# Ray Tracing

Gráficas por Computadora (CC2018) - 2026

# Ray Tracing

Semestre 02, 2026

## Objetivo

- Comprender qué es un raytracer y en qué se diferencia de las técnicas vistas hasta ahora.
- Entender el modelo físico de la luz que el raytracer imita y por qué se invierte.
- Conocer el flujo general de un raytracer: rayo primario, intersección, sombreado, color.
- Formalizar el rayo en 3D con su origen, su dirección y su ecuación paramétrica.
- Comparar raycasting y ray tracing, y ubicar sus casos de uso.

## Definición

El ray tracing es una técnica de renderizado que genera una imagen siguiendo el recorrido de rayos de luz a través de una escena tridimensional.

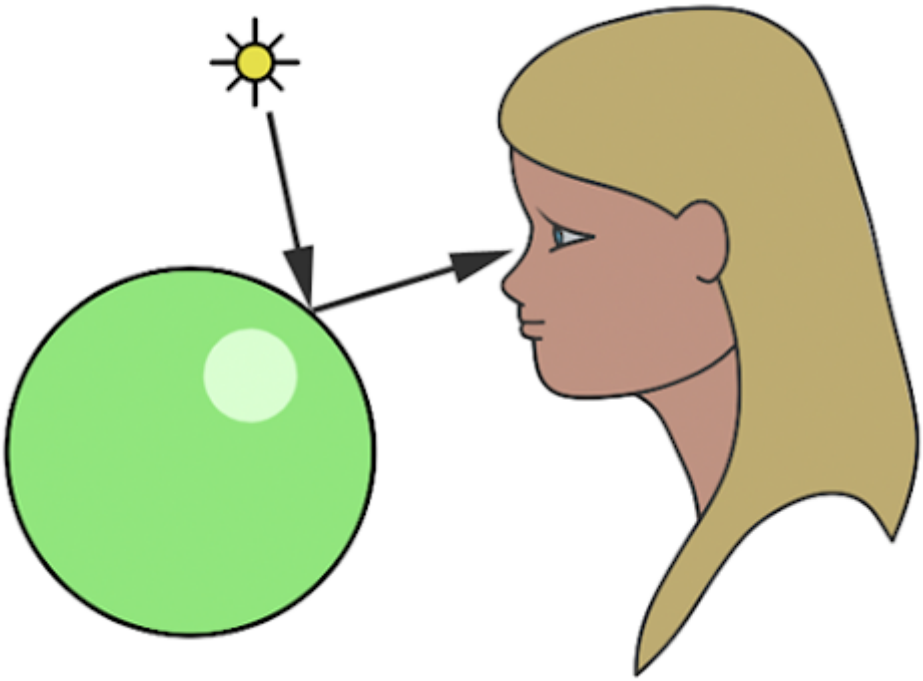
Un raytracer es el programa que implementa esa técnica. Su tarea es simple de enunciar: para cada píxel de la pantalla, lanza un rayo hacia la escena, averigua

qué objeto golpea primero y calcula de qué color se ve ese punto.

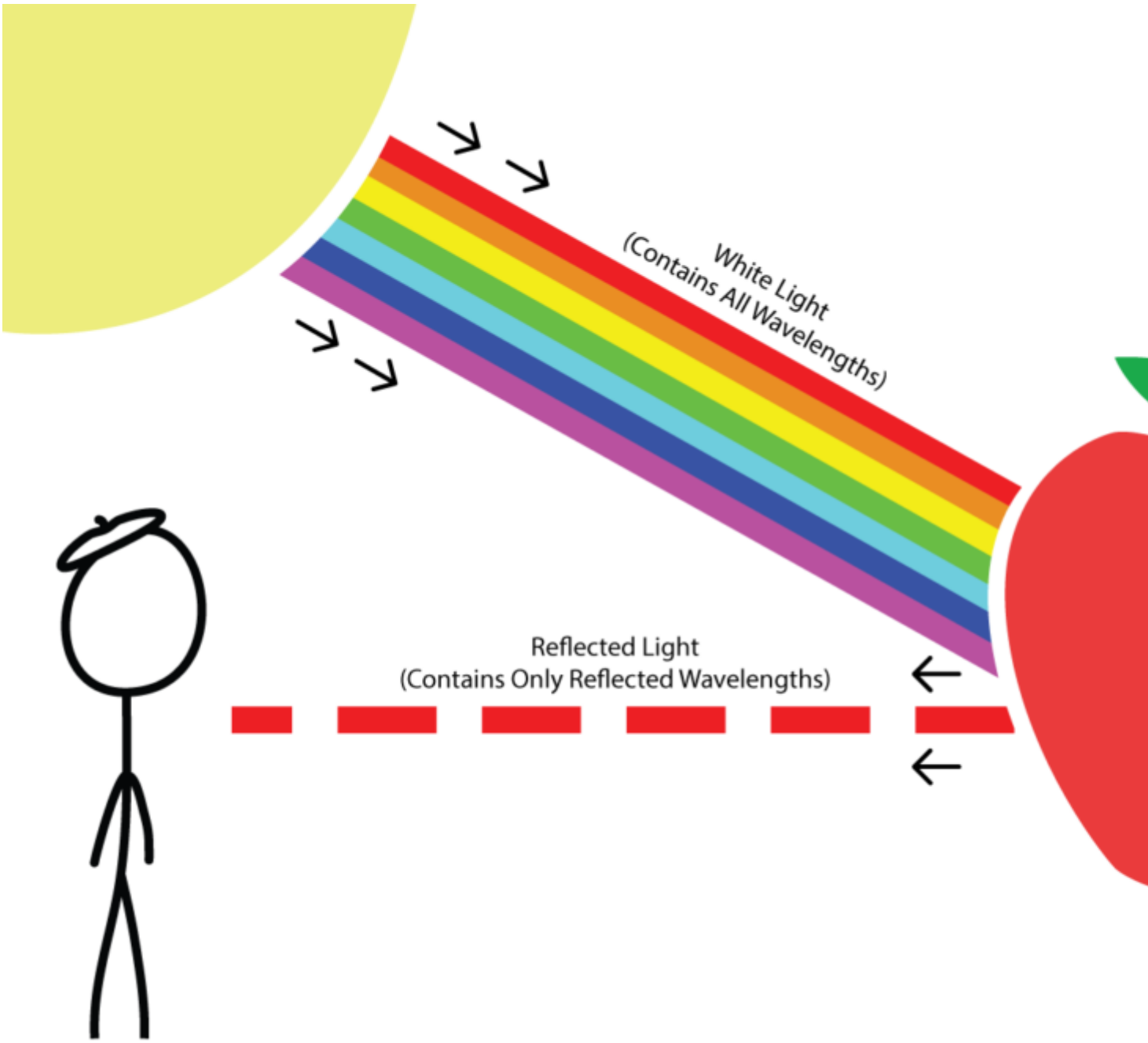
A diferencia del raycasting del maze, aquí no hay una cuadrícula 2D ni columnas de pantalla: hay geometría tridimensional real y un rayo por cada píxel de la imagen.

## Cómo vemos el mundo

Todo lo que vemos llega a nuestros ojos como luz. Una fuente emite fotones, esos fotones viajan, rebotan en las superficies y algunos terminan entrando al ojo.

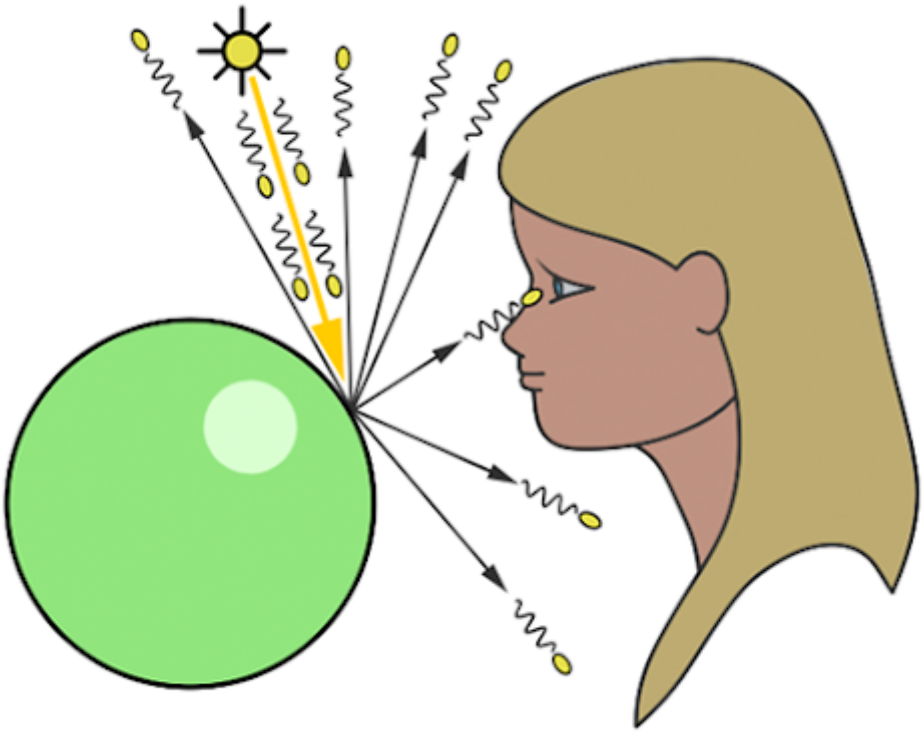


El color de un objeto no es una propiedad mágica: es el resultado de qué longitudes de onda absorbe la superficie y cuáles refleja. Una manzana se ve roja porque absorbe casi todo el espectro y refleja el rojo.




## El problema de simular la luz hacia adelante

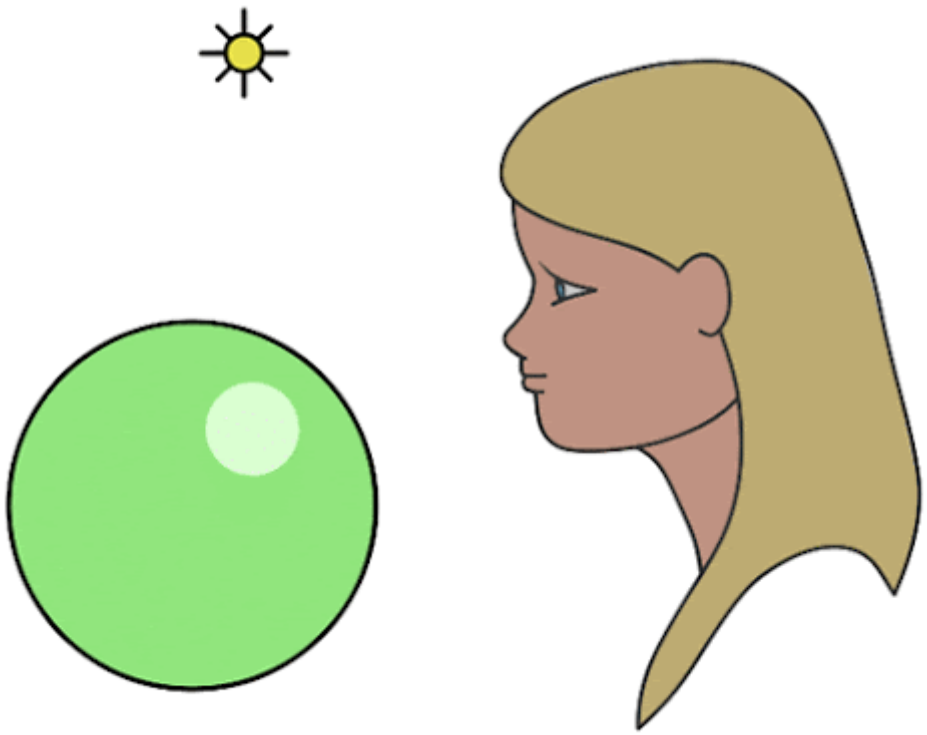
La simulación honesta sería la forward ray tracing: emitir fotones desde la fuente de luz, seguirlos mientras rebotan por la escena y quedarse con los que lleguen a la cámara.



El problema es de eficiencia. Una fuente de luz emite en todas las direcciones, y de todos esos fotones solo una fracción minúscula termina entrando a la cámara. Estaríamos gastando casi todo el cómputo en luz que nunca vamos a ver.

## Backward ray tracing

La solución es invertir el recorrido: en lugar de salir de la luz, los rayos salen de la cámara.



Se lanza un rayo por píxel hacia la escena. Cuando ese rayo golpea una superficie, se pregunta cuánta luz llega a ese punto. Así, todo rayo que trazamos contribuye al resultado: ninguno se desperdicia.

Este es el enfoque que usa prácticamente todo raytracer real, y es el que vamos a implementar. Cuando se habla de ray tracing sin más, se está hablando de backward ray tracing.

## Funcionamiento general

El raytracer recorre la imagen píxel por píxel. Para cada uno repite el mismo procedimiento:

1. Generar el rayo primario que sale de la cámara y atraviesa ese píxel.
2. Probar el rayo contra todos los objetos de la escena y quedarse con la intersección más cercana.
3. Si no golpea nada, pintar el color de fondo o del cielo.

4. Si golpea algo, calcular la normal y el material en el punto de impacto.
5. Sombrear: evaluar cuánta luz llega a ese punto y con qué intensidad.
6. Escribir el color resultante en el framebuffer.

El resultado de recorrer los píxeles es la imagen completa. No hay proyección de vértices ni relleno de triángulos: la imagen aparece porque cada píxel preguntó, por su cuenta, qué se ve a través de él.

## El rayo en 3D

Un rayo es una semirrecta: tiene un punto de partida y una dirección, y se extiende infinitamente hacia un solo lado. Se define con dos elementos.

- Origen $O$: el punto del espacio de donde sale el rayo. En un rayo primario es la posición de la cámara.
- Dirección $D$: un vector que indica hacia dónde apunta. Se guarda normalizado, es decir con $\lVert D \rVert = 1$.

Cualquier punto sobre el rayo se obtiene avanzando una cantidad $t$ desde el origen en la dirección $D$. Esa es la ecuación paramétrica del rayo:

$$ P(t) = O + t,D $$

El parámetro $t$ es la distancia recorrida a lo largo del rayo, y su signo tiene significado geométrico.

- $t = 0$ es exactamente el origen del rayo.
- $t > 0$ son los puntos que están frente al rayo, los únicos que nos interesan.
- $t < 0$ son puntos detrás del origen: existen sobre la recta, pero no sobre el rayo.

Cuando la dirección está normalizada, $t$ se lee directamente como distancia en unidades del mundo. Si $D$ no estuviera normalizado, $t$ quedaría escalado por la magnitud de $D$ y las comparaciones de distancia dejarían de ser confiables.

### Por qué la ecuación paramétrica importa

Todo el trabajo geométrico del raytracer se reduce a resolver esta ecuación contra la superficie de un objeto. Para intersectar un rayo con una esfera, con un plano o con un triángulo, se sustituye $P(t)$ en la ecuación de esa superficie y se despeja $t$.

- Si no hay solución real, el rayo no toca ese objeto.
- Si hay soluciones, la menor $t$ positiva es el primer punto de impacto.
- Comparando la $t$ de todos los objetos, la más pequeña gana: ese es el objeto visible.

De ahí sale, gratis, el manejo de la oclusión. No hace falta un z-buffer ni ordenar la escena: el objeto más cercano es simplemente el de menor $t$.

## Rayos primarios

Los rayos primarios son los que salen de la cámara, uno por píxel. Construirlos consiste en convertir las coordenadas de un píxel en una dirección dentro del espacio de la escena.

El proceso general es el siguiente:

1. Pasar el píxel a coordenadas normalizadas, con el centro de la pantalla en el origen.
2. Corregir por la relación de aspecto para que la imagen no salga estirada cuando el ancho y el alto difieren.

3. Escalar por el campo de visión, que abre o cierra la apertura de la cámara.
4. Formar el vector dirección con esas componentes y normalizarlo.
5. Emitir el rayo con origen en la cámara y esa dirección.

## El espacio normalizado

Las pantallas usan coordenadas de píxel, con el origen en una esquina y la fila cero arriba. Si se trabajara directamente en ese espacio, los cálculos de intersección quedarían atados al tamaño exacto de la ventana.

- Al mapear los píxeles al rango $[-1, 1]$ en ambos ejes se obtiene un sistema independiente de la resolución.
- Se puede cambiar el tamaño de la ventana sin tocar la lógica del trazado de rayos.
- El eje vertical se invierte, porque en el framebuffer la fila cero es la de arriba mientras que en el espacio de la cámara ese eje crece hacia arriba.

## La relación de aspecto

La mayoría de las pantallas no son cuadradas, así que el eje horizontal debe escalarse para que la escena se proyecte con la geometría correcta.

$$ ratio = \frac{ancho}{alto} $$

- Sin esta corrección una esfera se vería como una elipse estirada a lo ancho o a lo alto.
- El factor multiplica la coordenada horizontal antes de armar la dirección del rayo.

## Longitud unitaria

Una dirección se representa con un vector. Si ese vector tuviera una longitud arbitraria, su magnitud distorsionaría el resultado, así que se normaliza dividiéndolo entre su propia longitud.

$$ \vec{d}_{norm} = \frac{\vec{d}}{\lVert \vec{d} \rVert} $$

Con todas las direcciones de longitud uno, el parámetro $t$ se lee directamente como distancia y las comparaciones entre objetos son confiables.

## Objetos 3D

La escena no es una imagen: es una lista de objetos. Cada objeto se describe con dos cosas.

- Geometría: la forma que ocupa en el espacio. Es lo que permite responder si un rayo lo toca y a qué distancia.
- Material: cómo responde a la luz al ser golpeado. Es lo que decide el color final del punto de impacto.

El raytracer nunca dibuja un objeto. Lo único que le pregunta es si un rayo lo toca y dónde.

### Primitivas geométricas

Se usan formas que tienen una ecuación cerrada, porque contra ellas la intersección se resuelve con álgebra y no por aproximación.

- Esfera: definida por un centro y un radio. Es la primitiva más barata de intersectar.
- Plano: definido por un punto y una normal. Es infinito, y sirve para pisos, paredes y techos.

- Triángulo: definido por tres vértices. Es la pieza con la que se arma cualquier modelo 3D.

Una escena compleja no usa formas complejas: usa muchísimas primitivas simples.

## La esfera

Una esfera de centro $C$ y radio $r$ es el conjunto de puntos que están exactamente a distancia $r$ del centro:

$$ \lVert P - C \rVert^2 = r^2 $$

Se sustituye el punto por la ecuación del rayo, $P(t) = O + tD$, y queda todo en función de $t$:

$$ \lVert O + tD - C \rVert^2 = r^2 $$

Al desarrollar el producto punto, la expresión se reordena como una ecuación cuadrática en $t$:

$$ at^2 + bt + c = 0 \quad\text{con}\quad a = D\cdot D,; b = 2D\cdot(O - C),; c = \lVert O - C \rVert^2 - r^2 $$

El discriminante $\Delta = b^2 - 4ac$ dice, por sí solo, la relación geométrica entre el rayo y la esfera.

- $\Delta < 0$: no hay solución real, el rayo pasa de largo sin tocar la esfera.
- $\Delta = 0$: una sola solución, el rayo roza la esfera de forma tangente.
- $\Delta > 0$: dos soluciones, el punto por donde el rayo entra y el punto por donde sale.

De las dos raíces se toma la menor que sea positiva: es la cara de la esfera que mira hacia la cámara.

## El plano

Un plano se define con un punto $P_0$ que le pertenece y una normal $\vec{n}$ perpendicular a él. Todo punto del plano cumple:

$$ \vec{n}\cdot(P - P_0) = 0 $$

Sustituyendo el rayo y despejando, $t$ sale directo, sin cuadrática:

$$ t = \frac{\vec{n}\cdot(P_0 - O)}{\vec{n}\cdot D} $$

- Si el denominador es cero, el rayo es paralelo al plano y nunca lo corta.
- Si la $t$ resultante no es positiva, el plano queda detrás de la cámara y se descarta.

## El triángulo

La prueba del triángulo se hace en dos tiempos, apoyándose en la del plano.

1. Cortar el plano que lo contiene. Los tres vértices definen un plano, y contra ese plano se calcula la $t$ del rayo.
2. Verificar que el punto caiga dentro. El corte con el plano puede quedar fuera del triángulo, así que se comprueba que el punto esté del lado interno de los tres bordes.

Los modelos 3D no se guardan como fórmulas sino como mallas de triángulos, así que esta es la intersección que más veces se ejecuta en un render real.

## La normal en el punto de impacto

Saber dónde pegó el rayo no alcanza: hace falta saber hacia dónde mira la superficie en ese punto.

- Esfera: la normal cambia en cada punto y apunta hacia afuera desde el centro, $\vec{n} = \frac{P - C}{r}$.
- Plano y triángulo: la normal es constante en toda la superficie, la misma sirve para cualquier punto de impacto.

La normal es la que decide cuánta luz recibe el punto y hacia dónde sale un rayo reflejado.

### El impacto más cercano

Con varios objetos en la escena, el rayo se prueba contra todos y se conserva únicamente el impacto de menor distancia.

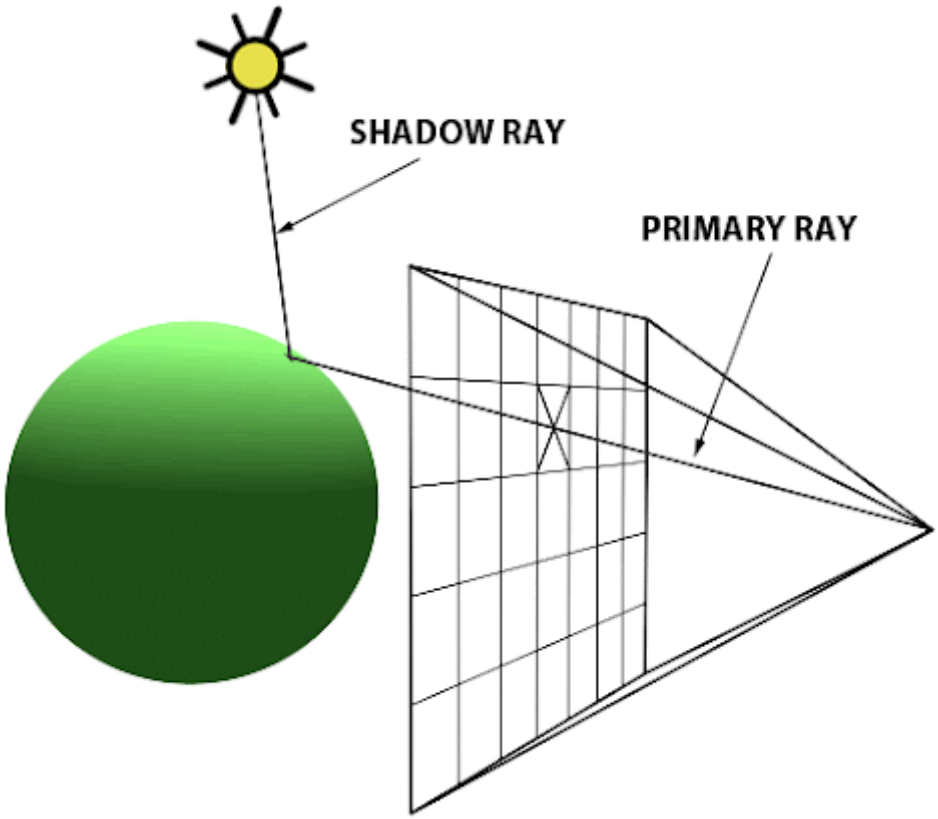
- Se recorren todos los objetos, sin ningún orden previo: el rayo no sabe cuál está adelante, así que no puede saltarse ninguno.
- Cada primitiva resuelve su propia ecuación, y todas devuelven lo mismo: una distancia sobre el rayo.
- El impacto se conserva solo si está frente a la cámara y es más cercano que el mejor encontrado hasta ese momento.
- Si el recorrido termina sin tocar nada, el píxel se pinta con el color de fondo.

## Rayos secundarios

El rayo primario solo responde qué se ve. Para saber cómo se ve, desde el punto de impacto se lanzan rayos nuevos, llamados rayos secundarios.

- Rayo de sombra: va del punto de impacto hacia la luz. Si encuentra un objeto en el camino, el punto está en sombra.
- Rayo de reflexión: rebota sobre la normal de la superficie y trae lo que se refleja en un material espejeante.

- Rayo de refracción: atraviesa la superficie y se dobla al cambiar de medio, como en el vidrio o el agua.

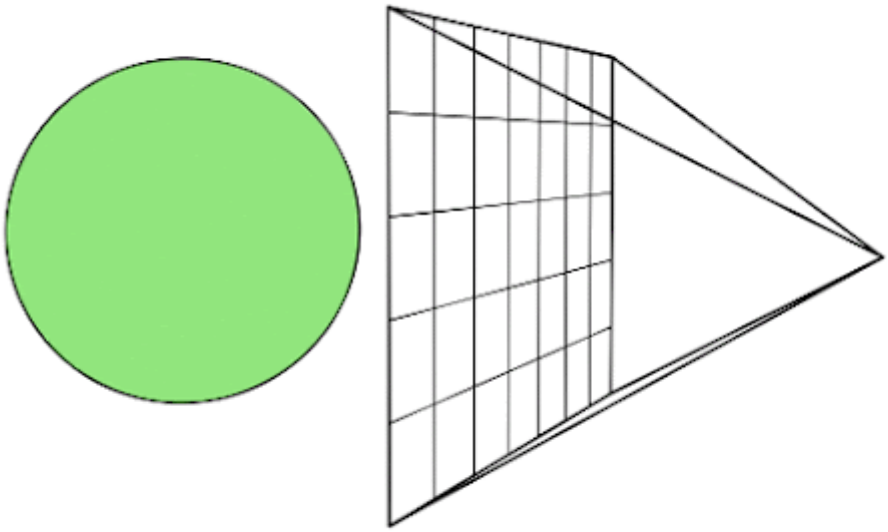



Cada rayo secundario puede golpear otra superficie y generar sus propios rayos, así que el proceso es recursivo. Se detiene con un límite de profundidad para que la recursión no sea infinita.

Esta capacidad es lo que distingue al ray tracing: las sombras, los reflejos y las refracciones no son efectos añadidos aparte, sino consecuencia directa de seguir trazando rayos.

## Construyendo la imagen

La combinación de todos los rayos genera la imagen final.



Es una proyección de alta definición, sin transformaciones geométricas de por medio.

Conceptualmente es un ciclo doble que recorre todos los píxeles de la imagen.

## Raycasting y ray tracing

El raycasting del maze y el ray tracing comparten la idea de lanzar rayos, pero resuelven problemas distintos.

| | **Raycasting** | **Ray Tracing** |
|---|---|---|
| Rayos | Uno por columna de pantalla | Uno o más por píxel |
| Escena | Cuadrícula 2D de celdas | Geometría 3D arbitraria |
| Dimensión del rayo | 2D, con altura simulada | 3D real |

| | Raycasting | Ray Tracing |
|---|---|---|
| Rebotes | Ninguno, el rayo termina al golpear | Recursivo: sombras, reflejos, refracciones |
| Iluminación | Trucos: sombreado por distancia o por lado | Modelo de luz calculado |
| Costo | Muy bajo, tiempo real desde 1992 | Alto, crece con la resolución y la escena |
| Ejemplo | Wolfenstein 3D | Cine, VFX, render de producto |

El raycasting se puede ver como el caso más restringido posible del ray tracing: rayos que solo viven en un plano, contra una escena que es una cuadrícula, sin rebotes. Esa restricción es exactamente lo que lo hacía viable en el hardware de los noventa.

El salto que vamos a dar es levantar esas restricciones una por una: el rayo pasa a ser tridimensional, la escena deja de ser una cuadrícula y el rayo aprende a rebotar.

## Casos de uso

- Cine y efectos visuales. Es el estándar de la animación y el VFX modernos: Pixar, Disney y las casas de efectos renderizan con trazado de rayos.
- Arquitectura y visualización de producto. Cuando el objetivo es que una imagen se vea como una fotografía del objeto real, el ray tracing es la herramienta.

- Videojuegos. Desde las GPU con unidades dedicadas al trazado de rayos, se usa en tiempo real de forma híbrida: rasterización para la imagen base y rayos para sombras, reflejos e iluminación global.
- Simulación científica y de ingeniería. Estudios de iluminación, análisis de radiación y trazado de ondas usan el mismo principio con otra magnitud física.
- Diseño óptico. Trazar rayos a través de lentes y espejos es el uso original de la técnica, anterior a las gráficas por computadora.

## Costo computacional

El ray tracing es caro y conviene entender por qué. Una imagen de 1920x1080 tiene poco más de dos millones de píxeles, y cada uno lanza al menos un rayo. Cada rayo se prueba contra los objetos de la escena, y cada impacto genera rayos secundarios que repiten el proceso.

Las estrategias para hacerlo manejable son de dos tipos:

- Reducir el número de pruebas. Estructuras de aceleración como bounding volumes, BVH o kd-trees descartan de golpe grandes porciones de la escena.
- Reducir el costo por rayo. Limitar la profundidad de recursión, simplificar el modelo de sombreado o bajar la resolución de trabajo.

Por eso durante décadas el ray tracing vivió solo en el render offline, donde un frame puede tardar horas, mientras la rasterización dominaba el tiempo real. La frontera se movió, pero el costo sigue siendo el eje de todas las decisiones de diseño de un raytracer.

## En el proyecto de gráficas

El raytracer se monta sobre lo que ya construimos en el curso:

- El framebuffer es donde se escribe el color de cada píxel.
- El main render loop envuelve el recorrido de píxeles y presenta el frame.
- La cámara define el origen de los rayos primarios y hacia dónde apuntan.
- El raycaster del maze ya entrenó la mecánica de lanzar un rayo y buscar su impacto; aquí ese rayo se vuelve tridimensional.

Las siguientes lecciones agregan las piezas que faltan: mover la cámara alrededor de la escena, calcular normales para sombrear correctamente, y trazar rayos secundarios para sombras, reflexiones y refracciones.