



UNIVERSIDAD  
DE GRANADA

# Informática Gráfica:

## Teoría. Tema 5. Realismo en Rasterización.

### Ray-tracing.

---

Carlos Ureña

2022-23

Grado en Informática y Matemáticas

Grado en Informática y Administración y Dirección de Empresas

Dpt. Lenguajes y Sistemas Informáticos

ETSI Informática y de Telecomunicación

Universidad de Granada

## Teoría. Tema 5. Realismo en Rasterización. Ray-tracing.

### Índice.

1. Técnicas realistas en rasterización
2. Ray tracing

## Sección 1. Técnicas realistas en rasterización.

- 1.1. Mipmaps.
- 1.2. Perturbación de la normal
- 1.3. Sombras arrojadas
- 1.4. Superficies transparentes. Refracción.
- 1.5. Superficies especulares

Informática Gráfica, curso 2022-23.

Teoría. Tema 5. Realismo en Rasterización. Ray-tracing.

Sección 1. Técnicas realistas en rasterización

Subsección 1.1.

Mipmaps..

# La resolución de las texturas.

En muchos casos la resolución a la que se ve la textura no coincide con la de la imagen sintetizada:

- ▶ Si la resolución es menor (objeto lejano), en un pixel se proyectan muchos texels.
- ▶ Si la resolución es mayor, un texel se proyecta en muchos pixels.

en ambos casos el efecto es una pérdida de realismo. El primer problema se puede solucionar usando anti-aliasing, o de forma mucho más eficiente usando la técnica de *mipmaps*

## Creación de los *mipmaps*

Un *mipmap* (de *multum in parvo maps*) es una serie de  $n + 1$  texturas (bitmaps) obtenida a partir de una imagen o textura de  $2^n \times 2^n$  texels.

- ▶ La primera imagen (imagen  $M_0$ ) coincide con la original
- ▶ La  $i$ -ésima imagen ( $M_i$ ) tiene como resolución  $2^{n-i} \times 2^{n-i}$  texels.
- ▶ Cada texel de la imagen  $i + 1$  ( $M_{i+1}$ ) se obtiene a partir de cuatro texels de la imagen número  $i$ , promediándolos:

$$M_{i+1}[j, k] = \frac{1}{4} ( M_i[2j, 2k] + M_i[2j + 1, 2k] + M_i[2j, 2k + 1] + M_i[2j + 1, 2k + 1] )$$

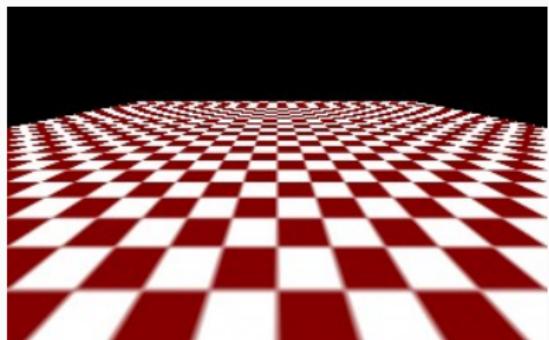
## Acceso a los *mipmaps*

Durante el sombreado, en cada punto  $\mathbf{p}$  a sombrear es necesario saber que versión de la trextura debemos de leer:

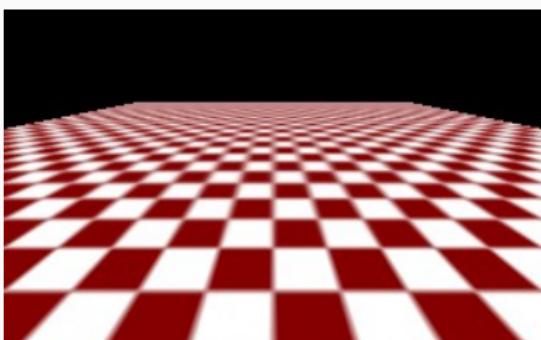
- ▶ Se usará la textura  $M_i$ , donde  $i$  crece linealmente con el logaritmo de la distancia  $d$  entre  $\mathbf{p}$  y el observador (menos resolución a mayor distancia).
- ▶ Esta solución puede presentar cambios bruscos de la resolución al pasar bruscamente de una resolución a otra en pixels cercanos. La solución consiste en interpolar entre las dos texturas más apropiadas en función de  $\log(d)$

## Ejemplo de *mipmapping*

En la parte más cercana se usa en ambos casos la textura original. Con mipmaps, a distancias mayores se usan sucesivamente texturas de menos resolución.



sin mipmapping



con mipmapping

[http://www.flipcode.com/archives/Advanced\\_OpenGL\\_Texture\\_Mapping.shtml](http://www.flipcode.com/archives/Advanced_OpenGL_Texture_Mapping.shtml)

Informática Gráfica, curso 2022-23.

Teoría. Tema 5. Realismo en Rasterización. Ray-tracing.

Sección 1. Técnicas realistas en rasterización

Subsección 1.2.

Perturbación de la normal.

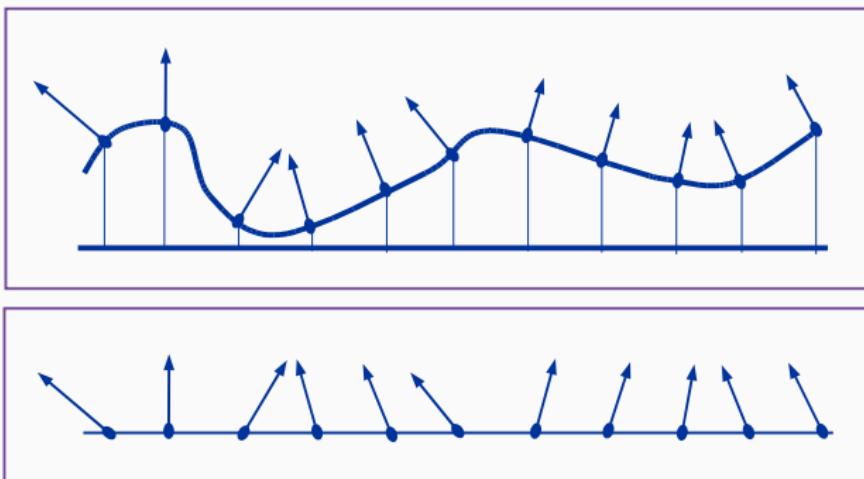
## Rugosidades a pequeña escala

Algunos tipos de superficies presentan cambios de orientación a pequeña escala (rugosidades)

- ▶ Esto se puede reproducir con mallas de polígonos con muchos polígonos pequeños, o con polígonos de detalle de diferente orientación. En cualquier caso, la complejidad en tiempo y espacio del proceso de rendering es muy alta.
- ▶ Una solución consiste en usar un textura para modificar a pequeña escala el vector normal que se usa en el MIL, a esto se le llama *mapas de perturbación de la normal (bump-maps)*.

# Rugosidades definidas por un campo de alturas

Es necesario usar una función real  $f_h$ , tal que para cada par de coordenadas de textura  $(u, v)$ , el valor real  $f_h(u, v)$  se interpreta como la altura de la superficie rugosa respecto del plano del polígono en el punto de coordenadas de textura  $(u, v)$



## Codificación del campo de alturas

Para evaluar  $f_h(u, v)$  dados  $u$  y  $v$  se pueden usar dos opciones:

- ▶  $f_h$  puede representarse como una función con una expresión analítica conocida y evaluable con algún algoritmo que tiene a  $u$  y  $v$  como datos de entrada (se llaman *texturas procedurales*).
- ▶ la opción más usual es que  $f_h$  este codificada como una textura cuyos texels son valores escalares (tonos de gris) que codifican la altura. Para evaluar  $f_h(u, v)$  se usa el mismo método visto para acceso a texturas en la sección anterior (se usan los texels más cercanos a  $(u, v)$  en el espacio de coords. de textura).

## Derivadas del campo de alturas

El procedimiento de perturbación de la normal usa como parámetros las derivadas parciales de  $f_h$  ( $d_u$  y  $d_v$ ):

$$d_u = \frac{\partial f_t(u, v)}{\partial u} \quad d_v = \frac{\partial f_t(u, v)}{\partial v}$$

- ▶ si  $f_h$  está definido por una función analítica conocida y derivable, estas derivadas se pueden conocer evaluando las expresiones de las derivadas parciales de  $f_h$ .
- ▶ si  $f_h$  está codificada con una textura, se usan diferencias finitas

# Aproximación de las derivadas por diferencias finitas

Cuando el campo de alturas  $f_t$  se codifica con una textura de grises, los valores de  $d_u$  y  $d_v$  se deben aproximar por diferencias finitas:

$$d_u \approx k \frac{f_h(u + \Delta, v) - f_h(u - \Delta, v)}{2\Delta}$$

$$d_v \approx k \frac{f_h(u, v + \Delta) - f_h(u, v - \Delta)}{2\Delta}$$

donde:

- ▶  $\Delta$  es usualmente del orden de  $1/n_t$  ( $n_t$  = resol. de la textura).
- ▶  $k$  es un valor real que sirve para atenuar o exagerar el relieve

## La superficie como una función de las c.t.

Los puntos de los polígonos que forman las superficies de los objetos pueden interpretarse como una función  $f_p$  de las coordenadas de textura, es decir, si las coord. de textura de un punto  $q$  son  $(u, v)$ , entonces:

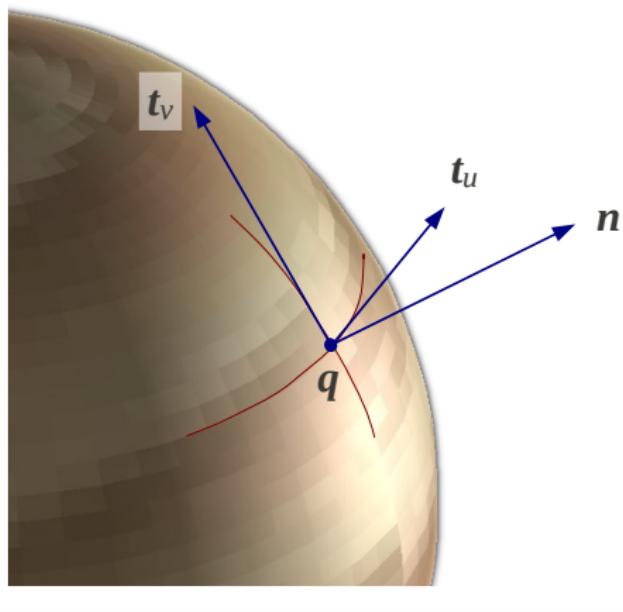
$$q = f_p(u, v)$$

para calcular la normal modificada es necesario conocer las derivadas parciales de  $f_p$  (dos vectores  $t_u$  y  $t_v$ )

$$t_u = \frac{\partial f_p(u, v)}{\partial u} \quad t_v = \frac{\partial f_p(u, v)}{\partial v}$$

# Las tangentes y la normal

A los vectores  $t_u$  y  $t_v$  se les suele llamar *tangente* y *bitangente*. Ambos definen un plano tangente, perpendicular a la normal.



## Cálculo de los vectores tangentes modificados

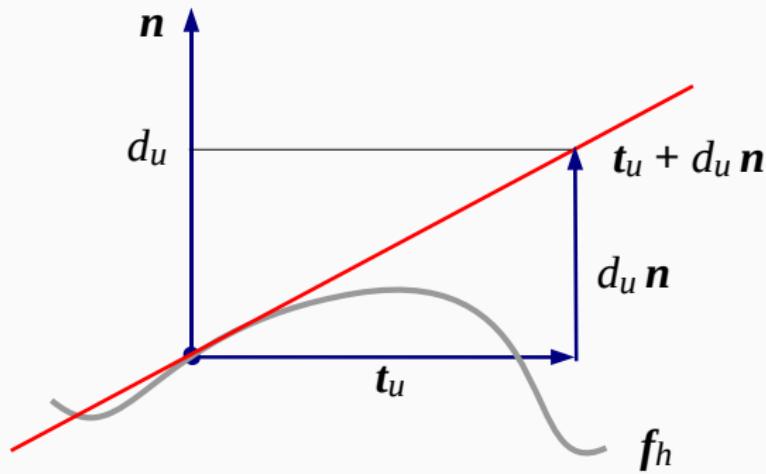
Estos vectores son tangentes a la superficie del objeto, ya que la normal original  $\mathbf{n}$  es colineal con  $\mathbf{t}_u \times \mathbf{t}_v$ . Existen varias alternativas para obtenerlos:

- ▶ Para objetos sencillos, los vectores tangentes son constantes o muy fáciles de calcular
- ▶ Para mallas de polígonos:
  - ▶ Se pueden calcular como constantes en cada polígono, a partir de las coordenadas de textura.
  - ▶ Se pueden asignar a los vértices (igual que las c.t.) y realizar una interpolación en el interior de los polígonos (igual que se interpola la normal).

## Obtención de las tangentes modificadas

Los vectores tangentes  $t'_u$  y  $t'_v$  a la superficie rugosa son:

$$t'_u = t_u + d_u \mathbf{n} \quad t'_v = t_v + d_v \mathbf{n}$$



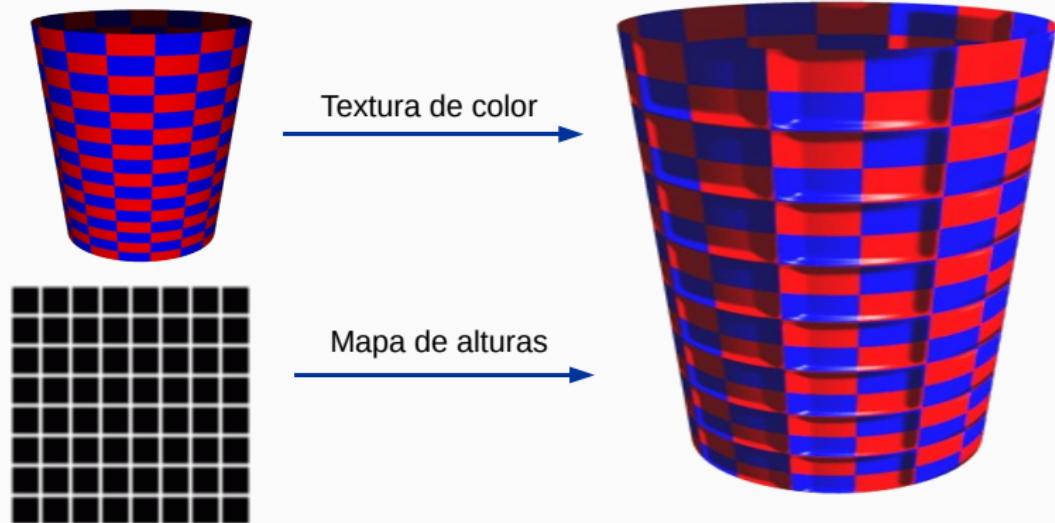
la normal modificada  $\mathbf{n}'$  es perpendicular a estos dos vectores, por tanto se calcula usando su producto vectorial (y normalizando)

$$\mathbf{n}' = \frac{\mathbf{n}''}{\|\mathbf{n}''\|} \quad \text{donde: } \mathbf{n}'' = \mathbf{t}_u' \times \mathbf{t}_v'$$

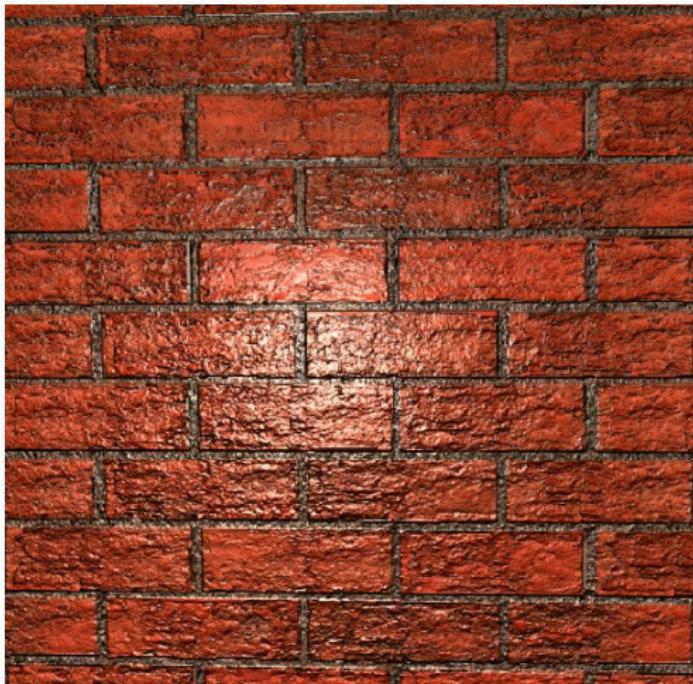
# Ejemplo de texturas + perturbación de la normal (1)

Imágenes de Fredo Durand y Barb Curtler:

<http://groups.csail.mit.edu/graphics/classes/6.837>



## Ejemplo de texturas + perturbación de la normal (2)



Informática Gráfica, curso 2022-23.

Teoría. Tema 5. Realismo en Rasterización. Ray-tracing.

Sección 1. Técnicas realistas en rasterización

Subsección 1.3.

Sombras arrojadas.

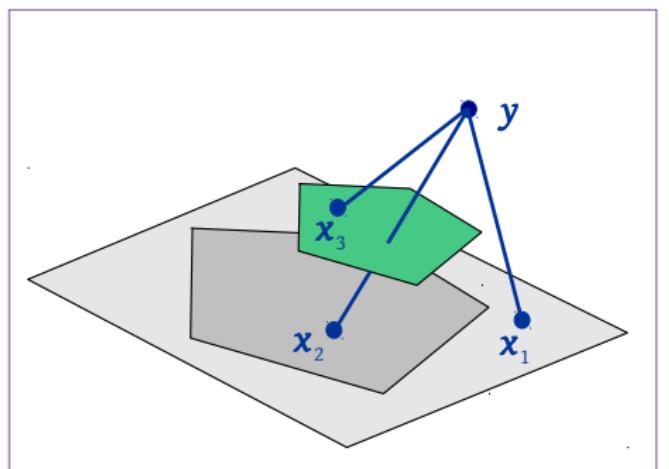
## Sombras arrojadas y el MIL

Ninguna de las técnicas anteriores tiene en cuenta la existencia de sombras arrojadas.

- ▶ Se supone que todas las fuentes son visibles desde todos los puntos de la superficie, lo cual no siempre es cierto.
- ▶ Si asumimos que los polígonos son opacos, y las fuentes puntuales (o direccionales), para cada punto en una superficie y para cada fuente de luz, el punto y la fuente pueden ser mutuamente visibles o no.
- ▶ Cuando la fuente no ilumina el punto, el sumando del MIL correspondiente a la fuente no debe añadirse para obtener el color reflejado.

# La función de visibilidad $V$

La visibilidad de la fuente de luz (en  $y$ ) está controlada por la función  $V$ :



$$V(x_1, y) = 1 \quad V(x_2, y) = 0 \quad V(x_3, y) = 1$$

## Sombras arrojadas y visibilidad

El problema de las sombras arrojadas es, por tanto, semejante al problema de la visibilidad:

- ▶ Se pueden usar algoritmos con precisión de objetos: se producen en la salida los polígonos (parte de los originales) iluminados por (visibles desde) las fuentes de luz.
- ▶ Se pueden usar algoritmos con precisión de imagen: se obtiene el primer punto visible en el centro de cada pixel de un plano de visión asociado a una fuente de luz.

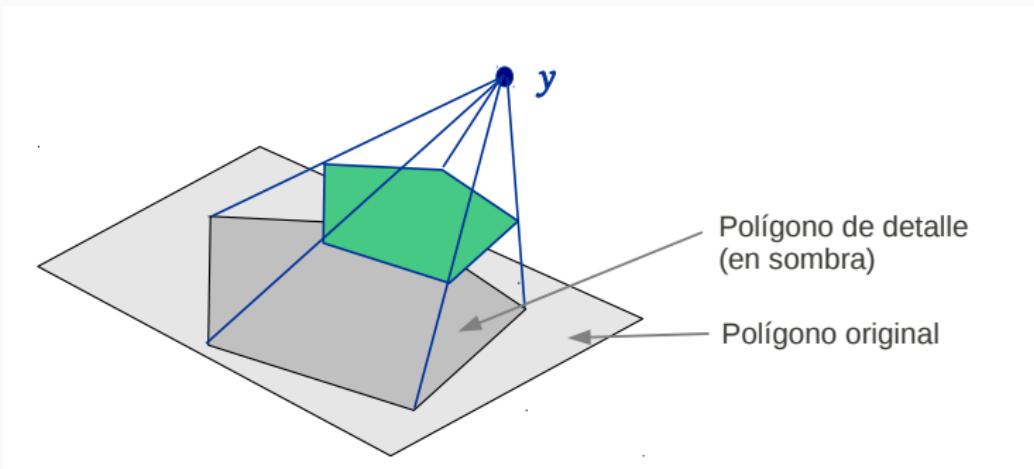
El papel del observador lo juega la fuente de luz. Puede ser posicional (observador a distancia finita) o direccional (observador a distancia infinita: proyección ortogonal).

## Algoritmo de fuerza bruta

Supondremos escenas formadas por poliedros opacos delimitados por caras planas o polígonos planos individuales.

- ▶ El algoritmo más sencillo consiste en proyectar todos los polígonos contra todos, usando la fuente de luz como foco.
- ▶ Para cada par de polígonos  $P$  y  $Q$  se calcula el polígono de sombra arrojada  $S$  que proyecta  $P$  sobre  $Q$  (si hay alguna), y se recorta  $S$  usando  $Q$  como polígono de recorte.
- ▶ Los polígonos producidos se tratan como polígonos de detalle. Son polígonos superpuestos a los originales en los cuales la fuente de luz no es visible.

## Algoritmo de fuerza bruta (2)



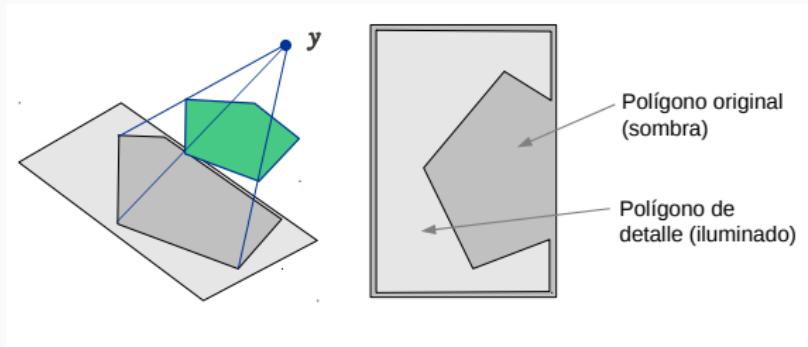
- ▶ tiene complejidad cuadrática con el número de polígonos
- ▶ se puede usar solo para un único polígono receptor y unos pocos que arrojan sombras.

## Algoritmo de Weiler-Atherton-Greenberg

Otros algoritmos de sombras arrojadas (más eficientes) están basados en algoritmos de eliminación de partes ocultas ya existentes. Un ejemplo es el algoritmo de Weiler-Atherton-Greenberg (1978) para sombras arrojadas:

- ▶ Se usa el algoritmo de Weiler-Atherton para eliminación de partes ocultas
- ▶ Se produce un modelo con polígonos iluminados asociados a los originales (son también polígonos de detalle).
- ▶ La complejidad en tiempo es mucho menor que cuadrática en el caso medio.

## Algoritmo de Weiler-Atherton-Greenberg (2)

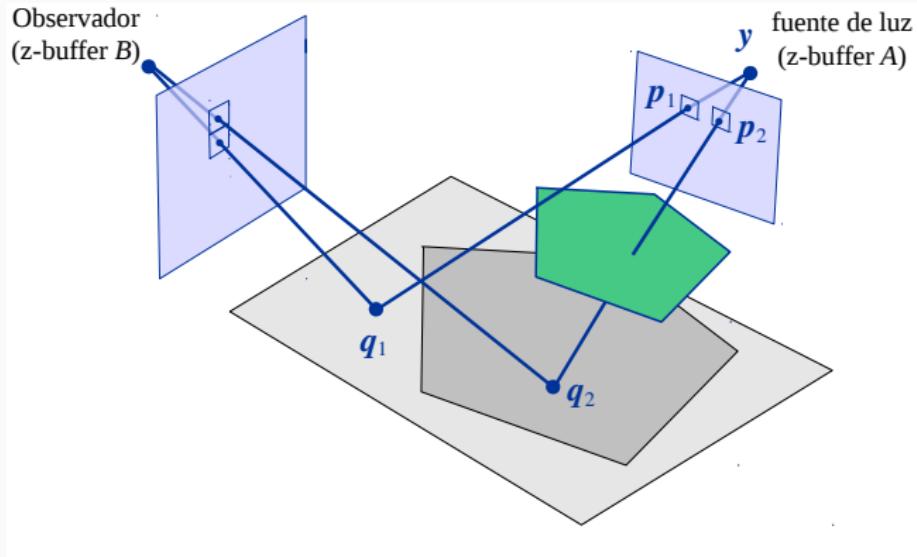


En general, los algoritmos con precisión de objetos para sombras son:

- ▶ muy complejos en tiempo para escenas complejas
- ▶ para algunas aplicaciones son los más idóneos (cuando se necesita un resultado en forma de dibujo vectorial).

# Z-buffer para sombras arrojadas

Otra posibilidad (mucho más eficiente) es usar Z-buffer para sombras arrojadas:

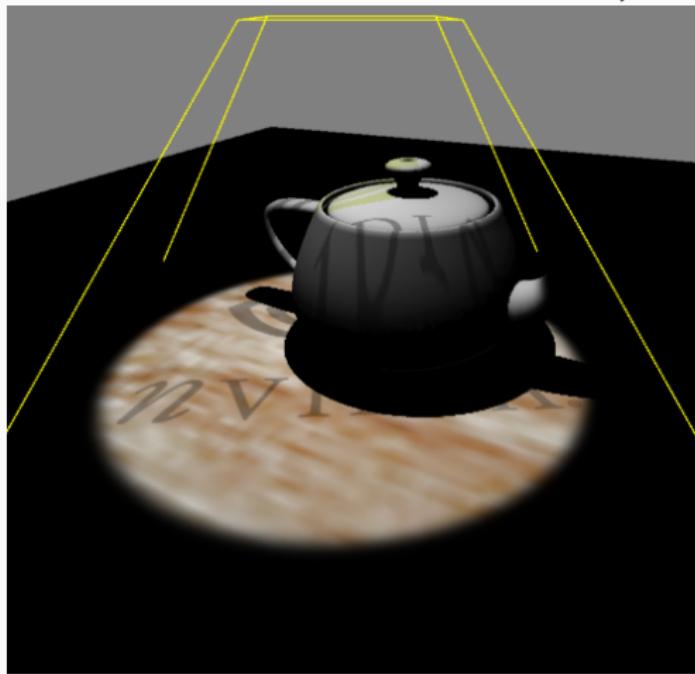


## Z-buffer para sombras arrojadas (2)

- ▶ En la primera pasada se calcula el Z-buffer  $A$  asociado a la fuente de luz (se proyectan los objetos contra la fuente)
- ▶ La segunda pasada es semejante al Z-buffer normal, se calcula el Z-buffer  $B$ , para cada punto visible  $\mathbf{q}_i$  desde el observador en un pixel, se debe calcular el color con el que se ve  $\mathbf{q}_i$ , y por tanto es necesario comprobar si es visible desde la fuente, para ello:
  - ▶ se calcula  $\mathbf{p}_i$  (la proyección de  $\mathbf{q}_i$  en el plano de visión asociado a la fuente de luz)
  - ▶ se accede al pixel del Z-buffer  $A$  correspondiente a  $\mathbf{p}_i$ , que contiene una distancia  $d$
  - ▶ si  $d < \|\mathbf{q}_i - \mathbf{y}\|$ , entonces  $\mathbf{q}_i$  no está iluminado (este es el caso de la figura), en otro caso  $\mathbf{q}_i$  sí está iluminado.

# Ejemplo de Z-buffer para sombras

(se proyecta una textura desde la fuente en los objetos):



## Errores de Z-buffer para sombras

son visibles si el observador está cerca de ellas en comparación con la distancia a la fuente de luz:



Informática Gráfica, curso 2022-23.

Teoría. Tema 5. Realismo en Rasterización. Ray-tracing.

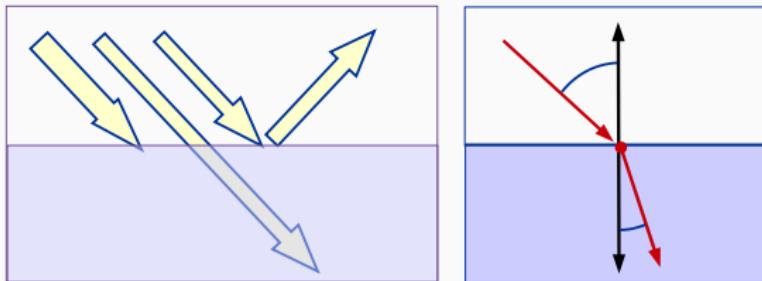
Sección 1. Técnicas realistas en rasterización

Subsección 1.4.

Superficies transparentes. Refracción..

# Materiales transparentes

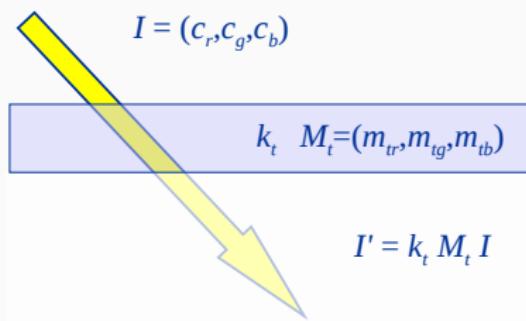
Hay objetos sólidos o líquidos que permiten pasar (además de reflejar o absorber) algunos fotones de la luz que los alcanza. Su estructura atómica permite a los rayos de luz viajar en línea recta.



la luz cambia de dirección debido a su progreso más lento en estos medios (debido al retraso por múltiples eventos de dispersión de fotones)

# Cambio del color en la refracción

Al pasar por un objeto delgado transparente, la cantidad de luz que no es absorbida en el medio (y atraviesa el objeto) depende de la longitud de onda:



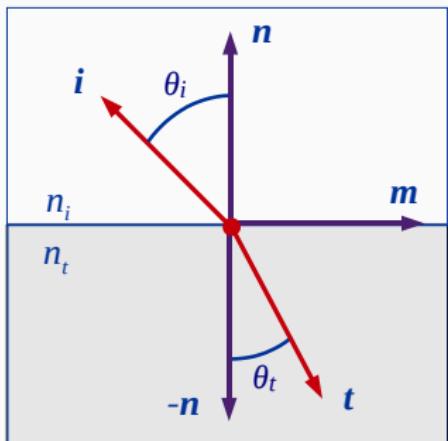
- ▶ La fracción global de luz refractada es  $k_t$ , que está entre 0 y 1
- ▶ En cada longitud de onda se refracta una fracción distinta, en RGB estas fracciones son un color  $M_t = (m_{tr}, m_{tg}, m_{tb})$

Por tanto, estos materiales están caracterizados por  $k_t$  y  $M_t$

# Cambio de dirección en la refracción

El vector  $\mathbf{t}$  puede calcularse a partir de  $\mathbf{n}$ ,  $\mathbf{i}$ , y los índices de refracción  $n_i$  y  $n_t$ , teniendo en cuenta la ley de Snell:

$$n_i \sin \theta_i = n_t \sin \theta_t$$



$$\mathbf{t} = (r \cos \theta_i - \cos \theta_t) \mathbf{n} - r \mathbf{i}$$

$$\cos \theta_i = \mathbf{i} \cdot \mathbf{n}$$

$$\cos \theta_t = \sqrt{1 - r^2 [1 - (\mathbf{i} \cdot \mathbf{n})^2]}$$

$$r = n_i / n_t$$

Si  $1 < r^2 [1 - (\mathbf{i} \cdot \mathbf{n})^2]$  entonces no hay refracción (hay *reflexión interna total*). Solo puede ocurrir cuando  $n_t < n_i$ , para  $\theta_i > \theta_{\max}$ .

## Superficies transparentes en Z-buffer

El método de Z-buffer solo puede tener en cuenta, para un pixel, los colores de los puntos en el proyector o rayo que pasa por el centro del pixel hacia el observador.

- ▶ Cuando  $n_i \neq n_t$  los rayos se desvían, y esto no puede reproducirse con Z-buffer
- ▶ Si suponemos que  $n_i = n_t$ , entonces no hay cambio de dirección debida a la refracción, y por tanto  $t = -i$
- ▶ Con esta simplificación, se puede adaptar el método de Z-Buffer para incluir polígonos transparentes.

a continuación vemos un esbozo del método

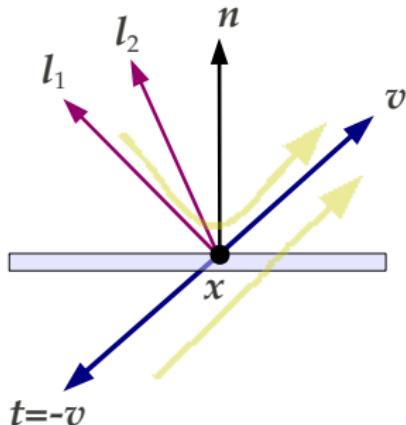
# Z-buffer adaptado a superficies transparentes

El algoritmo dibuja primero los polígonos opacos, y después los transparentes o semi transparentes (en cualquier orden)

- 
- 1: Inicializar Z-buffer ( $Z$ ) e Imagen ( $I$ )
  - 2: **for** cada polígono opaco  $P$  **do**
  - 3:     Rasterizar  $P$ , actualizando  $Z$  e  $I$
  - 4: **for** cada polígono transparente  $Q$  **do**
  - 5:      $k_t$  = fraccion de luz refractada de  $Q$
  - 6:      $M_t$  = color transparente de  $Q$
  - 7:     **for** cada pixel  $(x, y)$  ocupado por  $Q$  **do**
  - 8:         **if**  $Q$  es visible en  $(x, y)$  **then**
  - 9:              $I[x, y] = k_t M_t I[x, y]$
-

# Combinación de reflexión y refracción

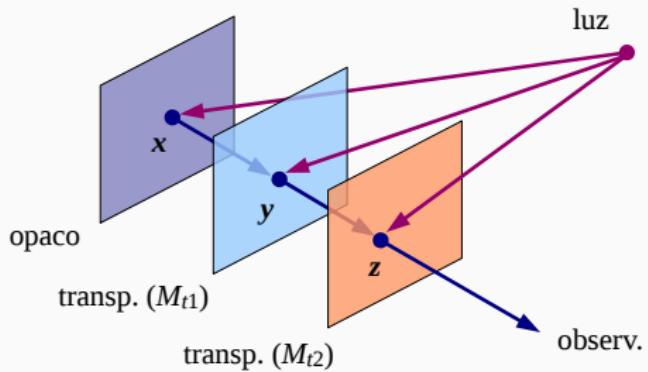
En la superficie entre una lámina de material transparente y el espacio (vacío) entre objetos puede también reflejarse la luz:



Si  $k_t > 0$ , al MIL debe sumársele la luz refractada proveniente del otro lado del polígono, en la dirección de  $v$

# Dependencia del orden

El color  $I$  que percibe el observador depende de los colores  $I_x$ ,  $I_y$  y  $I_z$  reflejados en los puntos  $x, y$  y  $z$ :



Este color depende del orden de los polígonos:

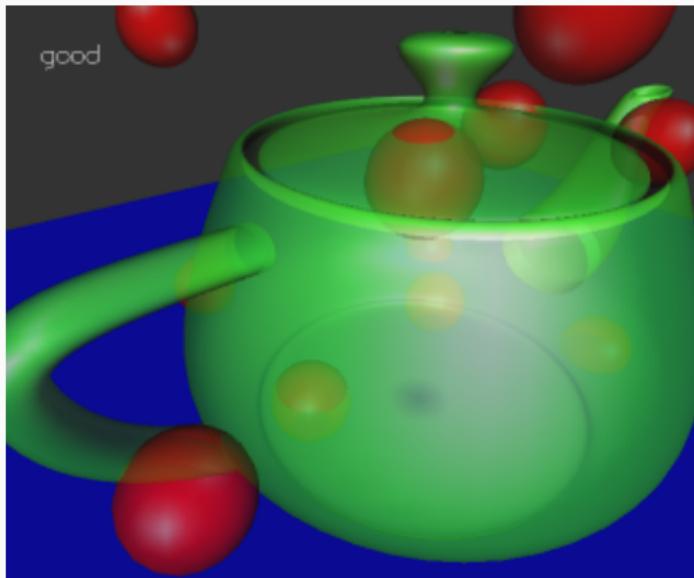
$$I = I_z + M_{t2}I_y + M_{t1}M_{t2}I_x \neq I_z + M_{t1}I_x + M_{t2}M_{t1}I_y$$

# Z-buffer en superf. transparentes y reflectantes

Los polígonos transp. deben dibujarse en orden de Z:

- 
- 1: Inicializar Z-buffer ( $Z$ ) e Imagen ( $I$ )
  - 2: **for** cada polígono opaco  $P$  **do**
  - 3:     Rasterizar  $P$ , actualizando  $Z$  e  $I$
  - 4: **for** cada polígono transparente  $Q$  (en orden de Z) **do**
  - 5:      $k_t$  = fraccion de luz refractada de  $Q$
  - 6:      $M_t$  = color transparente de  $Q$
  - 7:     **for** cada pixel  $(x,y)$  ocupado por  $Q$  **do**
  - 8:         **if**  $Q$  es visible en  $(x,y)$  **then**
  - 9:              $I_m$  = resultado de evaluar MIL
  - 10:             $I[x,y] = I_m + k_t M_t I[x,y]$
-

# Ejemplo de materiales transparentes



Informática Gráfica, curso 2022-23.

Teoría. Tema 5. Realismo en Rasterización. Ray-tracing.

Sección 1. Técnicas realistas en rasterización

Subsección 1.5.

Superficies especulares.

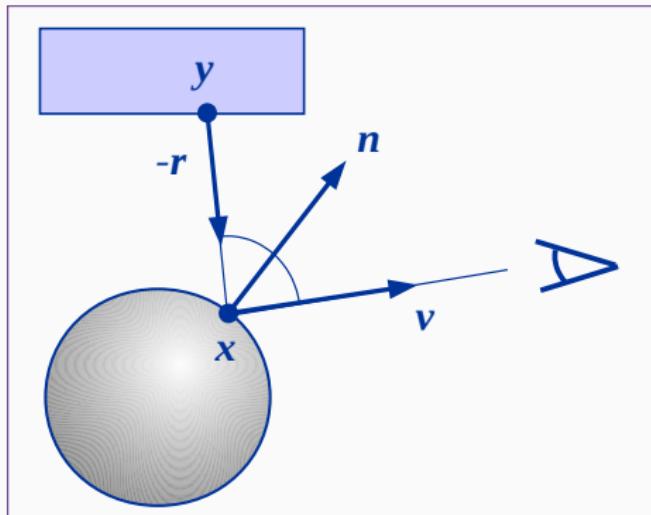
## Reflexión especular de la luz

Algunos objetos pulidos reflejan la luz de forma especular perfecta, como los espejos planos

- ▶ La componente especular perfecta es una componente más del modelo de iluminación local, que se suma a las anteriores (se suele dar en combinación con la refractada en los objetos de cristal).
- ▶ Este efecto no puede reproducirse con ninguno de los métodos que hemos visto, pues la iluminación no procede de la dirección del rayo central a un pixel, sino de otras direcciones.

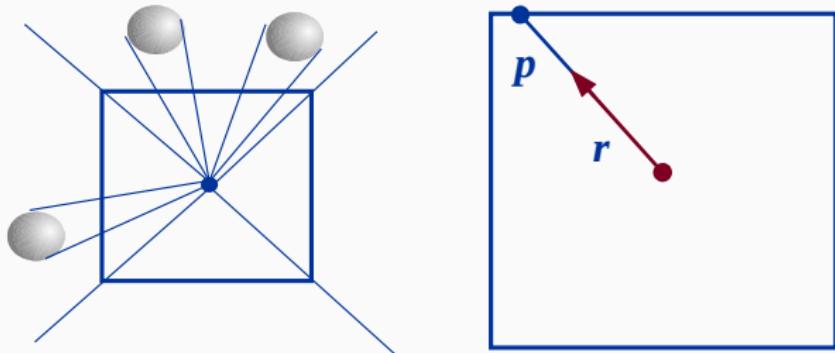
## Reflexión especular de la luz

Si la esfera es perfectamente reflectante, el color de  $x$  visto desde  $v$  es igual al color de  $y$  visto desde la dirección  $-r$  (el vector reflejado  $r$ , cambiado de signo).



## Mapas de entorno tipo caja (*box map*)

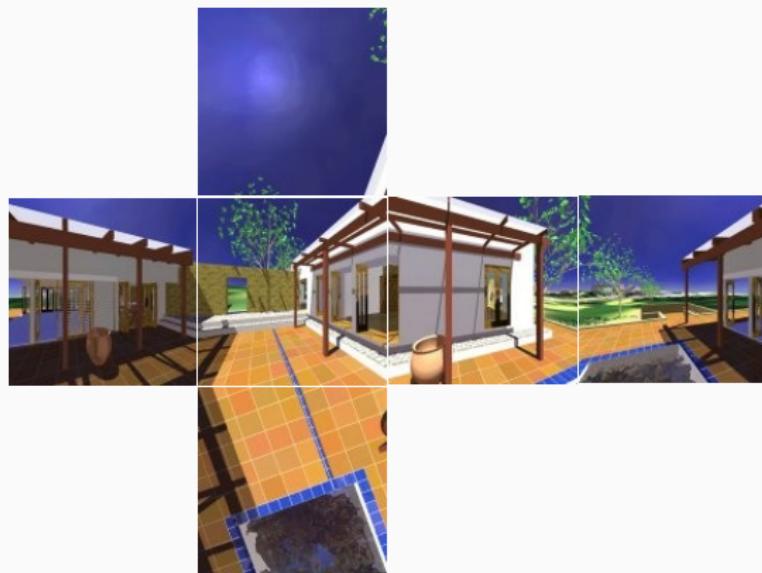
En esta técnica, el entorno se proyecta en las 6 caras de un cubo centrado en el objeto reflectante, obteniéndose 6 texturas.



En tiempo de rendering, el vector  $r$  se proyecta sobre la cara que corresponda (se calcula  $p$ ), y se obtienen el color RGB del texel que contiene a  $p$ .

# Texturas para mapas de entorno tipo caja

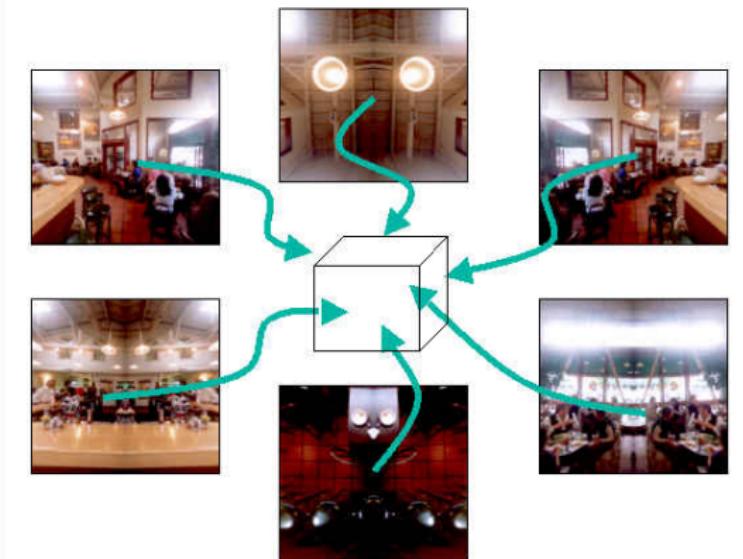
Ejemplo de 6 texturas para mapas de entorno



[http://developer.nvidia.com/object/cube\\_map\\_ogl\\_tutorial.html](http://developer.nvidia.com/object/cube_map_ogl_tutorial.html)

# Uso de fotografías de un entorno real

Tambien es posible usar fotografías de un entorno real



[http://developer.nvidia.com/object/cube\\_map\\_ogl\\_tutorial.html](http://developer.nvidia.com/object/cube_map_ogl_tutorial.html)

# Mapa de entorno esférico

Una sola imagen codifica el color reflejado para todas las posibles orientaciones de la normal



se asume una proyección ortográfica fija, en la cual el vector  $v$  es constante y paralelo al eje Z.

# Panoramas equirectangulares

Es una sola imagen que codifica, en cada texel  $(u, v)$ , la irradiancia en una dirección de coordenadas polares  $\alpha = au$  y  $\beta = bv$ :



se suelen obtener a partir de múltiples fotografías de un entorno. A su vez, pueden servir para crear mapas de entorno esféricos.

# Mapa de entorno esférico

Ejemplo del mapa de entorno esférico anterior en la tetera:



## Mapa de entorno

Ejemplo de combinación con texturas y perturbación de la normal.  
Además, la tetera se muestra en el entorno que refleja:

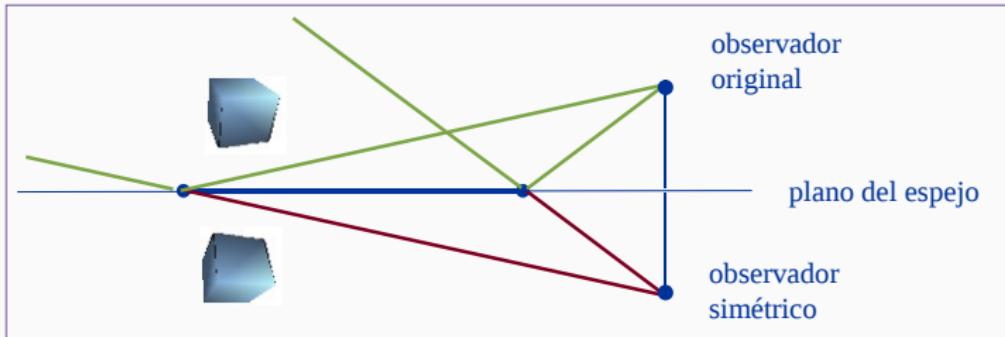


## Ejemplo en cine de animación



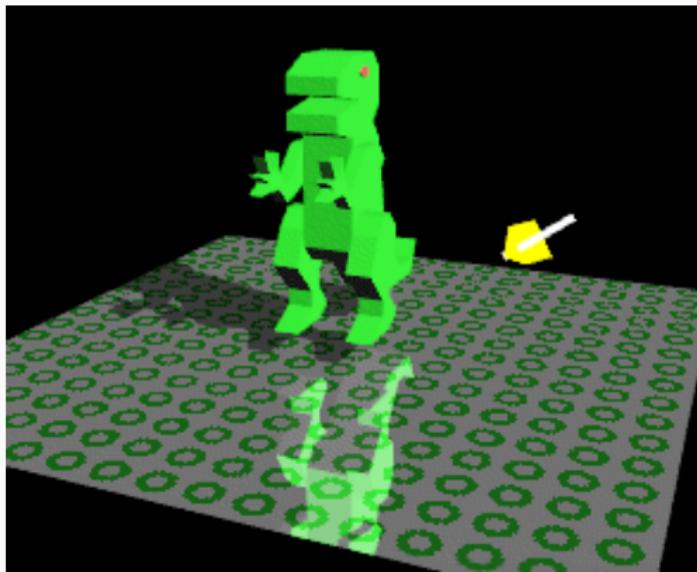
# Reflexión en espejos planos

En estos objetos, la escena reflejada es simétrica respecto de la original respecto del plano del espejo:



Se pueden reproducir las reflexiones sintetizando la imagen vista por una cámara *simétrica* respecto de la original.

## Ejemplo de duplicación de entorno:



## Sección 2. Ray tracing.

- 2.1. El algoritmo de Ray-Tracing
- 2.2. Intersecciones rayo-objeto y rayo-escena
- 2.3. Problemas: Cálculo de intersecciones
- 2.4. Ejemplos

Informática Gráfica, curso 2022-23.

Teoría. Tema 5. Realismo en Rasterización. Ray-tracing.

Sección 2. Ray tracing

Subsección 2.1.

El algoritmo de Ray-Tracing.

# Introducción

Hemos visto bastantes efectos:

- ▶ Superficies difusas y pseudo-especulares
- ▶ Texturas y mapas de perturbación de la normal
- ▶ Sombras arrojadas
- ▶ Superficies transparentes
- ▶ Superficies especulares perfectas

Considerarlos todos en visualización por rasterización lleva a software que es bastante complicado de implementar. Además los tiempos de síntesis de imágenes pueden hacerse bastante altos.

# La técnica de *Ray-Tracing*

Existe un algoritmo no muy eficiente en tiempo, pero bastante sencillo, que tiene en cuenta todos los efectos anteriores:

- ▶ Este algoritmo es el algoritmo de *Ray-Tracing* (*seguimiento de rayos*, usualmente traducido por *trazado de rayos*)
- ▶ Descrito completamente por primera vez por Turner Whitted en 1979-80.
- ▶ Está basado en la EPO por Ray-Casting, combinada con evaluación del MIL
- ▶ Es conceptualmente muy sencillo, y fácil de implementar.
- ▶ Obtiene un grado de realismo muy superior a Z-buffer, a costa de tiempos de cálculo usualmente más altos.

## Ventajas de la técnica

Frente a rasterización, Ray-Tracing puede visualizar objetos

- ▶ curvos (con superficie definida por ecuaciones implícitas).
- ▶ especulares perfectos (que reflejan el entorno).
- ▶ transparentes (con índices de refracción arbitrarios).
- ▶ con sombras arrojadas por otros.

Existen diversos algoritmos basados en Ray-Tracing con funcionalidad adicional:

- ▶ **Ray-Tracing distribuido**: fuentes de luz extensas, profundidad de campo, desenfoque de movimiento (*motion-blur*).
- ▶ **Path-Tracing**: iluminación indirecta o global.

El algoritmo de **Path-Tracing** y derivados se usa en generación por ordenador de películas y efectos especiales.

# Un M.I.L. extendido para Ray-Tracing

Ray-Tracing calcula la radiancia  $L_{\text{in}}(\mathbf{q}, \mathbf{u})$  incidente sobre un punto  $\mathbf{q}$  proveniente de una dirección  $\mathbf{u}$  (un vector unitario).

$$\begin{aligned} L_{\text{in}}(\mathbf{q}, \mathbf{u}) &= L(\mathbf{p}, \mathbf{v}) \\ &= M_E(\mathbf{p}) + L_{\text{ind}}(\mathbf{p}, \mathbf{v}) + \sum_{i=0}^{n_L-1} v_i \cdot S_i \cdot L_{\text{dir}}(\mathbf{p}, \mathbf{v}, \mathbf{l}_i) \end{aligned}$$

donde:

- ▶ El punto  $\mathbf{p}$  es el primero visible desde  $\mathbf{q}$  en la dirección de  $\mathbf{u}$ .
- ▶ El vector  $\mathbf{v}$  es la dirección de salida de radiancia (es  $-\mathbf{u}$ ).
- ▶  $M_E(\mathbf{p})$  es la emisividad en  $\mathbf{p}$ .
- ▶ El valor  $v_i$  es la visibilidad de la  $i$ -ésima fuente de luz (vale 0 o 1).
- ▶ Las funciones  $L_{\text{dir}}$  y  $L_{\text{ind}}$  representan la **iluminación directa** y la **iluminación indirecta**, respectivamente.

# MIL de Ray-Tracing. Iluminación directa.

La iluminación directa  $L_{\text{dir}}(\mathbf{p}, \mathbf{v}, \mathbf{l})$  es la radiancia reflejada en  $\mathbf{p}$  hacia  $\mathbf{v}$  debida a iluminación directa proveniente de una fuente de luz que está en la dirección  $\mathbf{l}$ . Se define así:

$$\begin{aligned} L_{\text{dir}}(\mathbf{p}, \mathbf{v}, \mathbf{l}) &\equiv k_a(\mathbf{p}) \cdot C(\mathbf{p}) \\ &+ k_d(\mathbf{p}) \cdot C(\mathbf{p}) \cdot \max(0, \mathbf{n} \cdot \mathbf{l}) \\ &+ k_s(\mathbf{p}) \cdot d_i \cdot [\max(0, \mathbf{r} \cdot \mathbf{v})]^e \end{aligned}$$

esta fórmula incorpora las componentes ambiental, difusa y pseudo-especular o *glossy*, de forma similar a como vimos en el tema 3 para rasterización. Aquí:

- ▶  $\mathbf{r} \equiv$  vector reflejado en  $\mathbf{p}$  (depende de  $\mathbf{v}$  y  $\mathbf{n}$ ).
- ▶  $\mathbf{n} \equiv$  normal en  $\mathbf{p}$  (depende de  $\mathbf{p}$  y  $O$ ).
- ▶  $k_a, k_d, k_s \equiv$  parámetros del material en  $\mathbf{p}$ .
- ▶  $C(\mathbf{p}) \equiv$  color del objeto en  $\mathbf{p}$

# MIL de Ray-Tracing. Iluminación indirecta.

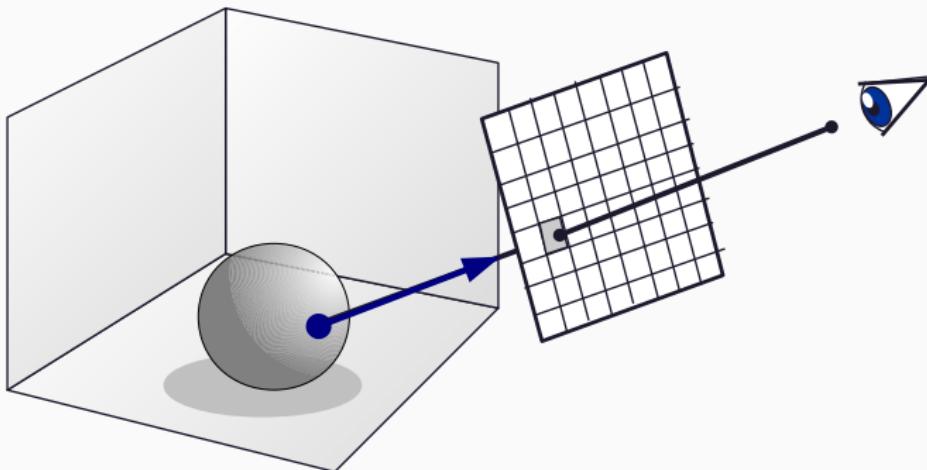
El término  $L_{\text{ind}}(\mathbf{p}, \mathbf{v})$  incluye la radiancia ambiente global, la reflejada perfectamente y la refractada. Se define **recursivamente** en términos de  $L_{\text{in}}$ :

$$L_{\text{ind}}(\mathbf{p}, \mathbf{v}) \equiv A_G(\mathbf{p}) + k_{\text{ps}}(\mathbf{p})M_{\text{PS}}(\mathbf{p})L_{\text{in}}(\mathbf{p}, \mathbf{r}) + k_t(\mathbf{p})M_T(\mathbf{p})L_{\text{in}}(\mathbf{p}, \mathbf{t})$$

- ▶  $A_G(\mathbf{p}) \equiv$  radiancia ambiente global (suple ilum. indirecta).
- ▶  $k_t(\mathbf{p}) \equiv$  fracción de luz refractada en  $\mathbf{p}$ .
- ▶  $k_{\text{ps}}(\mathbf{p}) \equiv$  fracc. de luz refl. de forma especular perfecta en  $\mathbf{p}$ .
- ▶  $M_{\text{PS}}(\mathbf{p})$  y  $M_T(\mathbf{p})$  son ternas RGB (con valores en  $[0, 1]$ ) que permiten modular el color de la componente refejada perfectamente o refractada.
- ▶  $\mathbf{r} \equiv$  vector reflejado en  $\mathbf{p}$  (depende de  $\mathbf{v}$  y  $\mathbf{n}$ ).
- ▶  $\mathbf{t} \equiv$  vector refractado en  $\mathbf{p}$  (depende de  $\mathbf{v}$  y  $\mathbf{n}$ ).

# Generación de Rayos-primarios

Los pixels se procesan secuencialmente, en cada uno se crea un rayo (llamado *rayo primario* o *rayo de cámara*) y se determina el primer objeto visible por Ray-Casting:



# El procedimiento principal de Ray-Tracing

El pseudocódigo del algoritmo, que recorre todos los pixels, puede quedar así:

- 
- 1:  **$\mathbf{o}$**  := posición del observador, en coords. del mundo
  - 2: **for** cada pixel  $(i, j)$  de la imagen **do**
  - 3:    **$\mathbf{q}$**  := punto central (en WCC) del pixel  $(i, j)$
  - 4:    **$\mathbf{u}$**  := vector desde  **$\mathbf{o}$**  hasta  **$\mathbf{q}$**  normalizado
  - 5:    **$rad$**  := RAYTRACING( **$\mathbf{o}, \mathbf{u}, 1$** )
  - 6:   fijar el pixel  $(i, j)$  al valor  **$rad$**
- 

- ▶ La función RAYTRACING es recursiva, devuelve un color, y tiene un parámetro que sirve para que la recursión no se haga infinita.
- ▶ Los colores de los pixels no están acotados, así que puede ser necesario un paso de post-proceso para normalizar la imagen.

# La función RAYTRACING

Esta función calcula la radiancia incidente sobre el punto **o**, proveniente de la dirección **u** (como una terna RGB). El entero *n* es el nivel de profundidad en las llamadas recursivas.

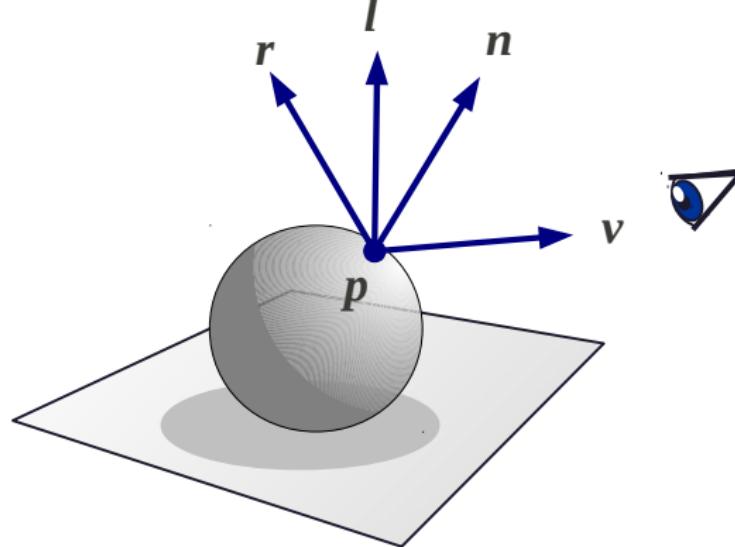
---

```
1: function RAYTRACING( punto o, vector u, entero n )
2:   if n > max then    // si se ha superado máximo nivel de recursión
3:     return (0,0,0)           // devolver radiancia nula
4:   O := primer objeto visible desde o en la dir. u    // (por ray-casting)
5:   if no existe ningun objeto visible then
6:     return radiancia de fondo correspondiente a u
7:   p := punto de O intersecado
8:   return EVALUAMILREC( O, p, -u, n )
```

---

# Evaluación del MIL

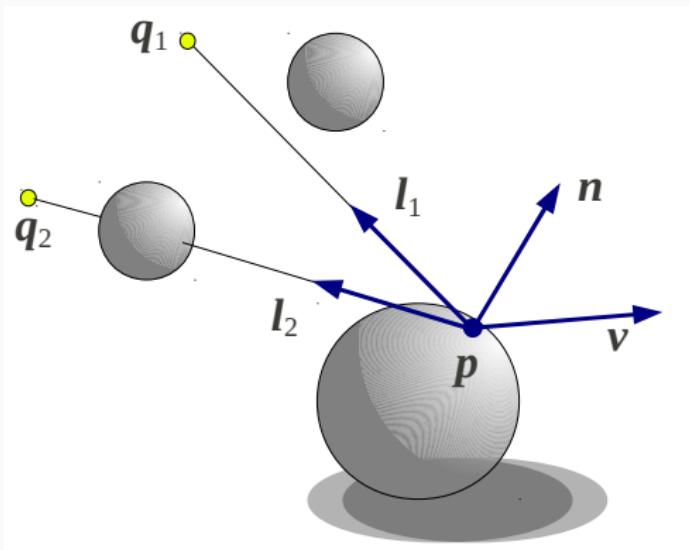
Una vez se conoce el punto  $p$ , se obtienen la normal  $n$ , y los parámetros del MIL, que es evaluado:



podemos considerar objetos curvos (esferas, cilindros, conos, etc..)

# Sombras arrojadas

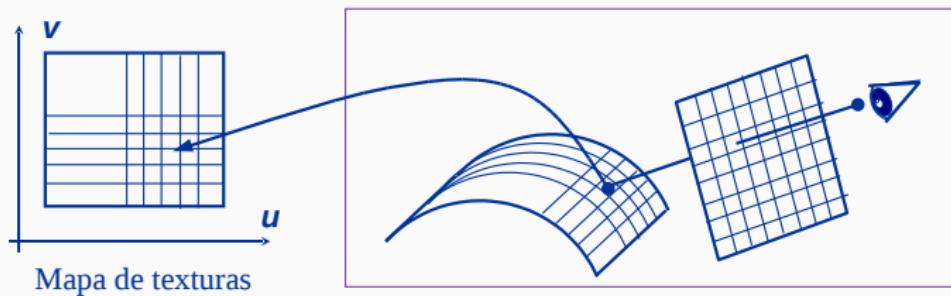
Este método permite incorporar sombras arrojadas, usando Ray-Casting para visibilidad. Se comprueba si un segmento de recta desde  $p$  hasta (o hacia) la fuente interseca algún objeto de la escena, es decir, se evalua  $V(p, q_i)$



# Detalles de las superficies

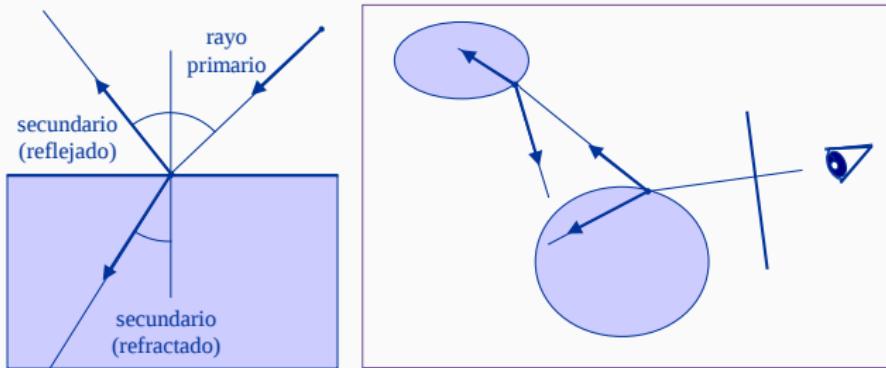
El objeto en el que está  $p$  puede tener asociadas texturas (o mapas de pert. de la normal)

- ▶ A partir de  $p$  se obtienen las coordenadas  $(u, v)$  en el espacio de la textura
- ▶ A partir de  $(u, v)$  se consulta la textura o texturas asociadas al objeto.



# Rayos secundarios y recursividad

También es posible tener en cuenta superficies perfectamente especulares y/o perfectamente transparentes. Esto se hace creando *rayos secundarios*, e invocando recursivamente al algoritmo.



Da lugar a un árbol de rayos asociado al árbol de llamadas recursivas.

# La función EVALUAMILREC

La función que evalua el MIL (EVALUAMILREC) llama recursivamente a la función de RAYTRACING,

---

```
1: function EVALUAMILREC( O, p, v, n )
2:   Obtener paráms. del material de O en p (n, C, ka, kd, ks, kps, MPS, MT, kt)
3:   Obtener parámetros de fuentes de luz (nL, li, Si)
4:   rad := ME(p) + AG(p)           // emisividad y comp. ambiente global
5:   for i := 0 to nL - 1 do          // para cada fuente de luz
6:     if la fuente i es visible desde p then // (que sea visible)
7:       rad := rad + Si · DIRECTA(p, v, li) // ilum. directa
8:     if kt > 0 and n < max then
9:       t := vector refractado respecto de v
10:      rad := rad + kt · MT · RAYTRACING( p, t, n + 1 ) // componente refractada
11:      if kps > 0 and n < max then
12:        r := vector reflejado respecto de v
13:        rad := rad + kps · MPS · RAYTRACING( p, r, n + 1 ) // componente reflejada
14:   return rad
```

---

# La función DIRECTA

Esta función evalua  $L_{\text{dir}}$  (las componentes ambiental, difusa y especular correspondientes a una fuente de luz).

---

```
1: function DIRECTA( p, v, l )
2:     rad :=  $k_a \cdot C$ 
3:     if  $k_d > 0$  then
4:         rad := rad +  $k_d \cdot C \cdot \max(0, \mathbf{n} \cdot \mathbf{l})$ 
5:     if  $k_s > 0$  then
6:         rad := rad +  $k_s \cdot d_i \cdot [\max(0, \mathbf{r} \cdot \mathbf{v})]^e$ 
7:     return rad
```

---

Informática Gráfica, curso 2022-23.

Teoría. Tema 5. Realismo en Rasterización. Ray-tracing.

Sección 2. Ray tracing

Subsección 2.2.

Intersecciones rayo-objeto y rayo-escena.

# Cálculo de intersecciones: rayo-objeto y rayo-escena

El algoritmo de Ray-Tracing emplea la mayor parte de su tiempo en:

- ▶ Evaluación de MIL avanzados: a los subprogramas que los evaluan se les llama *shaders*.
- ▶ Cálculo de intersecciones. Dado un punto  $\mathbf{o}$  y un vector unitario  $\mathbf{v}$  (en coordenadas de mundo, codificando una **semirrecta** o **rayo**), se trata de calcular el menor valor real  $t > 0$  tal que el punto  $\mathbf{p}(t) \equiv \mathbf{o} + t\mathbf{v}$  está en la frontera o superficie de algún objeto de la escena.

Hay dos algoritmos fundamentales:

- ▶ Intersección rayo-objeto: cada tipo de geometría posible tiene asociado un algoritmo.
- ▶ Intersección rayo-escena: podemos recorrer exhaustivamente los objetos, pero es  $O(n_o)$ . Se reducen los tiempos con *indexación espacial*, que es  $O(\log n_o)$ .

## Intersecciones rayo-objeto: método general

Calcular la intersección de un rayo  $(\mathbf{o}, \mathbf{v})$  con un objeto cuya geometría es  $O \subseteq \mathbb{R}$  consiste en obtener el mínimo valor de  $t > 0$  (si hay alguno) tal que  $\mathbf{o} + t\mathbf{v} \in \partial O$ . El objeto  $O$  puede estar caracterizado por estas dos funciones:

- ▶ Ecuación implícita: un campo escalar  $F$  tal que si  $\mathbf{p} \in \partial O$  entonces  $F(\mathbf{p}) = 0$ .
- ▶ Condiciones adicionales: un predicado o función lógica  $C$  tal que  $\mathbf{p} \in \partial O$  si y solo si  $F(\mathbf{p}) = 0$  y  $C(\mathbf{p})$ .

El algoritmo requiere:

1. Calcular el conjunto  $S = \{t_0, t_1, \dots, t_{n-1}\}$  con las raíces de la ecuación  $F(\mathbf{o} + t\mathbf{v}) = 0$ .
2. Eliminar de  $S$  los  $t_i$  que no cumplen  $C(\mathbf{o} + t_i\mathbf{v})$ .
3. Si  $S \neq \emptyset$  la solución es  $t = \min(S)$ . Si  $S = \emptyset$  no hay inters.

# Intersecciones rayo-objeto: métodos de solución

Hay dos tipos de algoritmos para obtener la menor raíz  $t$ :

- ▶ Directos:  $t$  se obtiene con una fórmula explícita.

Por ejemplo: si  $\partial O$  es un subconjunto de una **superficie cuádrica**, entonces

$$F(\mathbf{o} + t\mathbf{v}) = 0 \iff \exists a, b, c \in \mathbb{R} \quad \text{t.q: } at^2 + bt + c = 0$$

- ▶ Iterativos:  $t$  se obtiene por sucesivas aproximaciones

Por ejemplo: se puede usar si  $F$  es la ***signed distance function*** (SDF) de  $O$ , es decir, si

$$F(\mathbf{p}) = s \cdot \left( \min_{\mathbf{q} \in \partial O} \|\mathbf{p} - \mathbf{q}\| \right) \quad \text{donde: } s \equiv \begin{cases} -1 & \text{si } \mathbf{p} \in O \\ +1 & \text{si } \mathbf{p} \notin O \end{cases}$$

Si no se dispone de SDF existen alternativas (Bisección, Newton, cotas de distancia, etc...).

# Indexación espacial para métodos directos.

Para calcular las intersecciones rayo-escena podemos usar los algoritmos directos de intersección rayo-objeto para las distintos tipos de objetos que forman la escena:

- ▶ Las escenas usualmente consisten en mallas de triángulos.
- ▶ Solución de *fuerza bruta*: comprobar la intersección con cada triángulo de la escena (tiempo en  $O(n_t)$ ).
- ▶ La **indexación espacial** se basa en descartar partes de la escena si el rayo no interseca un volumen englobante  $V$  de todos los triángulos de esa parte de la escena.
- ▶ La intersección rayo- $V$  es muy rápida de calcular.
- ▶ Se hace jerárquicamente (recursivamente), y se obtiene un árbol de volúmenes englobantes, con conjuntos de triángulos en los nodos terminales. Se obtiene tiempo en  $O(\log n_t)$ .

# El algoritmo *Sphere Tracing*

Un método iterativo, llamado **Sphere Tracing**, usa las SDFs  $F_i$  de los objetos. El algoritmo se basa en avanzar un punto a lo largo del rayo. A cada paso se avanza la mínima distancia del punto a los objetos de la escena, hasta que diverge o converge a un punto de intersección:

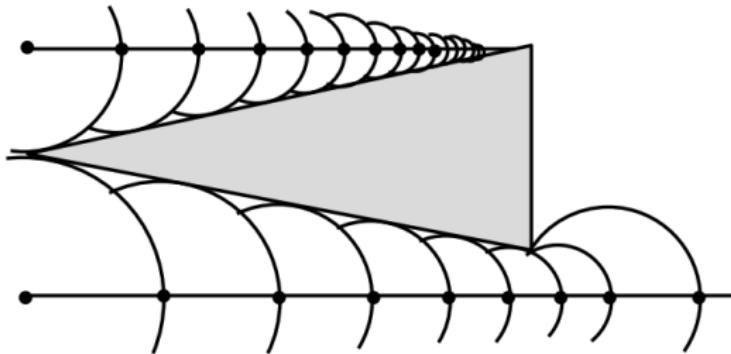


Figura obtenida del artículo *Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces*, de John C. Hart (1995). [PDF aquí.](#)

## Intersecciones rayo-escena con *sphere tracing*

Si se disponen de SDFs  $F_0, \dots, F_{n-1}$  de los objetos  $O_0, \dots, O_{n-1}$  se puede usar este algoritmo (donde  $0 \lesssim \epsilon$  es la tolerancia):

---

```
1: function INTERSECCIONSDF( o, v,  $O_0, \dots, O_{n-1}$  )           //  $\|\mathbf{v}\| = 1$ 
2:   p = o // mejor punto de intersección encontrado hasta ahora
3:    $d = 0$  //  $d \equiv$  máxima distancia que se puede avanzar desde p
4:   repeat
5:     p = p +  $d\mathbf{v}$  // avanzamos a lo largo del rayo
6:      $d = \min \{F_0(\mathbf{p}), \dots, F_{n-1}(\mathbf{p})\}$  // actualizamos  $d$  en p
7:   until  $\|d\| \leq \epsilon$  (o hasta un número máximo de iteraciones)
8:   if  $\|d\| \leq \epsilon$  then // si p está ya muy cerca de la superficie
9:     Hay intersección con  $O_i$  en p (con  $i$  tal que  $d \equiv F_i(\mathbf{p})$ )
10:  else
11:    No hay intersección con ningún objeto
```

---

Hay numerosos ejemplos de esta técnica en  Shadertoy

Informática Gráfica, curso 2022-23.

Teoría. Tema 5. Realismo en Rasterización. Ray-tracing.

Sección 2. Ray tracing

Subsección 2.3.

Problemas: Cálculo de intersecciones.

# Problema: intersección rayo-disco (1/2)

## Problema 5.1.

Supongamos que un *rayo* (una semirecta en 3D) tiene como origen o extremo el punto cuyas coordenadas del mundo es la tupla **o**, y como vector de dirección la tupla **d** (la suponemos normalizada).

Además sabemos que un *disco* de radio  $r$  tiene como centro el punto de coordenadas de mundo **c** y está en el plano perpendicular al vector **n**.

Con estos datos de entrada, diseña un algoritmo para calcular si hay intersección entre el rayo y el disco.

(ten en cuenta las indicaciones que hay en la siguiente transparencia).

# Problema: intersección rayo-disco (2/2)

## Problema 5.1. (continuación)

Ten en cuenta que habrá intersección si y solo si se cumplen cada una de estas dos condiciones:

1. El rayo interseca con el plano que contiene al triángulo, es decir, existe  $t > 0$  tal que el punto  $\mathbf{p}_t \equiv \mathbf{o} + t\mathbf{d}$  está en dicho plano. Equivale a decir que el vector  $\mathbf{p}_t - \mathbf{v}_0$  es perpendicular a la normal al plano  $\mathbf{n}$ .
2. El punto  $\mathbf{p}_t$  citado arriba está dentro del disco, es decir, su distancia a  $\mathbf{c}$  es inferior al radio.

# Problema: intersección rayo-esfera

## Problema 5.2.

Diseña un algoritmo para calcular la primera intersección entre un rayo (con origen en  $\mathbf{o}$  y vector  $\mathbf{d}$ , normalizado) y una esfera de radio unidad y centro en el origen, si hay alguna.

Ten en cuenta que un punto cualquiera  $\mathbf{p}$  está en esfera si y solo si el módulo de  $\mathbf{p}$  es la unidad, es decir, si y solo si  $F(\mathbf{p}) = 0$ , donde  $F$  es el campo escalar definido así:

$$F(\mathbf{p}) \equiv \mathbf{p} \cdot \mathbf{p} - 1$$

Describe como podría usarse ese mismo algoritmo para calcular la intersección con una esfera con centro y radio arbitrarios (este problema puede reducirse al anterior si el rayo se traslada a un espacio de coordenadas donde la esfera tiene centro en el origen y radio unidad).

# Problema: intersección rayo-cilindro y rayo-cono

## Problema 5.3.

Describe como podemos definir el campo escalar cuyos ceros son los puntos en un cilindro con altura unidad y radio unidad (sin considerar los discos que forman la base ni la tapa).

Usando esa definición diseña el algoritmo para calcular la intersección rayo-cilindro.

Describe asimismo el campo escalar y el algoritmo correspondientes a un cono de altura unidad y radio de la base unidad (sin considerar el disco de la base).

Informática Gráfica, curso 2022-23.

Teoría. Tema 5. Realismo en Rasterización. Ray-tracing.

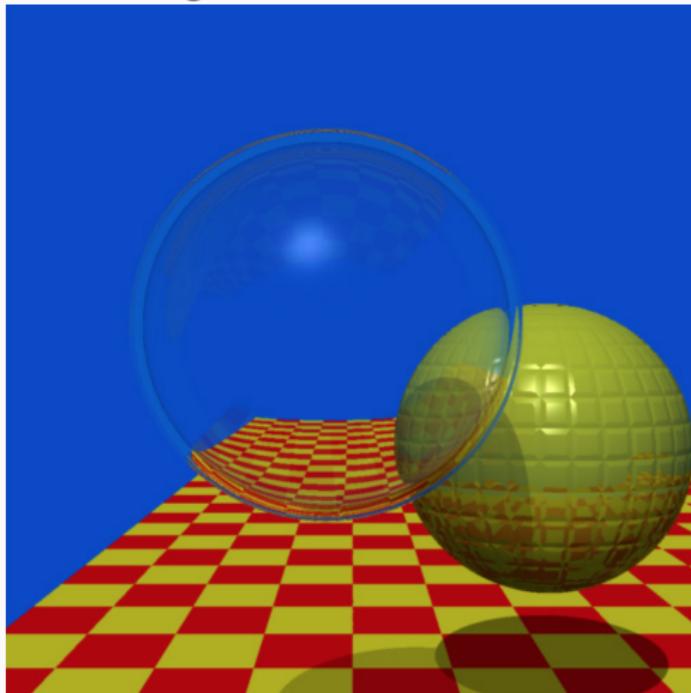
Sección 2. Ray tracing

Subsección 2.4.

Ejemplos.

# Ejemplos: primeras imágenes

Una de las primeras imágenes creadas con esta técnica (en 1980)



Turner Whitted (1980) [An Improved Illumination Model for Shaded Display \(CACM\)](#)  
Entrevista a T. Whitted, vídeo [sitio web de nVidia](#)

# Ejemplos: reflexión y refracción

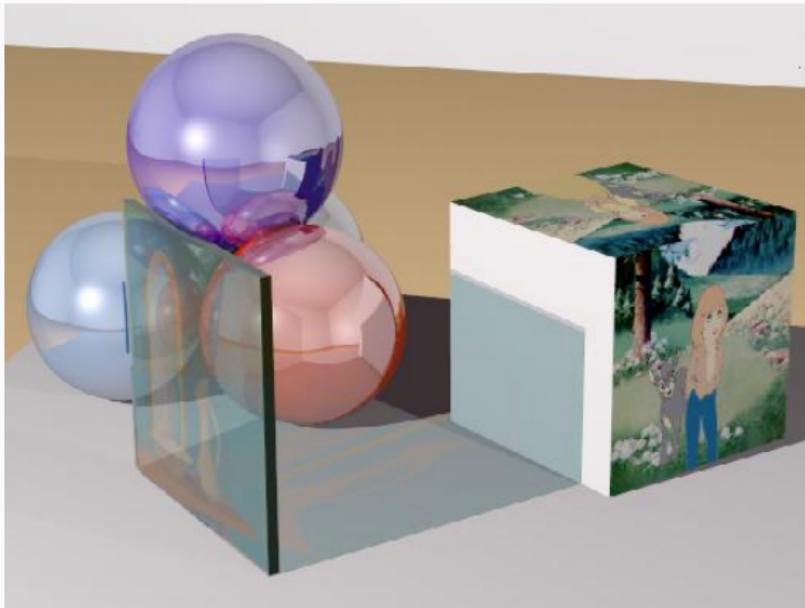
Incluye: texturas, sombras arrojadas, reflexiones, *bump-mapping*.



Universidad de Cornell (1998)  Computer Graphics programme: Reflection and Transparency.

# Ejemplos: reflexiones especulares, sombras arrojadas.

Incluye: reflexiones, texturas, sombras arrojadas con texturas.



Universidad de Cornell (1998)  Computer Graphics programme: Reflection and Transparency.

# Ejemplos: escenas complejas, indexación espacial.

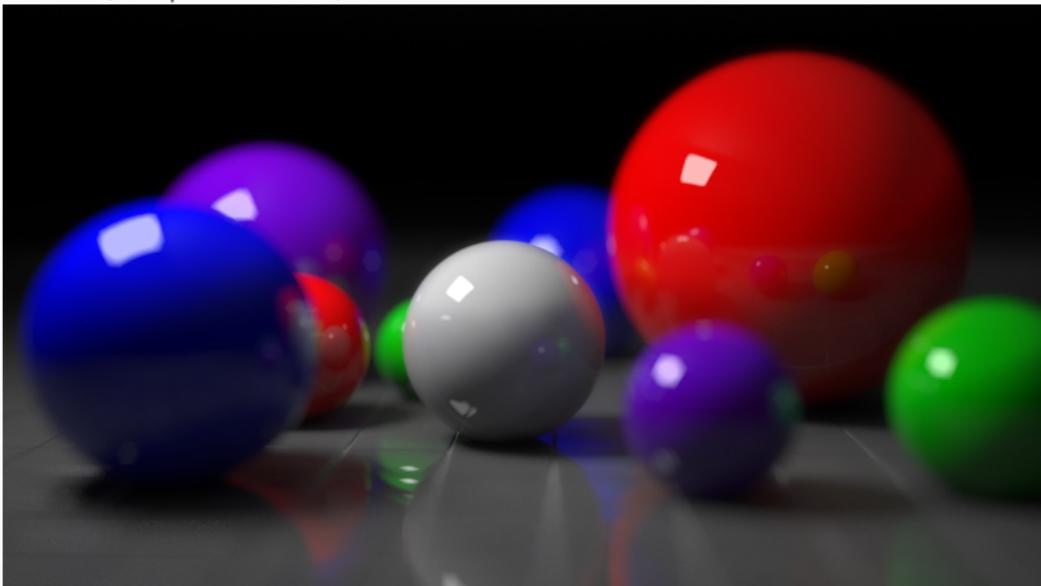
Escena compleja: múltiples objetos complejos. Indexación espacial.



Autor: Gilles Tran Sitio web Oyonale. Public Domain

## Ejemplos: *Distributed Ray-Tracing*

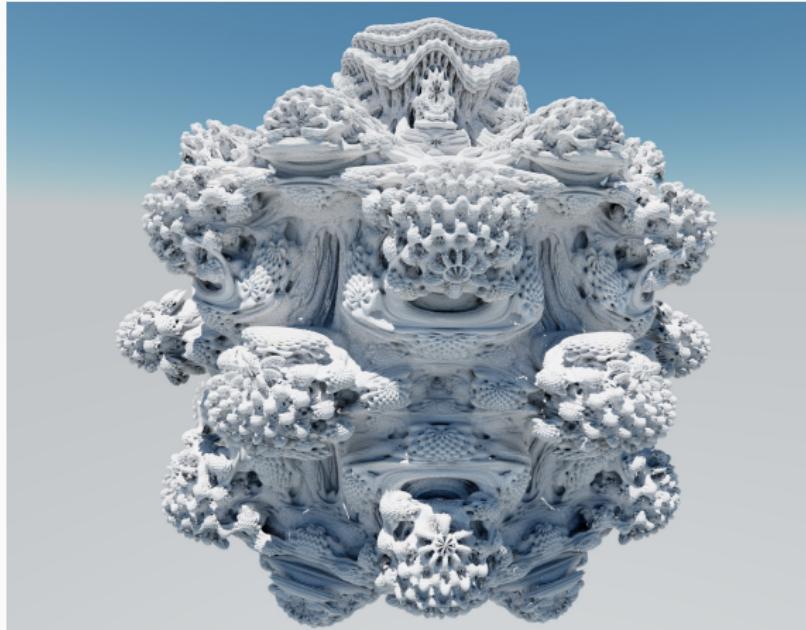
Se usa *Distributed Ray-Tracing* (Cook, 1984): se simula la profundidad de campo (*Depth of Field*) y las penumbras generadas por luces de extensas (no puntuales):



By [Mimigu](#) at [English Wikipedia](#): Ray Tracing (Graphics). CC BY 3.0.

## Ejemplos: *Path-tracing*, fractales.

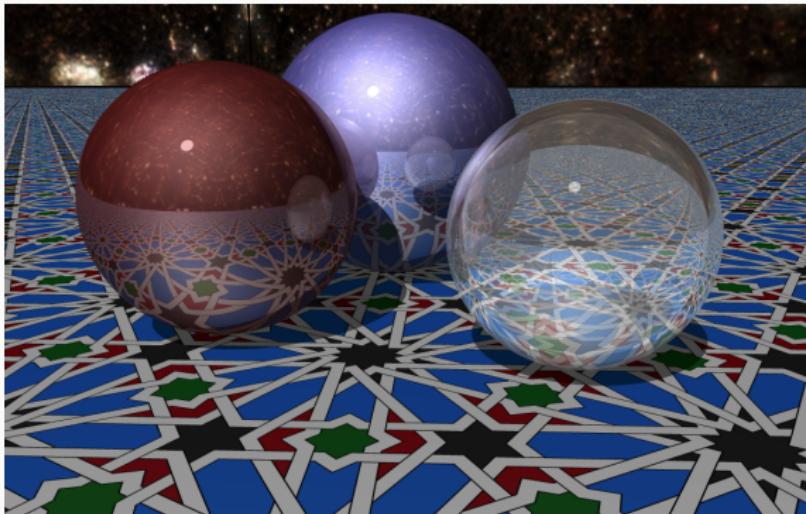
Se usa *Path-tracing* (Kajiya, 1986, *Global Illumination*), e intersecciones con un fractal (con métodos numéricos iterativos).



Por: Mikael Hvidtfeldt Christensen  Blog sobre arte generativo.  
 <https://www.flickr.com/photos/syntopia/>).

# Ejemplos: Ray-tracing sencillo en GPU

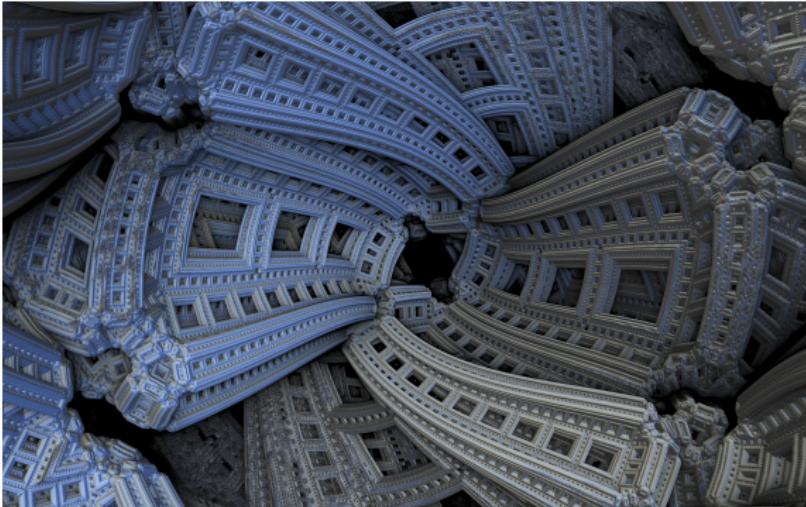
Ray-tracing en tiempo real en la GPU usando un *fragment shader*.  
Incluye: reflexiones, refracciones, texturas procedurales (grupos cristalográficos del plano: pmm6)



Animación disponible on-line en Shadertoy:  <https://www.shadertoy.com/view/4sdfzn>

## Ejemplos: uso de SDF en GPU

Se usan objetos definidos por su SDF (*Signed Distance Function*) para calcular intersecciones iterativamente. Se hace en la GPU en un *fragment shader*.



*Menger Journey* por **Mikael Hvidtfeldt Christensen**. Animación on-line en [Shadertoy](#).

Fin de la presentación.