np.array([-0.3, 0, 0]), 'radio': 0.15, 'ambiente': np.array([0, 0.1, 0]),
'difuso': np.array([0, 0.6, 0]), 'especular': np.array([1, 1, 1]), 'brillo': 100,
'reflexion': 0.5 },
    { 'centro': np.array([0, -9000, 0]), 'radio': 9000 - 0.7, 'ambiente': np.array([0.1,
0.1, 0.1]), 'difuso': np.array([0.6, 0.6, 0.6]), 'especular': np.array([1, 1, 1]),
'brillo': 100, 'reflexion': 0.5 }
]

imagen = np.zeros((altura, ancho, 3))
for i, y in enumerate(np.linspace(pantalla[1], pantalla[3], altura)):
    for j, x in enumerate(np.linspace(pantalla[0], pantalla[2], ancho)):
        # pantalla esta en el origen
        pixel = np.array([x, y, 0])
        origen = camara
```

```
        direccion = normalizar(pixel - origen)

        color = np.zeros((3))
        reflexion = 1

        for k in range(profundidad_maxima):
            # verifica por intersecciones
            objeto_mas_cercano, distancia_minima =
objeto_intersectado_mas_cercano(objetos, origen, direccion)
            if objeto_mas_cercano is None:
                break

            interseccion = origen + distancia_minima * direccion
            normal_a_superficie = normalizar(interseccion - objeto_mas_cercano['centro'])
            punto_desplazado = interseccion + 1e-5 * normal_a_superficie
            interseccion_con_luz = normalizar(luz['posicion'] - punto_desplazado)

            _, distancia_minima = objeto_intersectado_mas_cercano(objetos,
punto_desplazado, interseccion_con_luz)
            interseccion_con_luz_distancia = np.linalg.norm(luz['posicion'] -
interseccion)
            esta_sombreado = distancia_minima < interseccion_con_luz_distancia

            if esta_sombreado:
                break

            iluminacion = np.zeros((3))

            # ambiente
            iluminacion += objeto_mas_cercano['ambiente'] * luz['ambiente']

            # difuso
            iluminacion += objeto_mas_cercano['difuso'] * luz['difuso'] *
np.dot(interseccion_con_luz, normal_a_superficie)

            # especular
            interseccion_a_camara = normalizar(camara - interseccion)
            H = normalizar(interseccion_con_luz + interseccion_a_camara)
            iluminacion += objeto_mas_cercano['especular'] * luz['especular'] *
np.dot(normal_a_superficie, H) ** (objeto_mas_cercano['brillo'] / 4)

            # reflexion
            color += reflexion * iluminacion
            reflexion *= objeto_mas_cercano['reflexion']

            origen = punto_desplazado
            direccion = reflejado(direccion, normal_a_superficie)

        imagen[i, j] = np.clip(color, 0, 1)
    print("%d/%d" % (i + 1, altura))

plt.imsave('imagen.png', imagen)
imgplot = plt.imshow(imagen)
plt.show()
```

Salida obtenida:



B. Utilizando el código anterior como ejemplo, construye lo mas similar a ambas imágenes siguientes: