P1. ***Engadir a este motor a posibilidade de traballar cun número arbitrario de fontes de luz, cada unha delas coas súas propias características (posición, cor). Modificade a escena predeterminada a renderizar para que inclúa novas luces con distintas cores, mostrando a nova feature incorporada.***

La solución a este apartado se ha dividido en dos partes:

1) Crear una lista de iluminaciones (posición, color), por lo cual si se desea incorporar otro número arbitrario diferente de iluminaciones basta con incorporar/remover de la lista:

```
lights = [{'position': np.array([5., 5., -10.]), 'color': np.ones(3)},
    {'position': np.array([-3., 5., -15.]), 'color': np.array([0.5, 0.5, 0.5])},
    {'position': np.array([0., 10., -5.]), 'color': np.array([0.2, 0.2, 0.8])}]
```

Los colores corresponden al blanco, gris, y azul. Las posiciones son modificaciones con respecto a la original .

2) Modificar la función de **trace_ray** encargada de trazar los rayos en la escena.
Esta función se divide en dos partes, en primer lugar encuentra todas las colisiones con los objetos dentro de la escena para poder calcular las sombras y en segundo lugar, construye los colores. Debido a esto, se ha realizado la modificación de la función según las siguientes partes:

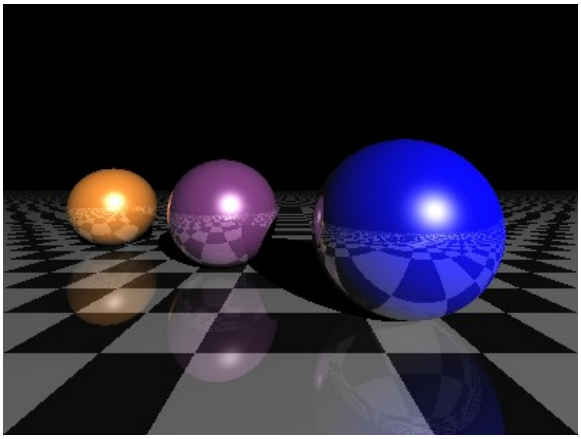
```
def trace_ray(rayO, rayD):
 # Parte 1
 # Find first point of intersection with the scene.
 t = np.inf
 for i, obj in enumerate(scene):
 t_obj = intersect(rayO, rayD, obj)
 if t_obj < t:
 t, obj_idx = t_obj, i
 # Return None if the ray does not intersect any object.
 if t == np.inf:
 return
 # Find the object.
 obj = scene[obj_idx]
 # Find the point of intersection on the object.
 M = rayO + rayD * t
 # Find properties of the object.
 N = get_normal(obj, M)
 color = get_color(obj, M)

 # Start computing the color.
 col_ray = ambient
```

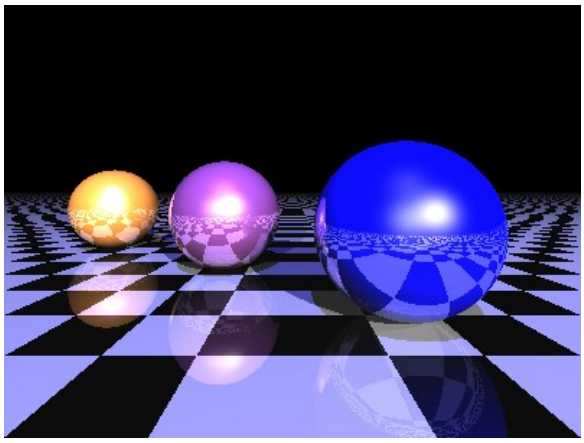
En la **#Parte 1** se encuentra la primera intersección del rayo con los diferentes objetos que están en la escena y se almacena la información (objeto, punto intersección, normal y color) para utilizar en el trazado de la iluminación en #Parte 2.

```
# Parte 2
for light in lights:
    toL = normalize(light['position'] - M)
    toO = normalize(O - M)
    # Shadow: find if the point is shadowed or not.
    l = [intersect(M + N * .0001, toL, obj_sh)
        for k, obj_sh in enumerate(scene) if k != obj_idx]
    if l and min(l) < np.inf:
        continue
    light_color = light['color']
    # Lambert shading (diffuse).
    col_ray += obj.get('diffuse_c', diffuse_c) * max(np.dot(N, toL), 0) * color * light_color
    # Blinn-Phong shading (specular).
    col_ray += obj.get('specular_c', specular_c) * max(np.dot(N, normalize(toL + toO)), 0) **
specular_k * light_color
return obj, M, N, col_ray
```

En la **#Parte 2** se toma cada iluminación declarada y para cada una de ella se calcula las sombras que genera y se aplica el color, de la misma forma que en el código original.

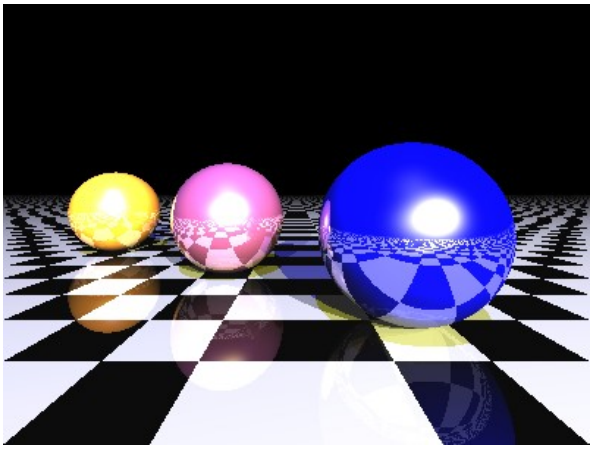


Base



P1-1

Como se puede observar en **P1-1** se han procesado 3 iluminaciones (blanca, gris, azulada), calculando sus respectivas contribuciones a las sombras de forma individual.
Adicionalmente se ha creado un ejemplo donde se le incorpora una luz amarilla adicional:



P2-2

Las variables de las luces serían (para **P2-2**):

```
lights = [
    {'position': np.array([5., 5., -10.]), 'color': np.ones(3)},
    {'position': np.array([-3., 5., -15.]), 'color': np.array([0.5, 0.5, 0.5])},
    {'position': np.array([0., 10., -5.]), 'color': np.array([0.2, 0.2, 0.8])},
    {'position': np.array([2.5, 4., -9.]), 'color': np.array([1.0, 1.0, 0.0])}
]
```