

Mestrado em  
Engenharia Informática

# The 3D Rendering Problem

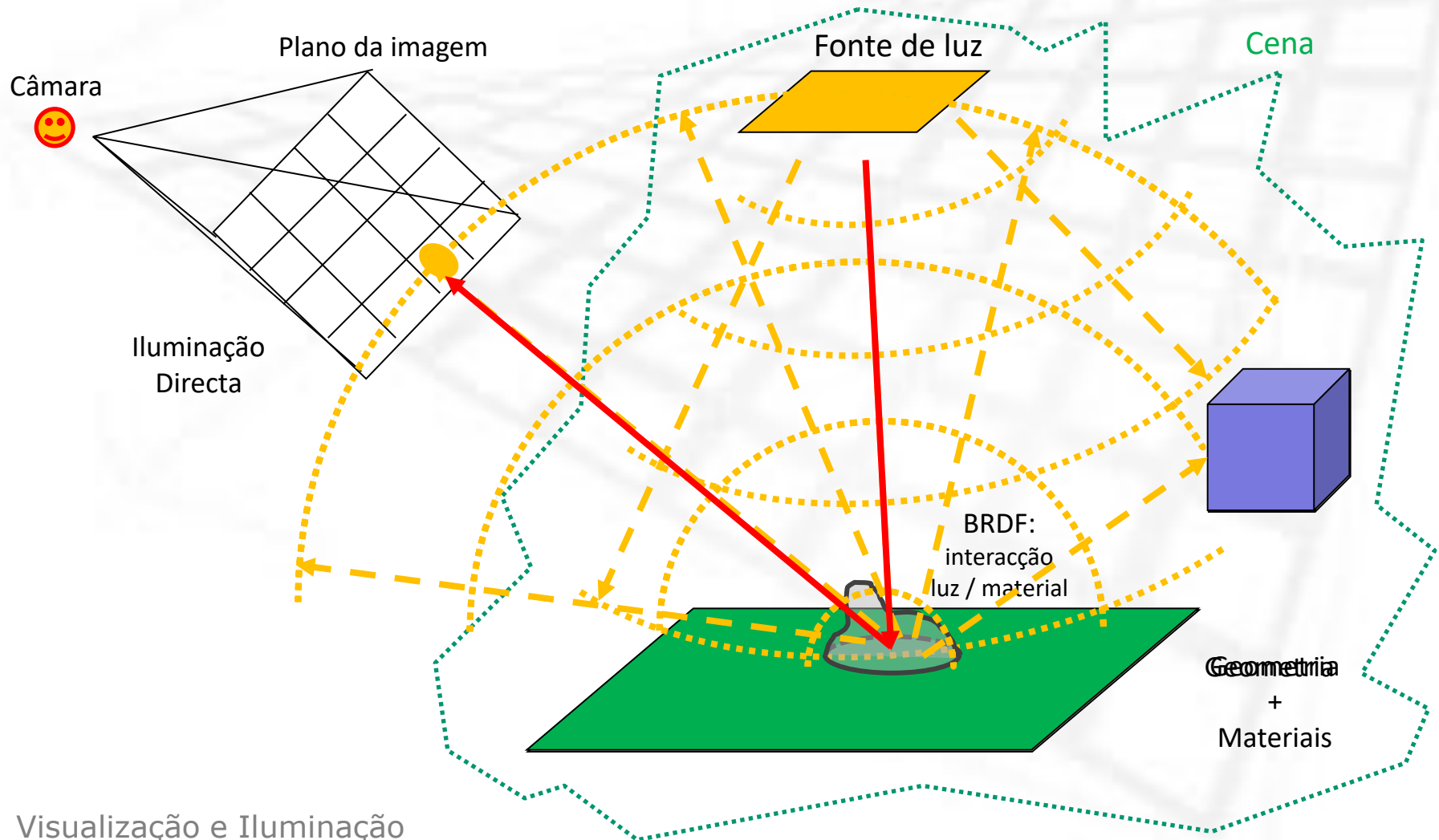
Visualização e Iluminação

Luís Paulo Peixoto dos Santos

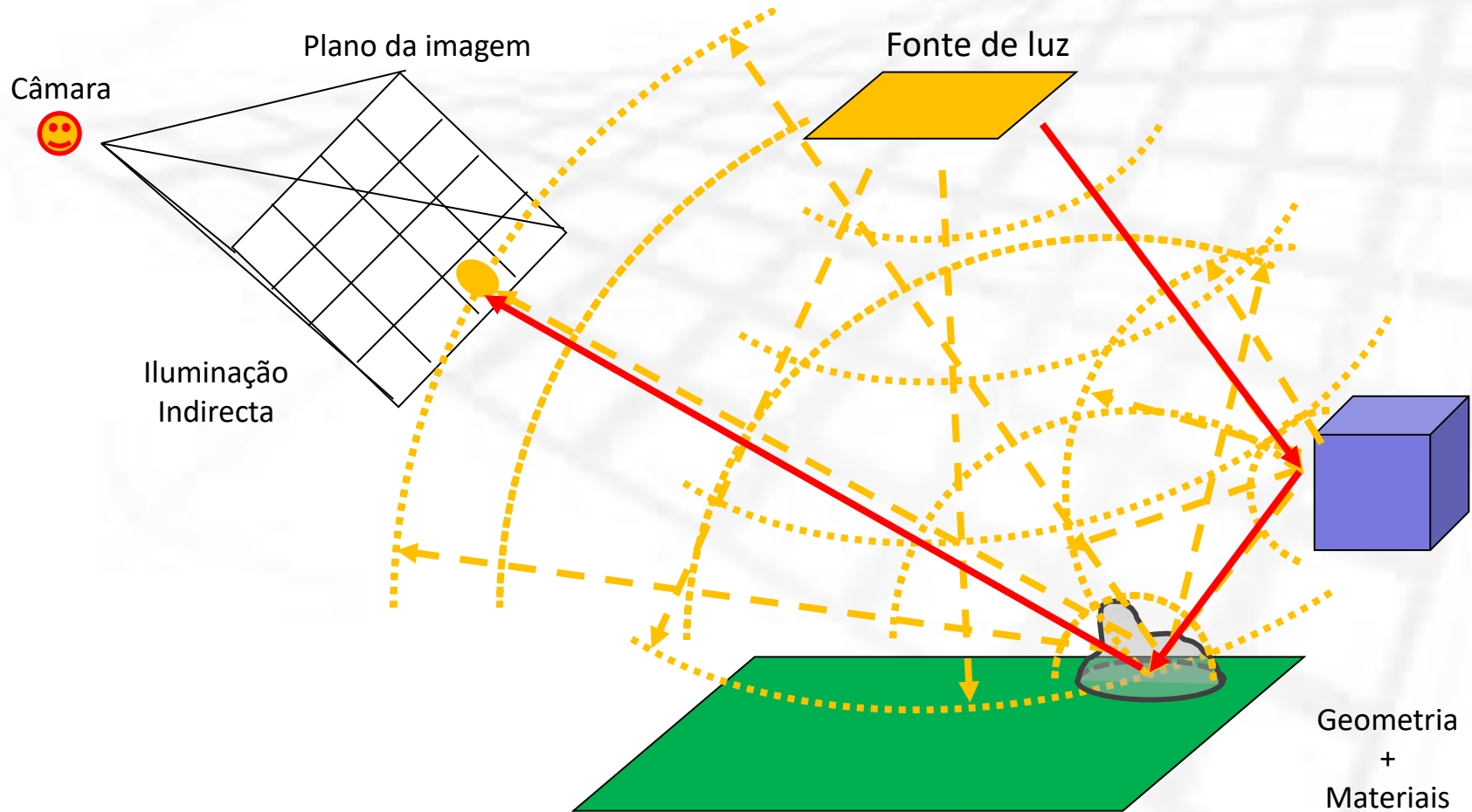
# Global Illumination



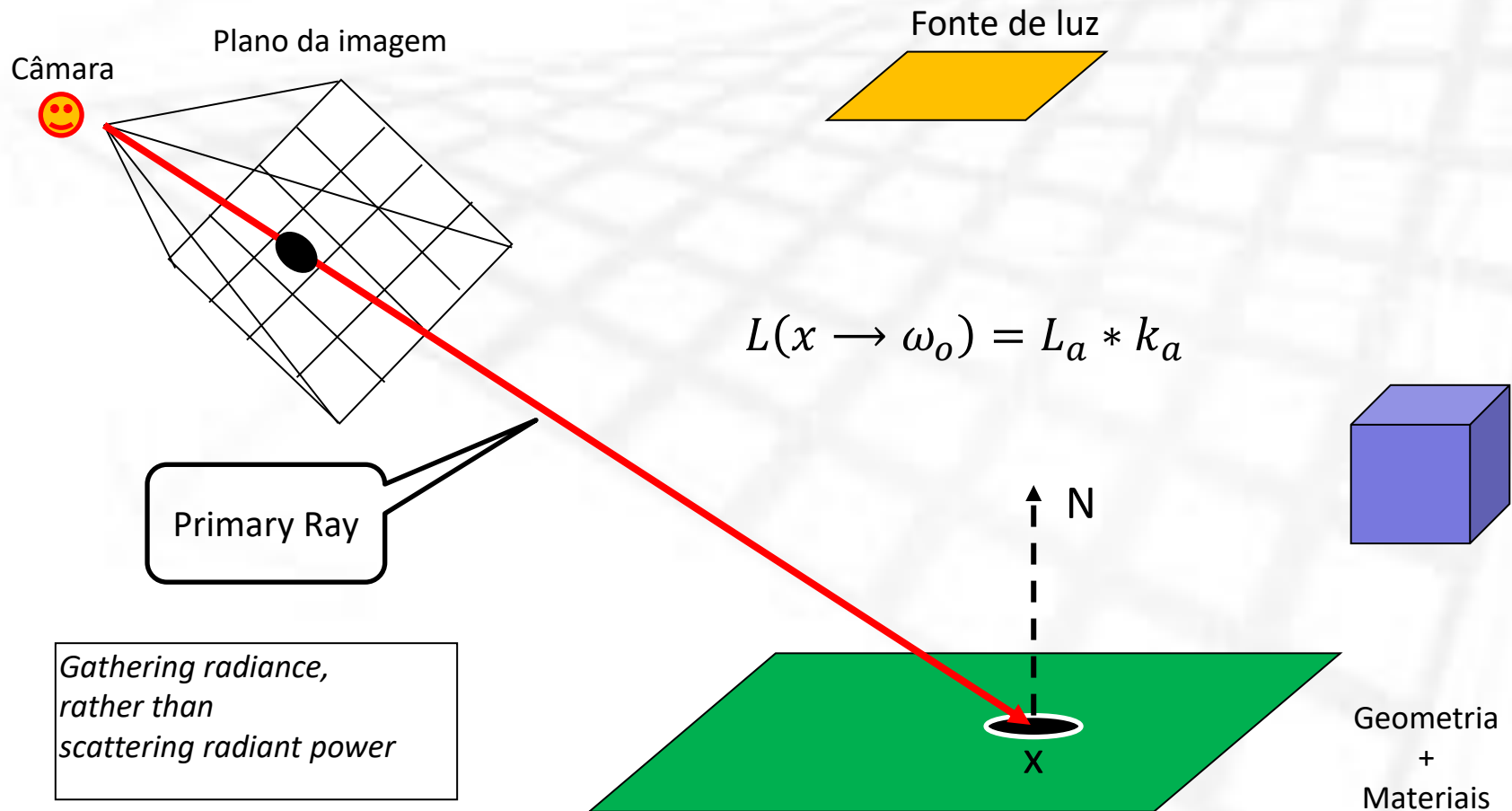
# *The 3D Rendering Problem*



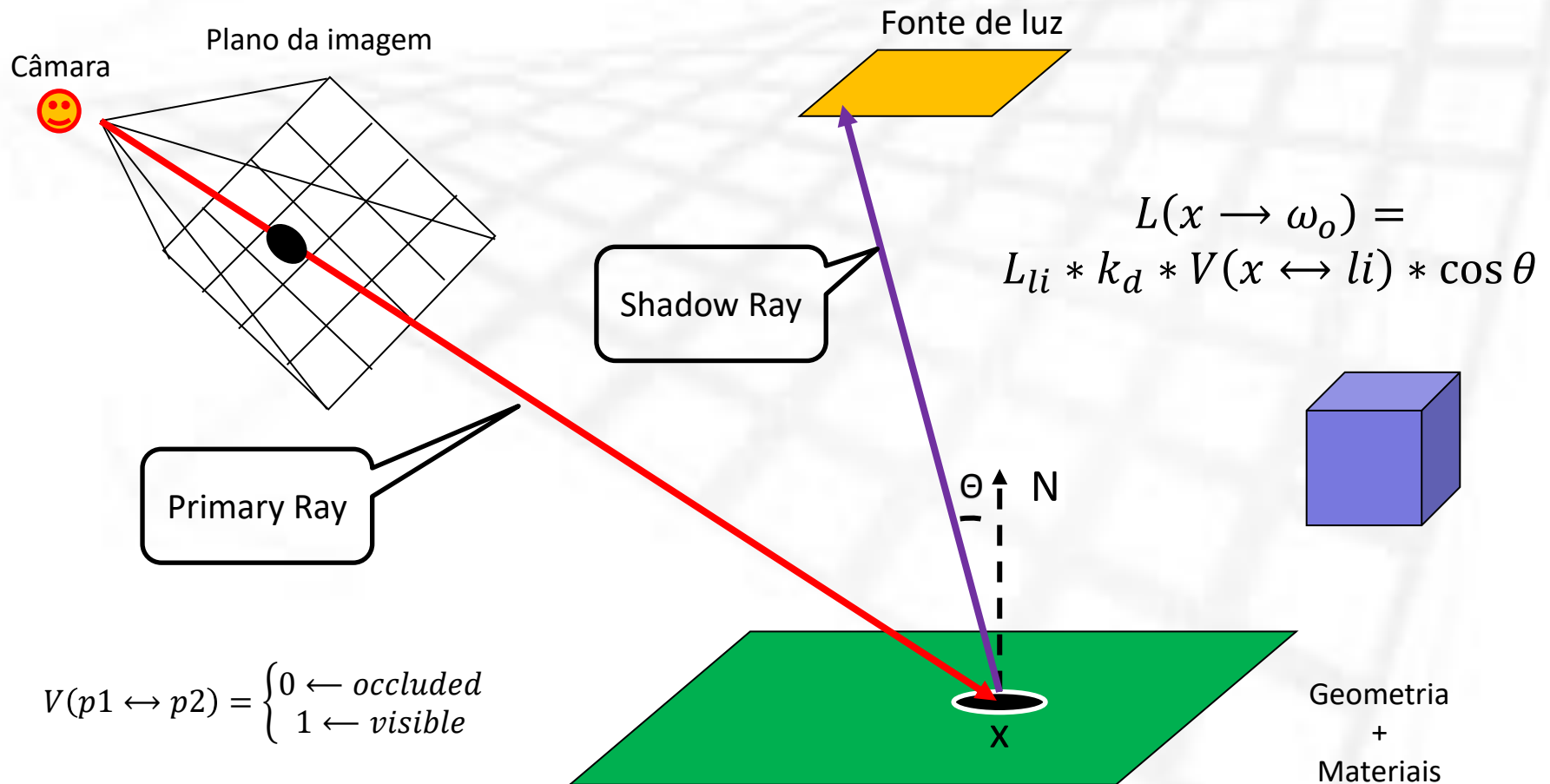
# *The 3D Rendering Problem*



# Backward Ray Tracing

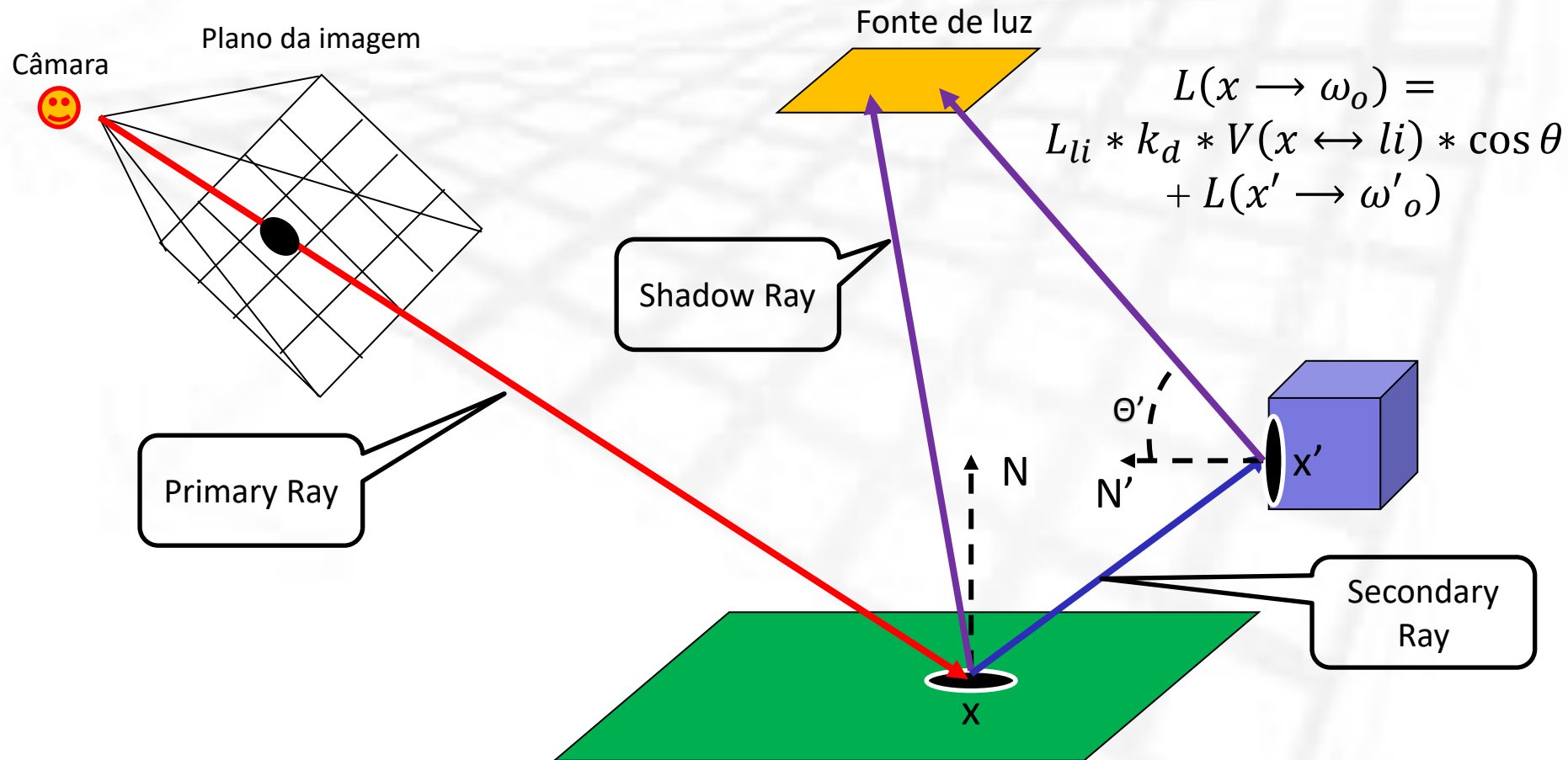


# Backward Ray Tracing





# Backward Ray Tracing



```
// main loop
computeImage (viewPoint) {
    para cada ponto p in plano_imagem {
        ray = camera.GetRay (p)
        radiance[p] = rad (ray)
    }
}

rad (ray) {
    primitive, x = scene.trace (ray)
    shade (x, ray, primitive)
}
```

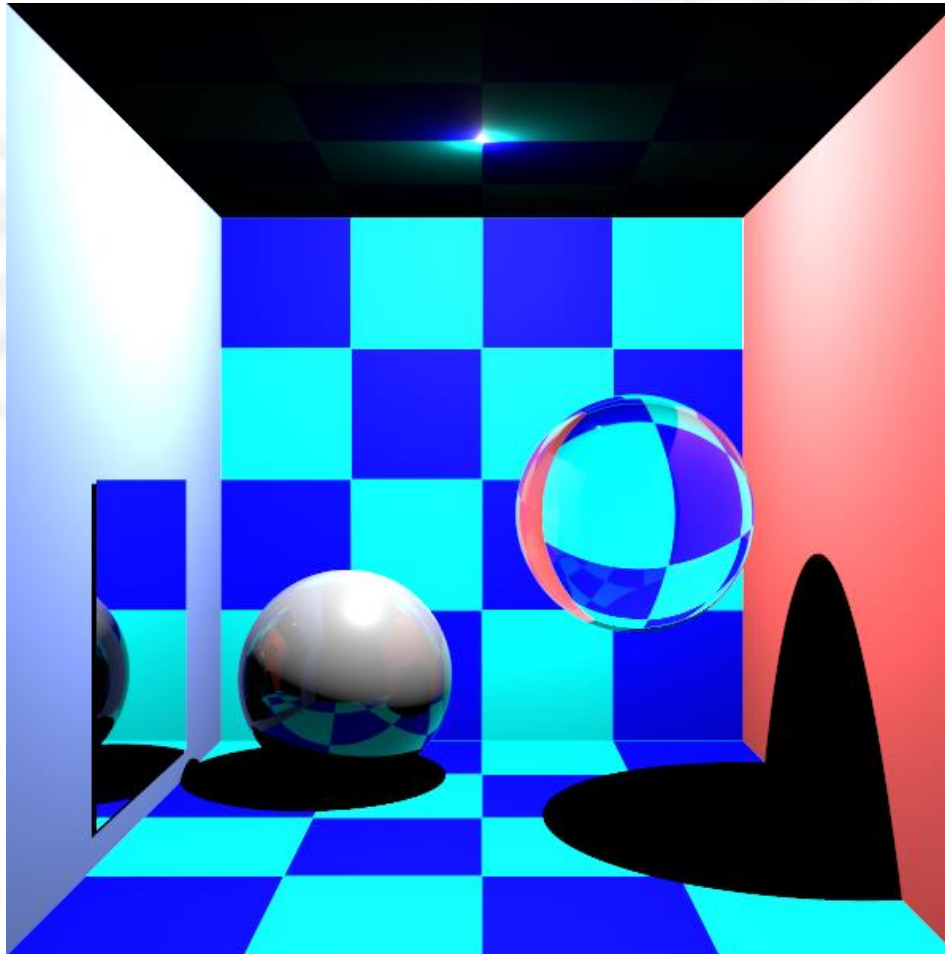


## *Ray Tracing: Algoritmo*

// intersecção mais próxima da origem do raio

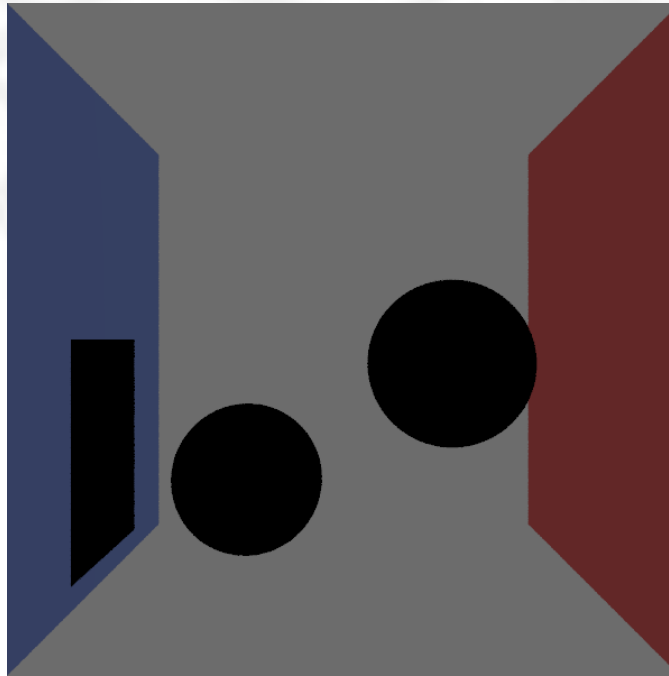
```
Scene::trace (ray)  {  
    tmin = Max_dist  
    while (primitive = scene.nextPrim())  {  
        x = primitive.geom.intersect (ray)  
        dist = x.distance (ray.origin)  
        if (dist < tmin)  {  
            tmin = dist  
            p = x  
            prim = primitive  
        }  
    }  
    return (prim, p)  
}
```

## *Ray Tracing: Cornell Box*



## *shading*: diffuse BRDF

```
shade (x, ray, prim){ // BRDF
    return (prim.BRDF(x, ray.dir))
}
```

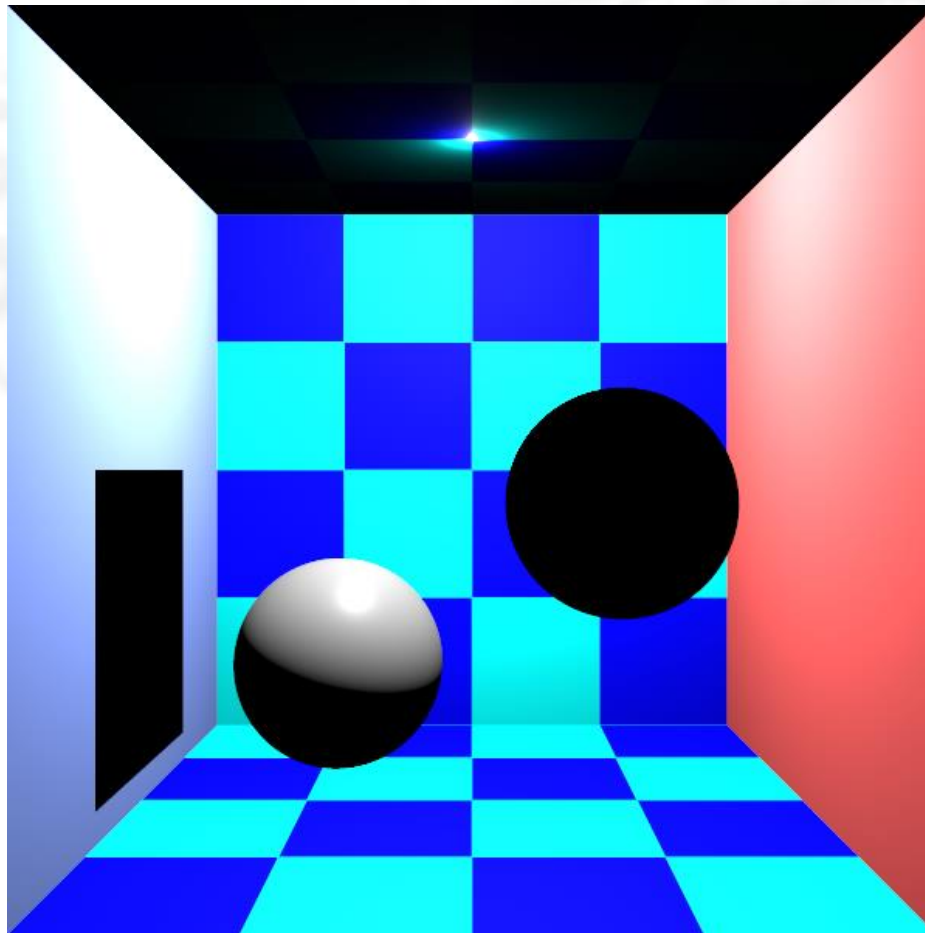


## Ray Tracing: Iluminação Directa

- Se os *shadow rays* não forem disparados assume-se  $V(x,y)=1$
- O algoritmo não calcula sombras (**NÃO** é fisicamente plausível)

```
shade (x, ray, prim) {  
    radiance = directIllum_NoShadows (x, ray.dir, prim)  
    return (radiance)  
}  
  
directIllum_NoShadows (x, dir, prim) {  
    rad = 0;  
    while (l = scene.nextLight()) {  
        pl = l.Sample(); dir_l = x.vec2Point (pl);  
        rad += prim.brdf (dir, dir_l) * l.L * cos (x.N, dir_l)  
    }  
    return (rad)  
}
```

## *Ray Tracing: Iluminação Directa*



## *Ray Tracing*: Iluminação Directa

```
shade(x, ray, prim) {
    radiance = directIllum (x, ray.dir, prim)
    return (radiance)
}

directIllum (x, dir, prim) {
    rad = 0;
    while (l = scene.nextLight()) {
        pl = l.Sample(); dir_l = x.vec2Point (pl);
        ray = GenerateRay (x, l, SHADOW)
        if (scene.visibility (ray, pl))
            rad += prim.brdf (dir, dir_l)* l.L * cos (Nx, dir_l)
    }
    return (rad)
}
```

```
// visibilidade da fonte de luz
```

```
Scene::visibility (ray, pl) {      // V(x,y)
    tmin = distance (ray.origin, pl)
    while (primitive = scene.nextPrim()) {
        x = primitive.geom.intersect (ray)
        dist = x.distance (ray.origin)
        if (dist < tmin) {
            return 0
        }
    }
    return 1
}
```



## *Ray Tracing: Iluminação Directa*

