Informática Gráfica:
Teoría. Tema 5. Realismo en Rasterización. Ray-tracing.

Carlos Ureña

Dpt. Lenguajes y Sistemas Informáticos
ETSI Informática y de Telecomunicación
Universidad de Granada

2018-19

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

Sección 1
Técnicas realistas en rasterización

1.1. Mipmaps.
1.2. Perturbación de la normal
1.3. Sombras arrojadas
1.4. Superficies transparentes

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 un 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:

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

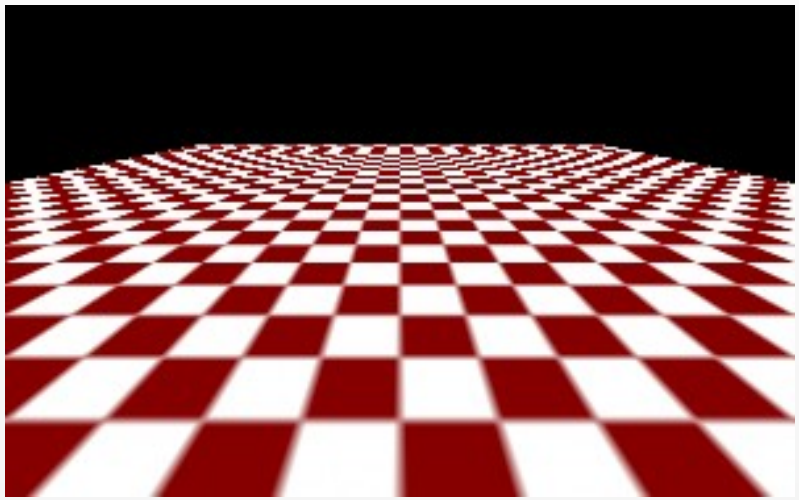
Acceso a los *mipmaps*

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

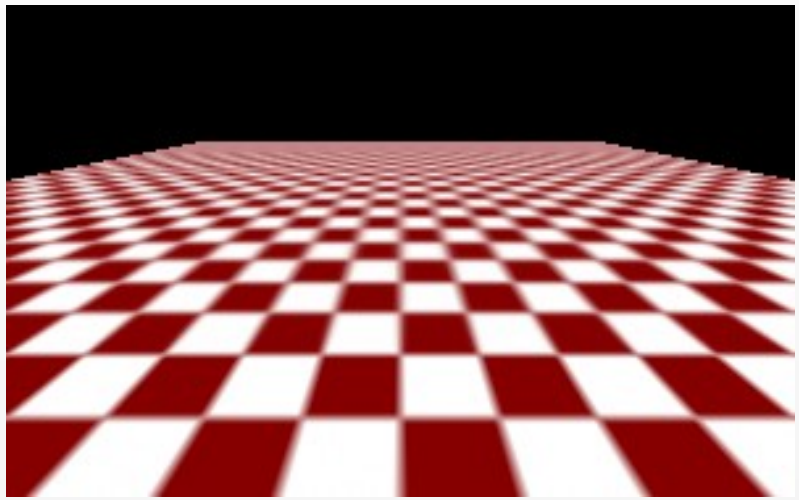
- Se usará la textura M_i, donde i crece linealmente con el logaritmo de la distancia d entre $\mathbf{p}$ y el observador (menos resolución a mayor distancia).
- Esta solución puede presentar cambios bruscos de la resolución al pasar bruscamente de una resolución a otra en pixels cercanos. La solución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

Subsección 1.2
Perturbación de la normal

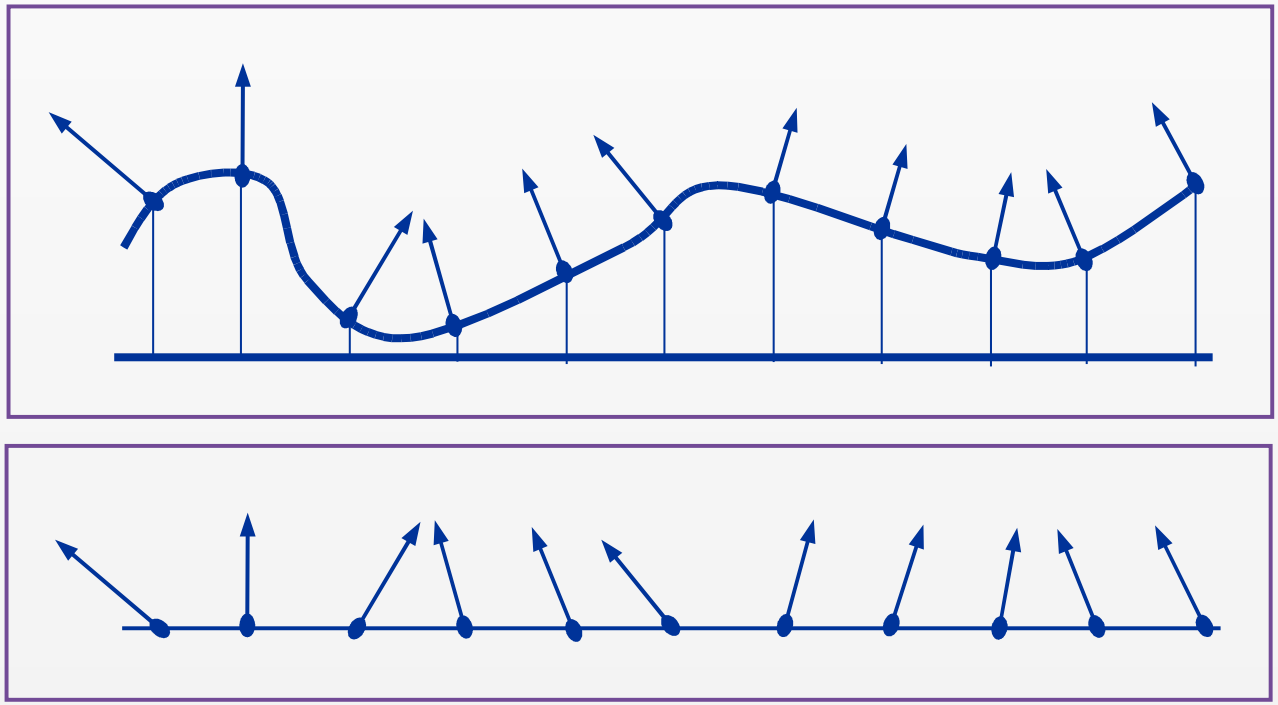
Rugosidades a pequeña escala

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):

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

- 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:

- Δ 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 $\boldsymbol{f}_p$ de las coordenadas de textura, es decir, si las coord. de textura de un punto $\boldsymbol{q}$ son (u,v), entonces:

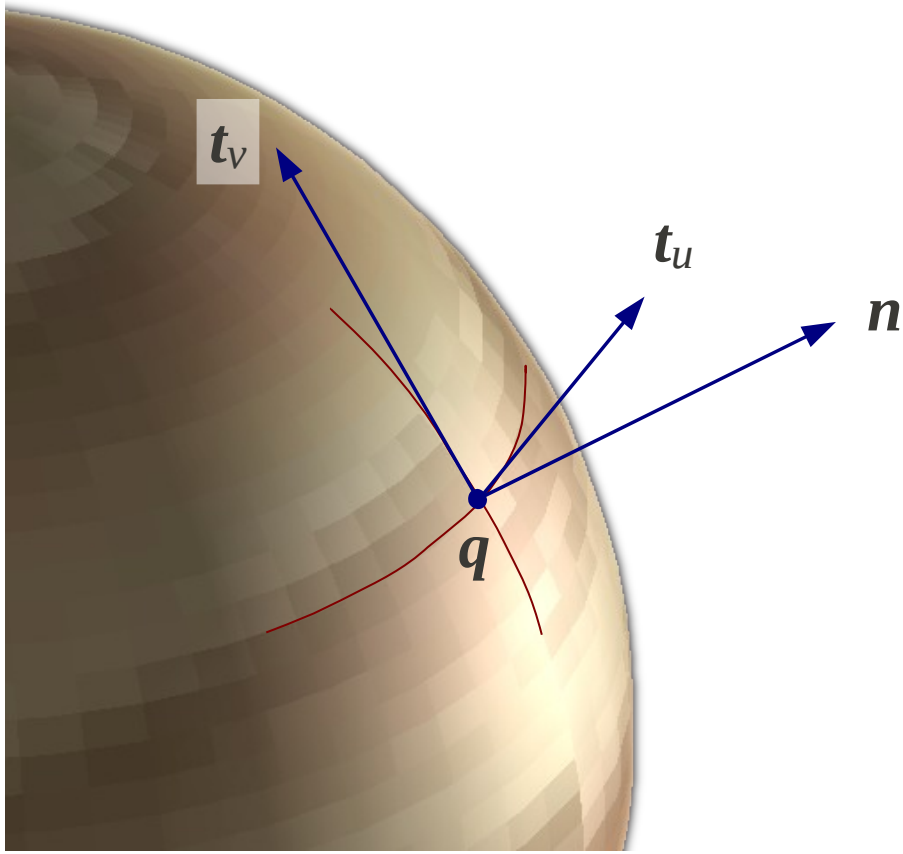
$$\boldsymbol{q} = \boldsymbol{f}_p(u,v)$$

para calcular la normal modificada es necesario conocer las derivadas parciales de $\boldsymbol{f}_p$ (dos vectores $\boldsymbol{t}_u$ y $\boldsymbol{t}_v$)

$$\boldsymbol{t}_u = \frac{\partial \boldsymbol{f}_p(u,v)}{\partial u} \qquad \boldsymbol{t}_v = \frac{\partial \boldsymbol{f}_p(u,v)}{\partial v}$$

Las tangentes y la normal

A los vectores $\boldsymbol{t}_u$ y $\boldsymbol{t}_v$ se les suele llamar *tangente* y *bitangente*. Ambos definen un plano tangente, perpendicular a la normal.



Cálculo de los vectores tangentes modificados

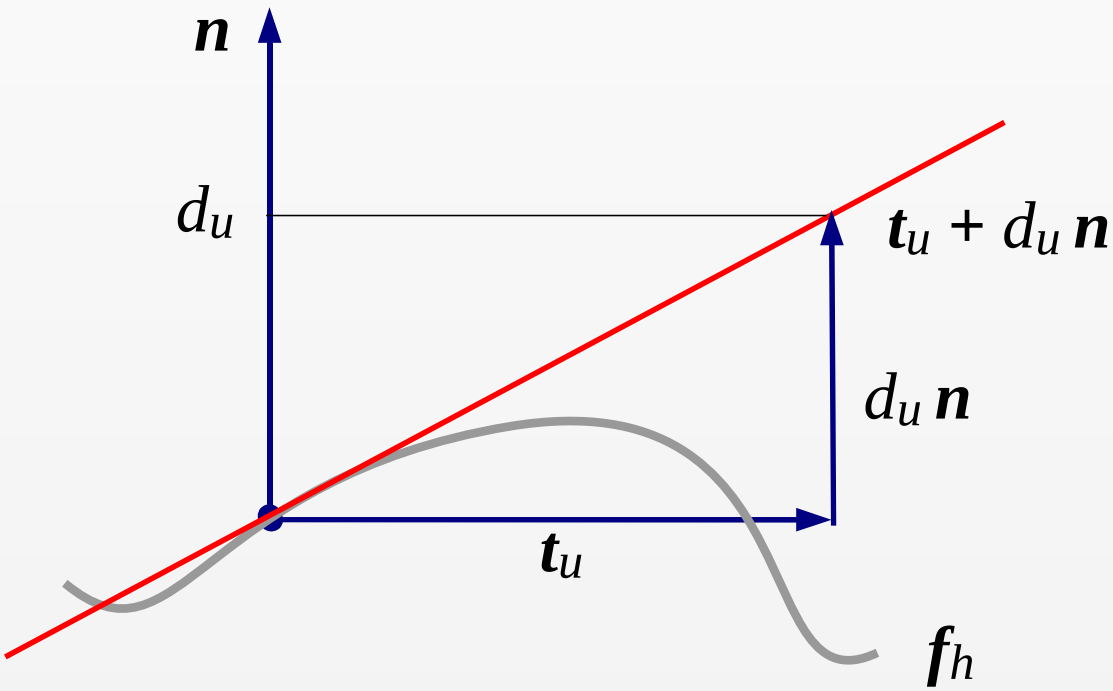
Estos vectores son tangentes a la superficie del objeto, ya que la normal original $\boldsymbol{n}$ es colineal con $\boldsymbol{t}_u \times \boldsymbol{t}_v$ Existen varias alternativas para obtenerlos:

- 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 $\boldsymbol{t}'_u$ y $\boldsymbol{t}'_v$ a la superficie rugosa son:

$$\boldsymbol{t}'_u = \boldsymbol{t}_u + d_u \boldsymbol{n} \qquad \boldsymbol{t}'_v = \boldsymbol{t}_v + d_v \boldsymbol{n}$$

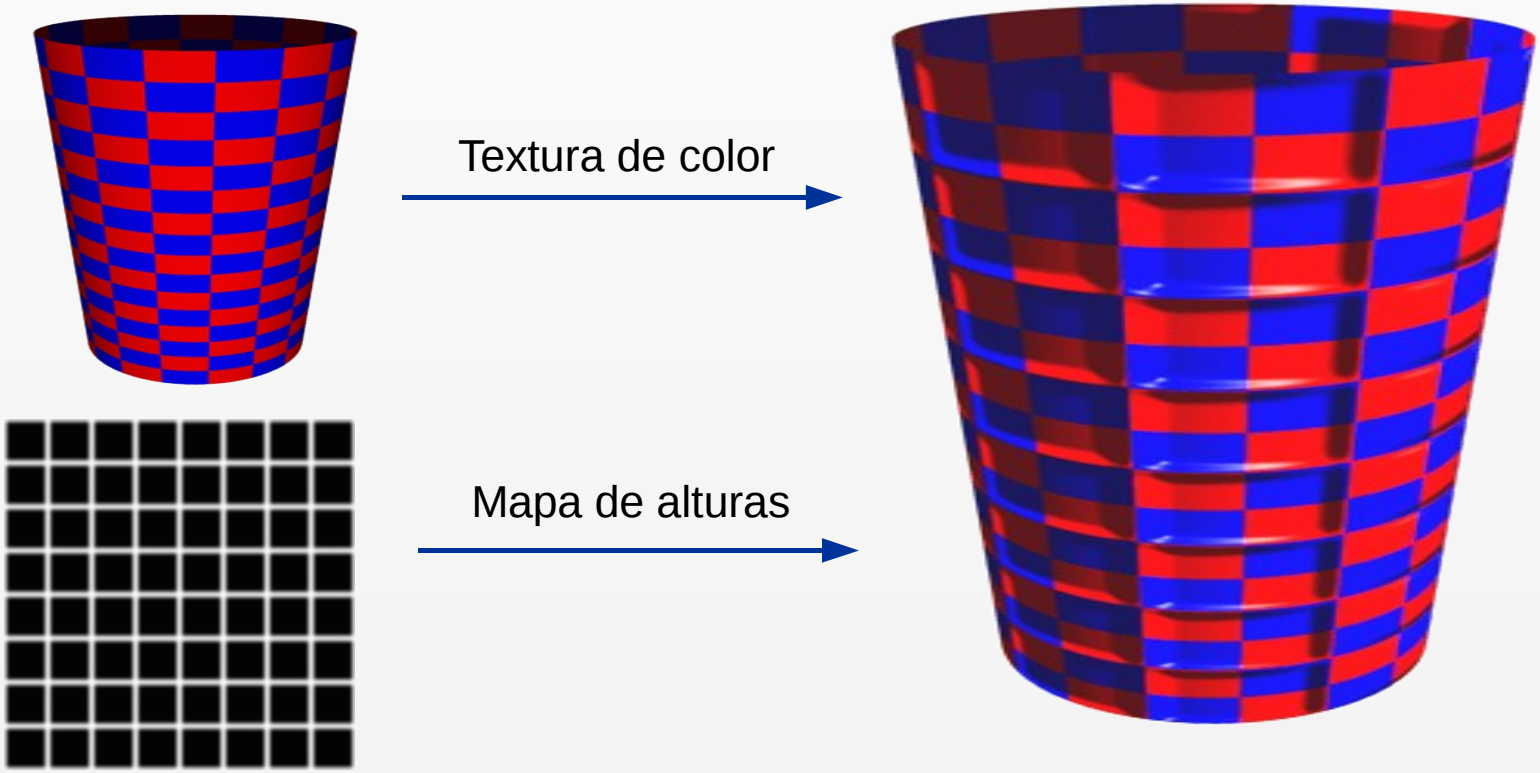


la normal modificada $\boldsymbol{n}'$ es perpendicular a estos dos vectores, por tanto se calcula usando su producto vectorial (y normalizando)

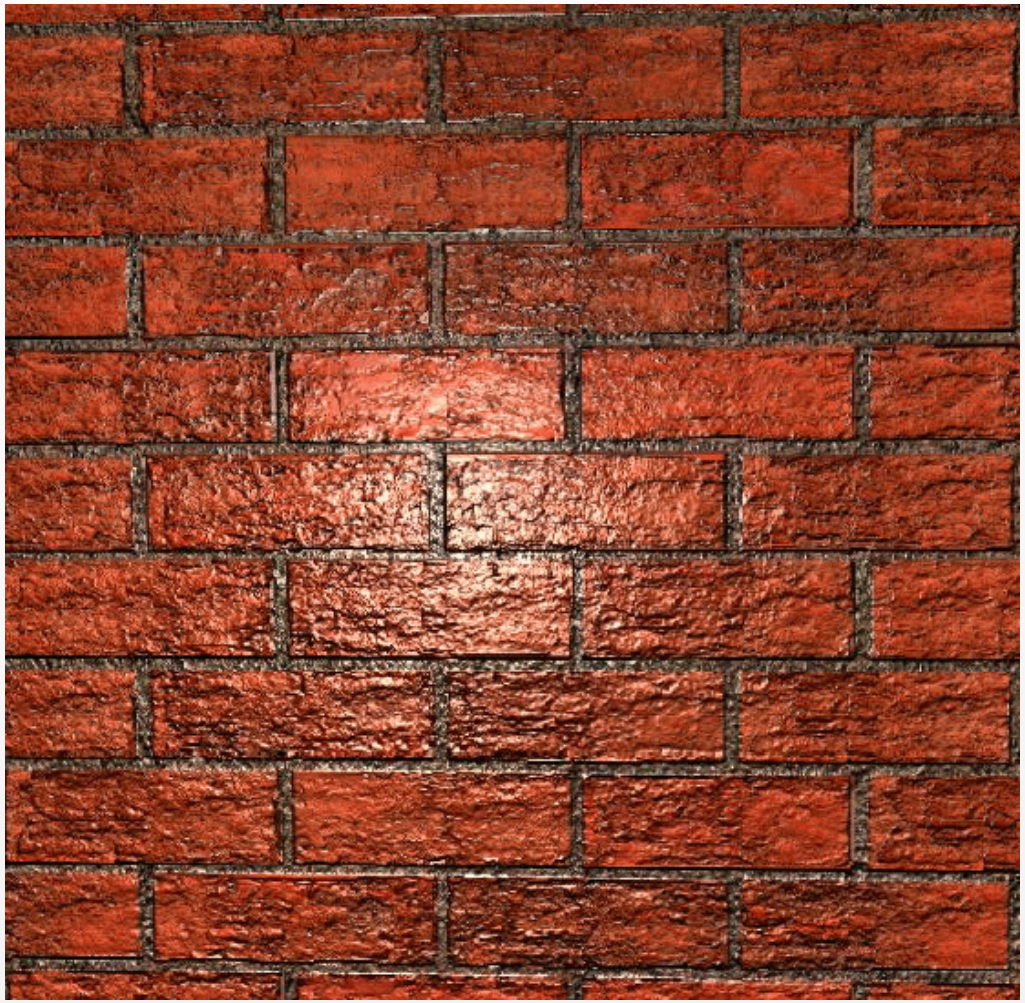
$$\boldsymbol{n}' = \frac{\boldsymbol{n}''}{\|\boldsymbol{n}''\|} \qquad \text{donde:} \quad \boldsymbol{n}'' = \boldsymbol{t}'_u \times \boldsymbol{t}'_v$$

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)



Subsección 1.3
Sombras arrojadas

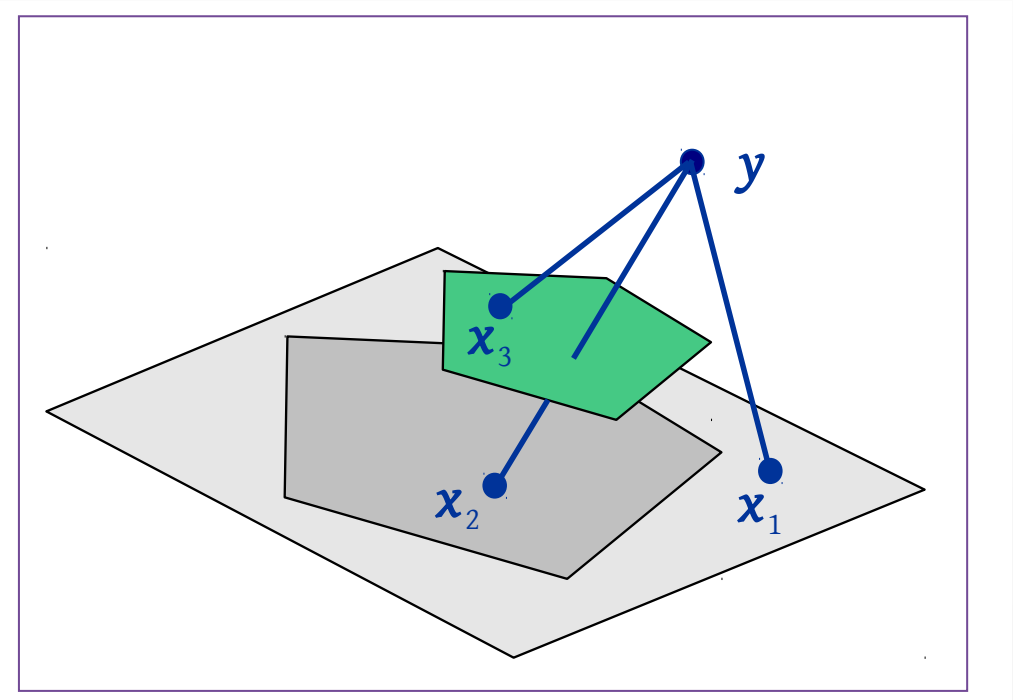
Sombras arrojadas y el MIL

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 polñigonos 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:



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

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 imágen: se obtiene el primer punto visible en el centro de cada pixel de un plano de visión asociado a una fuente de luz.

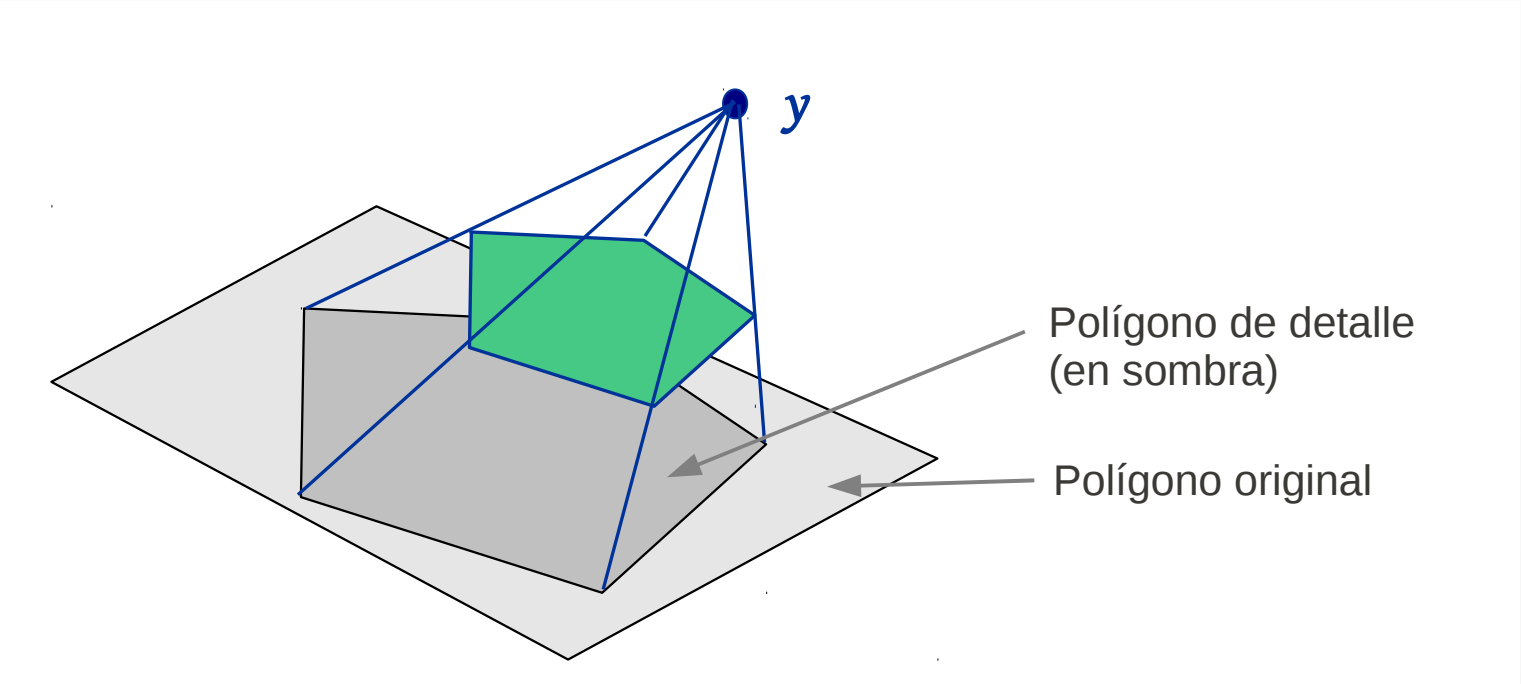
El papel del observador lo juega la fuente de luz. Puede ser posicional (observador a distancia finita) o direccional (observador a distancia infinita: proyección ortogonal).

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)



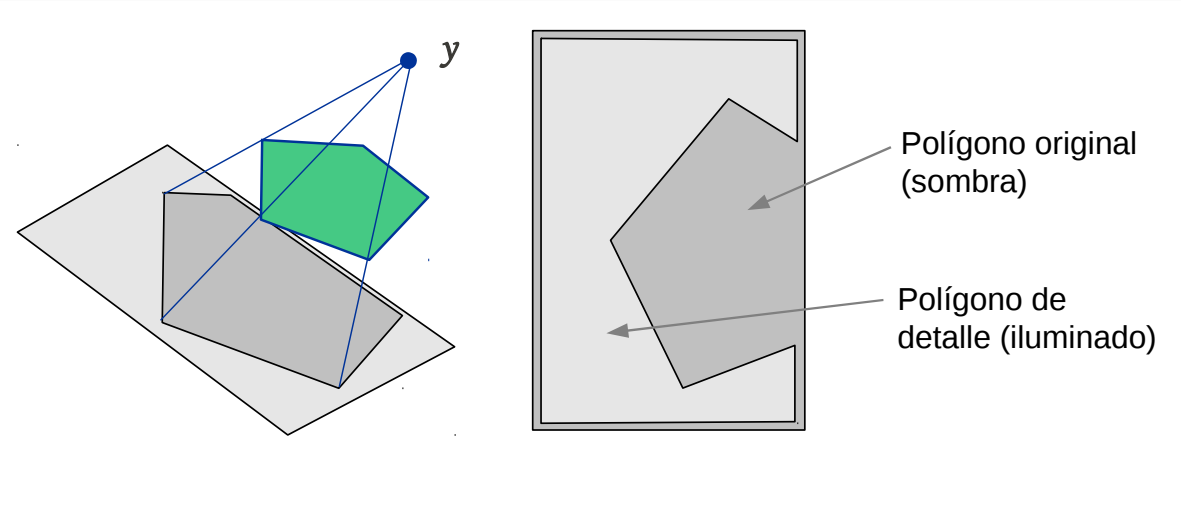
- tiene complejidad cuadrática con el número de polígonos
- se puede usar solo para un único polígono receptor y unos pocos que arrojas sombras.

Algoritmo de *Weiler-Atherton-Greenberg*

Otros algoritmos de sombras arrojadas (más eficientes) están basados en algoritmos de eiminación 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)

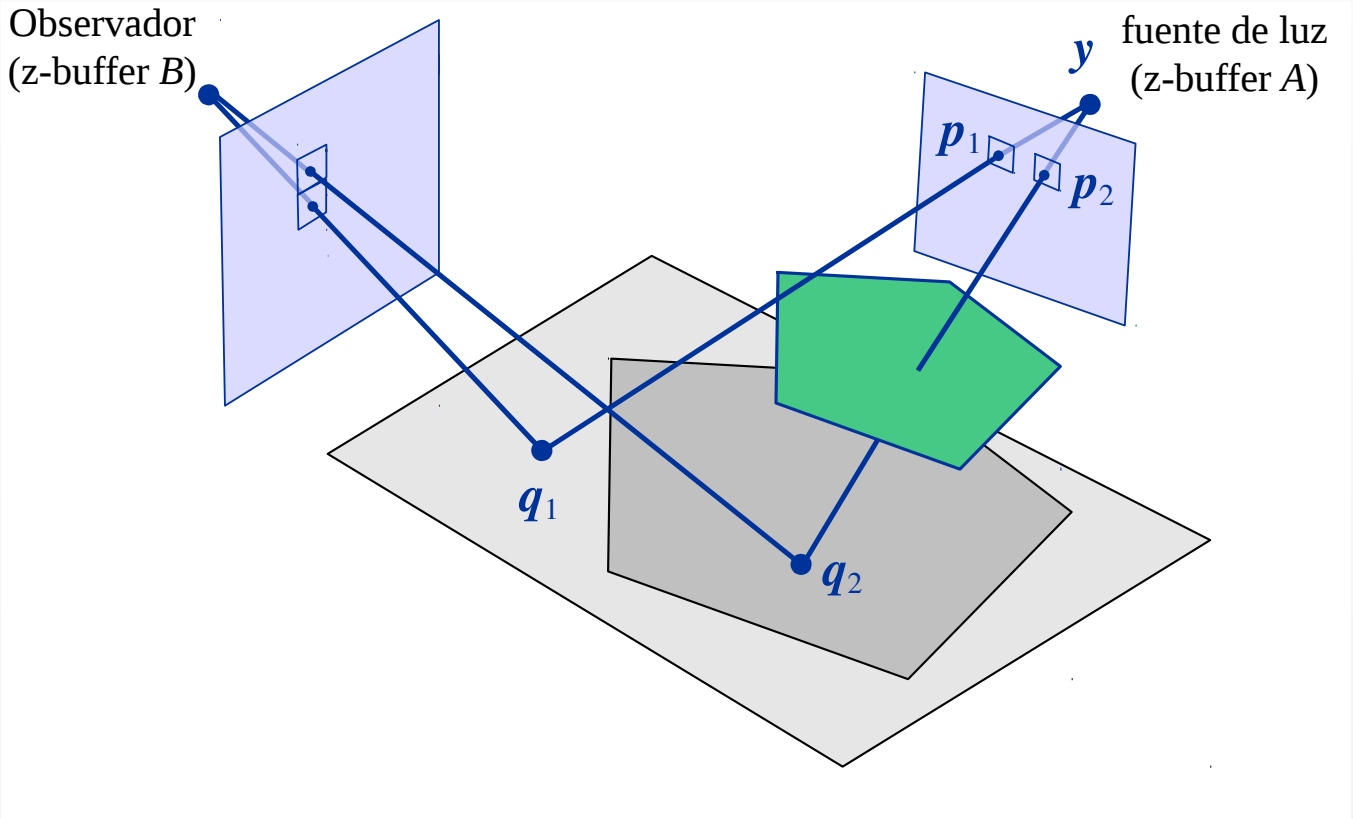


En general, los algoritmos con precisión de objetos para sombras son:

- 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:

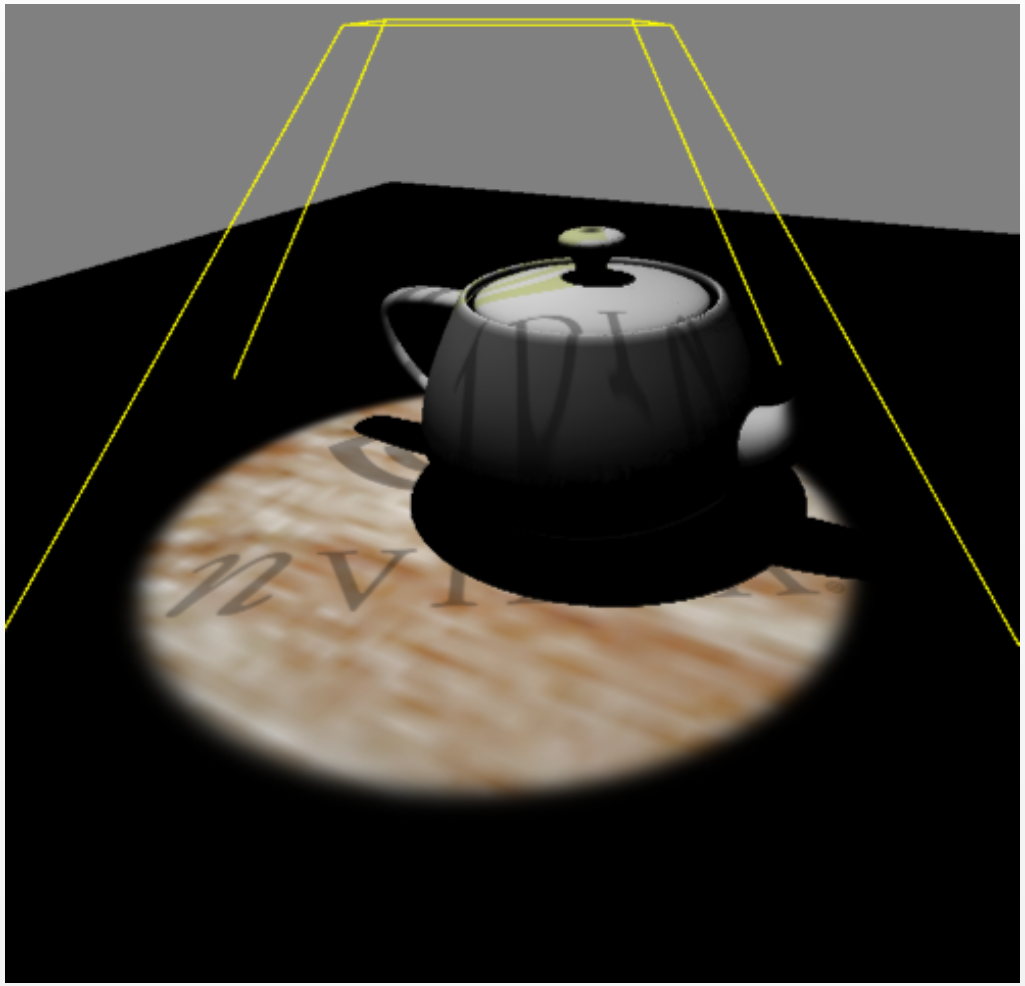


Z-buffer para sombras arrojadas (2)

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

Ejemplo de Z-buffer para sombras

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



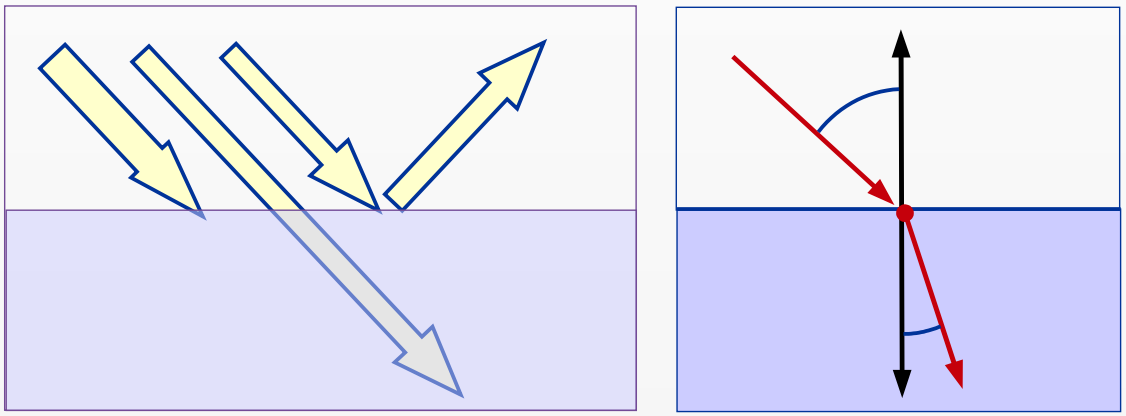
Errores de Z-buffer para sombras

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

Subsección 1.4
Superficies transparentes

Materiales transparentes

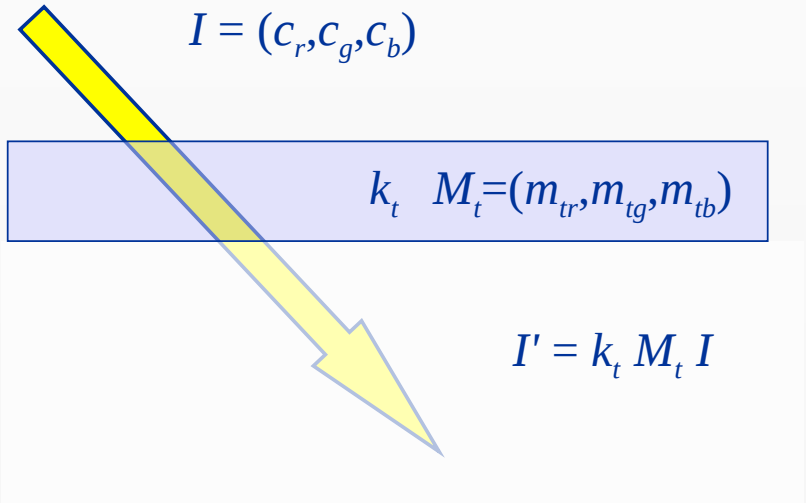
Hay objetos sólidos o líquidos que permiten pasar (además de reflejar o absorber) algunos fotones de la luz que los alcanza. Su estructura atómica permite a los rayos de luz viajar en línea recta.



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:



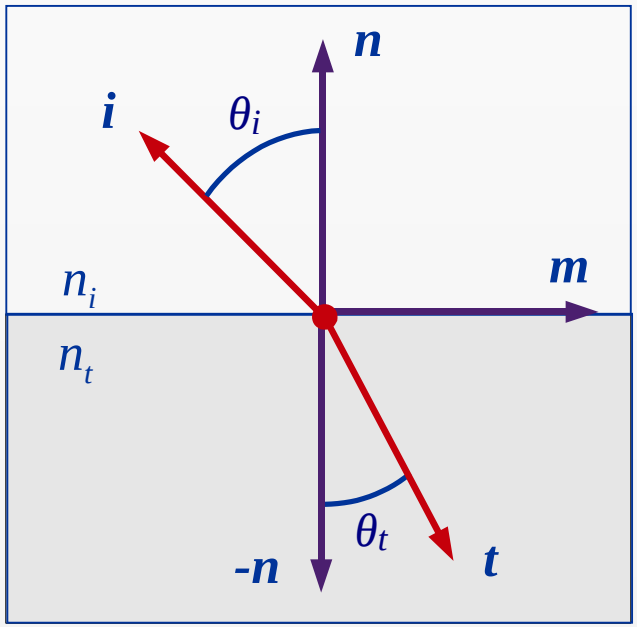
- La fracción global de luz refractada es k_t, que está entre 0 y 1
- En cada longitud de onda se refracta una fracción distinta, en RGB estas fracciones son un color $M_t = (m_{tr}, m_{tg}, m_{tb})$

Por tanto, estos materiales están caracterizados por k_t y M_t

Cambio de dirección en la refracción

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

$$n_i \sin(\theta_i) \quad = \quad n_t \sin(\theta_t)$$



para obtener $\boldsymbol{t}$:

1. $\boldsymbol{m} = \boldsymbol{n} \times (\boldsymbol{n} \times \boldsymbol{i})$
2. $\cos(\theta_i) = \boldsymbol{i} \cdot \boldsymbol{n}$
3. $\sin(\theta_i) = \sqrt{1 - \cos^2(\theta_i)}$
4. $\sin(\theta_t) = \sin(\theta_i)\, n_i \,/\, n_t$
5. $\cos(\theta_t) = \sqrt{1 - \sin^2(\theta_t)}$
6. $\boldsymbol{t} = \sin(\theta_t)\, \boldsymbol{m} - \cos(\theta_t)\, \boldsymbol{n}$

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 es to 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 $\boldsymbol{t} = -\boldsymbol{i}$
- 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)

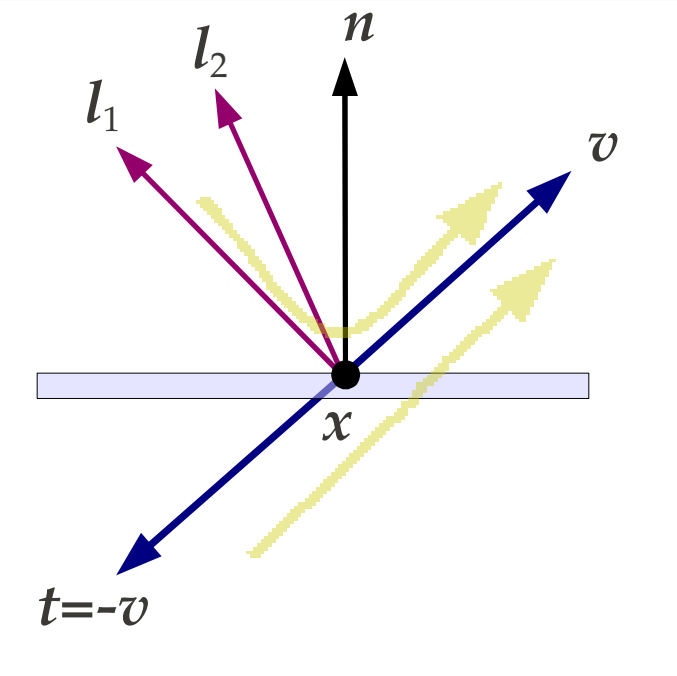
```
Inicializar Z-buffer (Z) e Imagen (I)

Para cada poligono opaco P
   Rasterizar P, actualizando Z e I

Para cada poligono transparente Q
   k_t := fraccion de luz refractada de Q
   M_t := color transparente de Q
   Para cada pixel (x,y) ocupado por Q
       Si Q es visible en (x,y)
          I[x,y] := k_t M_t I[x,y]
```

Combinación de reflexión y refracción

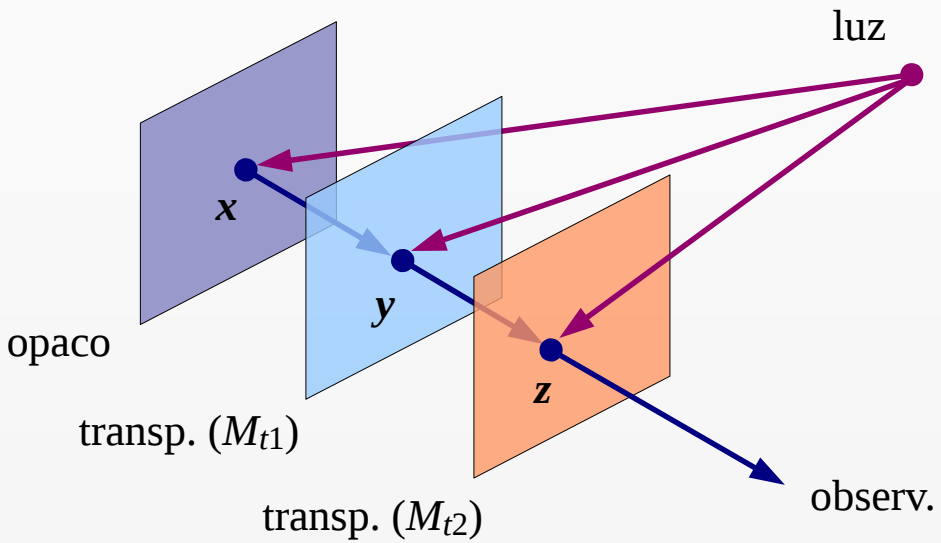
En la superficie entre una lámina de material transparente y el espacio (vacío) entre objetos puede también reflejarse la luz:



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

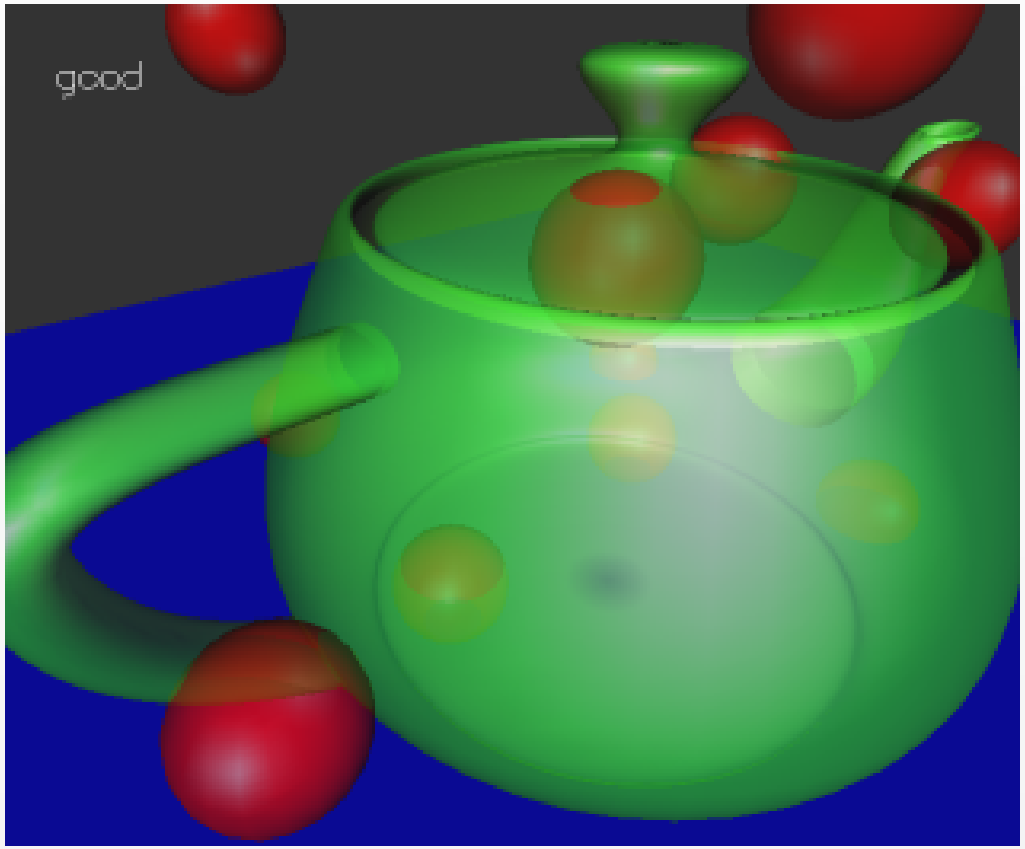
Los polígonos transp. deben dibujarse en orden de Z:

```
Inicializar Z-buffer (Z) e Imagen (I)

Para cada poligono opaco P
   Rasterizar P, actualizando Z e I

Para cada poligono transparente Q (en orden de Z)
   k_t := fraccion de luz refractada de Q
   M_t := color transperente de Q
   Para cada pixel (x,y) ocupado por Q
       Si Q es visible en (x,y)
          I_m := resultado de evaluar MIL
          I[x,y] := I_m + k_t M_t I[x,y]
```

Ejemplo de materiales transparentes



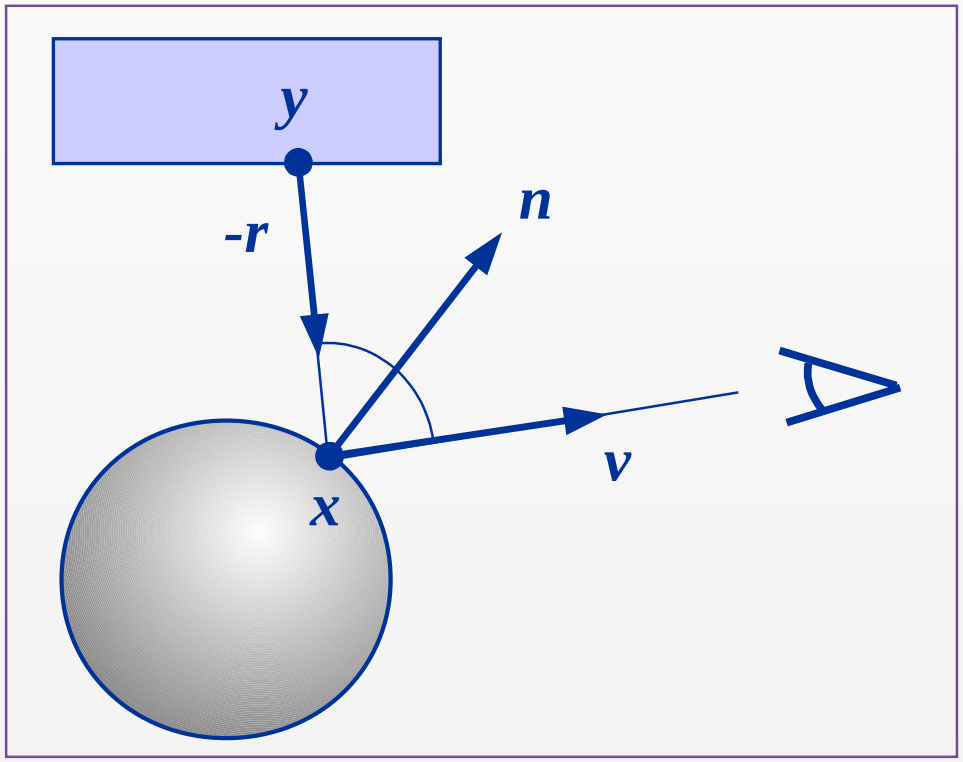
Reflexión especular de la luz

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.

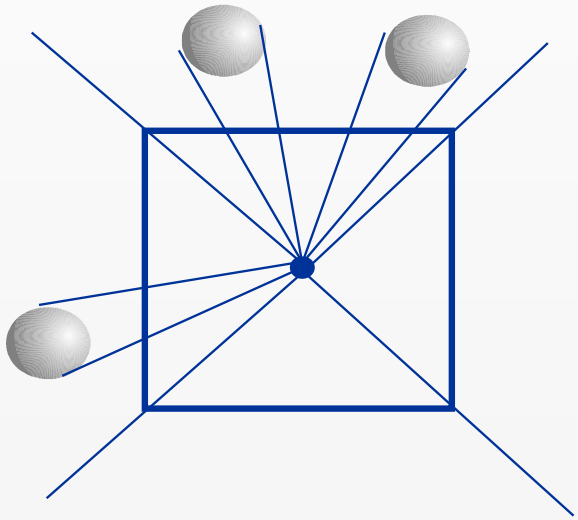
Reflexión especular de la luz

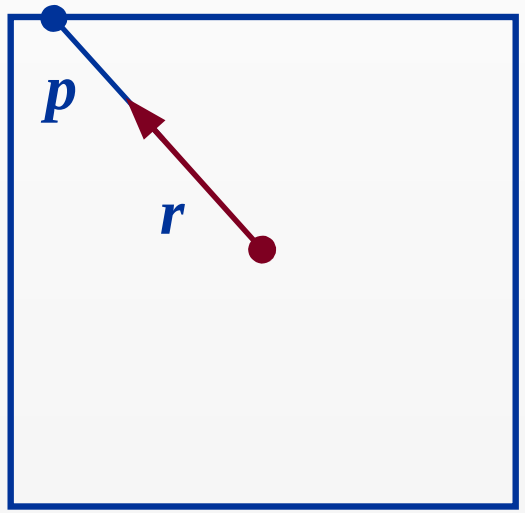
Si la esfera es perfectamente reflectante, el color de x visto desde v es igual al color de y visto desde la direccion $-r$ (el vector reflejado r, cambiado de signo).



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.

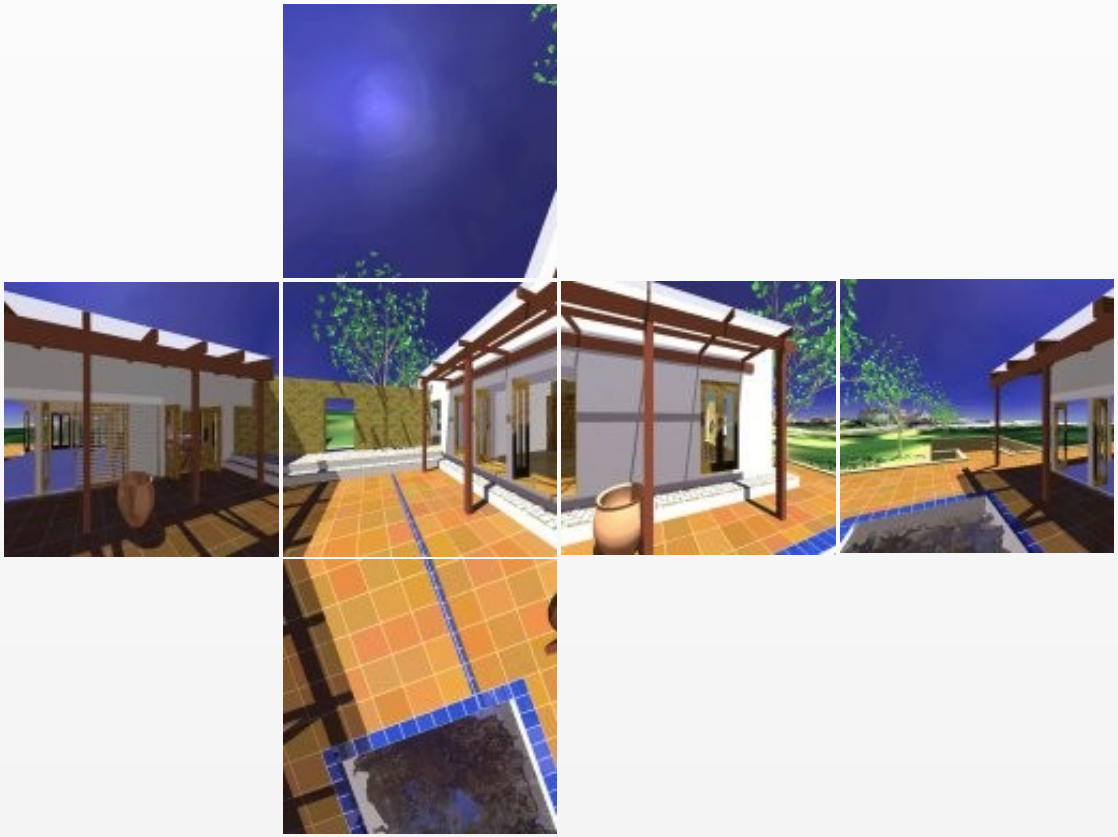




En tiempo de rendering, el vector r se proyecta sobre la cara que corresponda (se calcula p), y se obtienen el color RGB del texel que contiene a p.

Texturas para mapas de entorno tipo caja

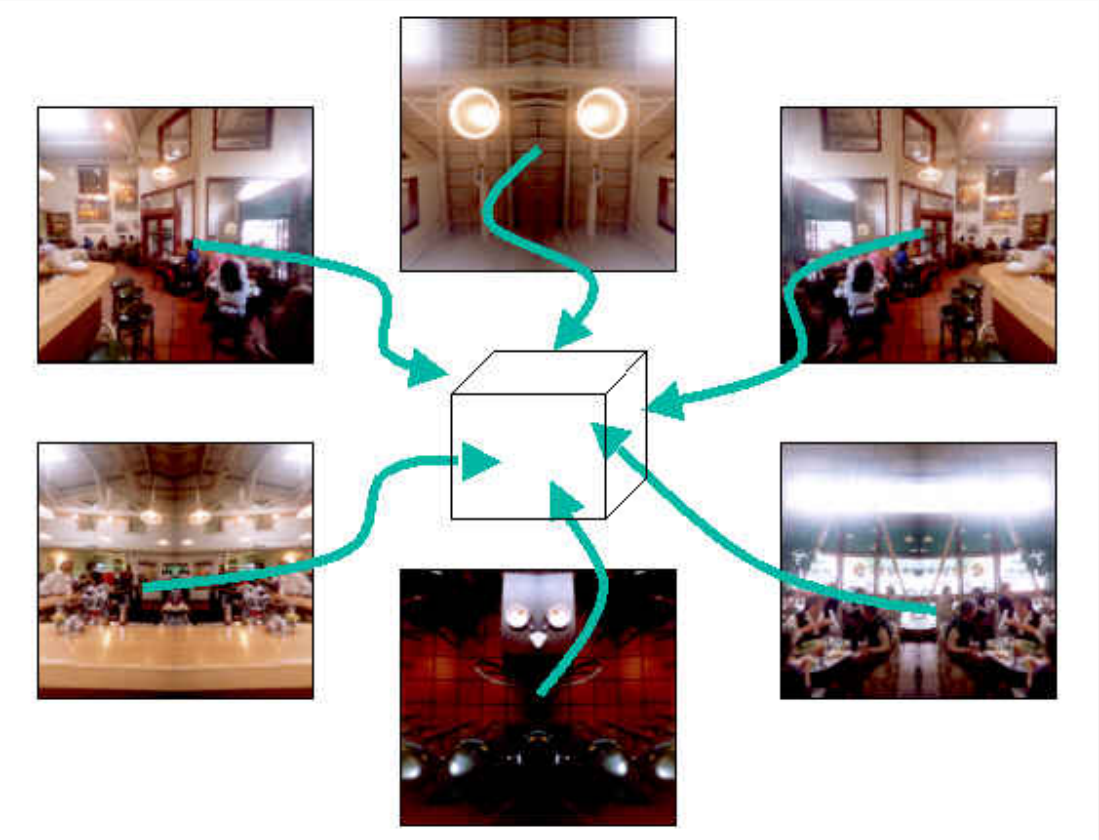
Ejemplo de 6 texturas para mapas de entorno



http://developer.nvidia.com/object/cube_map_ogl_tutorial.html

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

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 $\boldsymbol{v}$ 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:

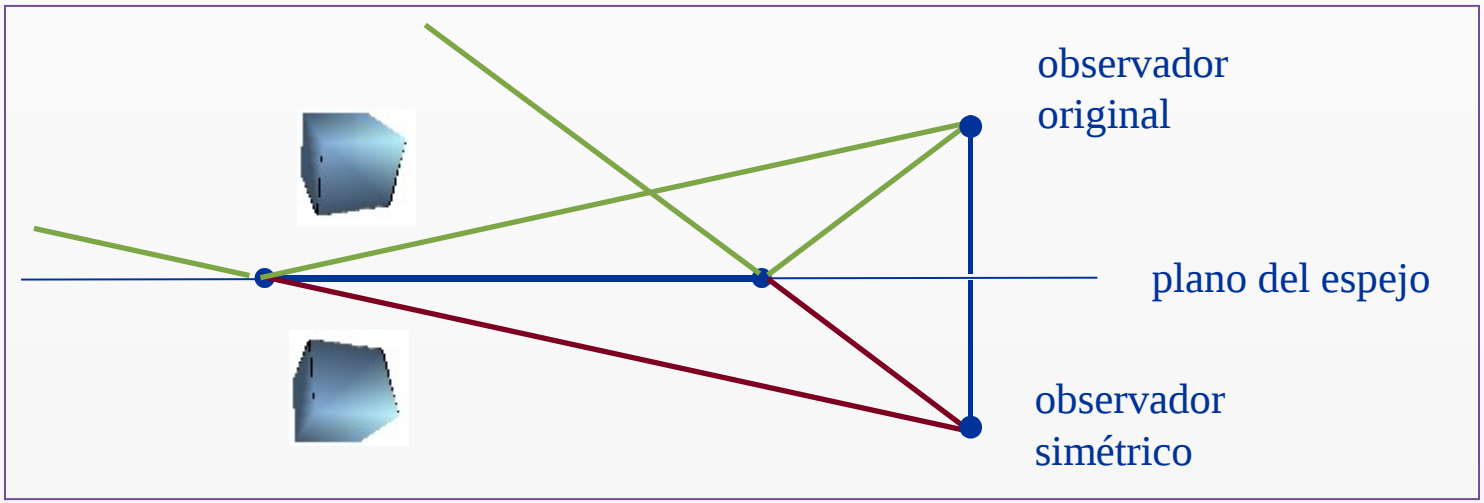
Mapa de entorno

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

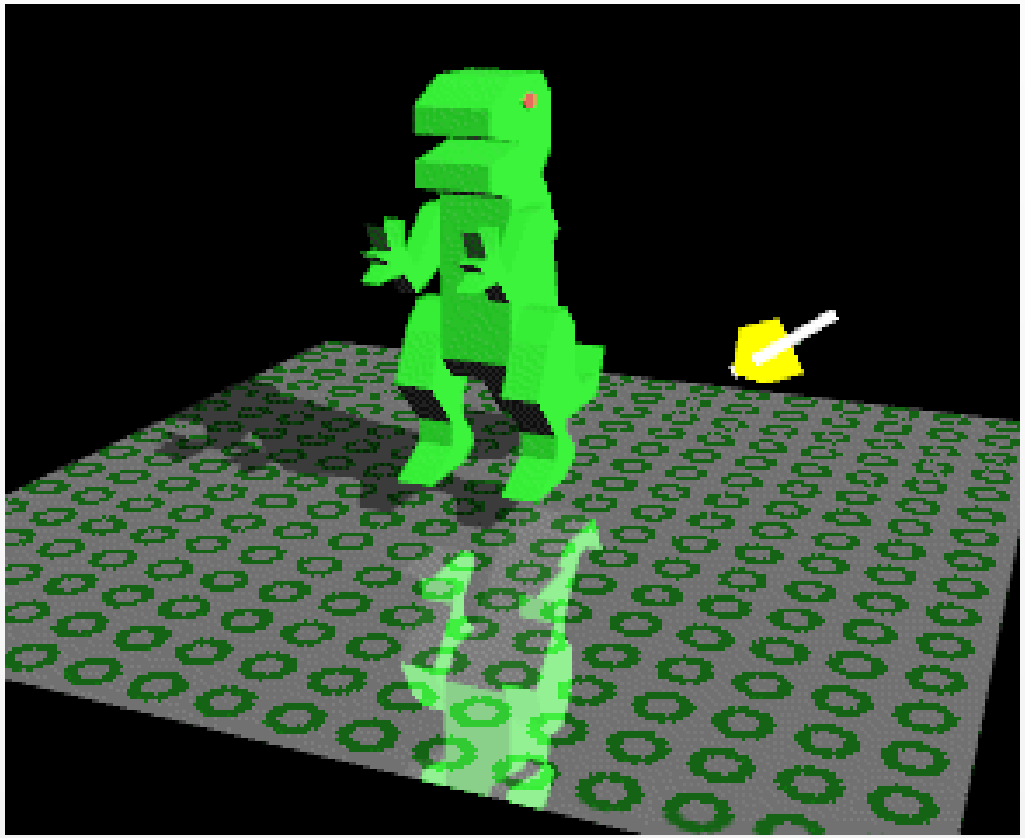
Reflexión en espejos planos

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

Subsección 2.1
El algoritmo de Ray-Tracing

Introducción

Hemos vistos bastantes efectos:

- Superficies difusas y especulares
- Texturas y mapas de perturbacion de la normal
- Sombras arrojadas
- Superficies transparentes
- Superficies especulares perfectas

Considerarlos todos lleva a software que es bastante complicado de implementar Además los tiempos de sintesis de imágenes pueden hacerse bastante altos

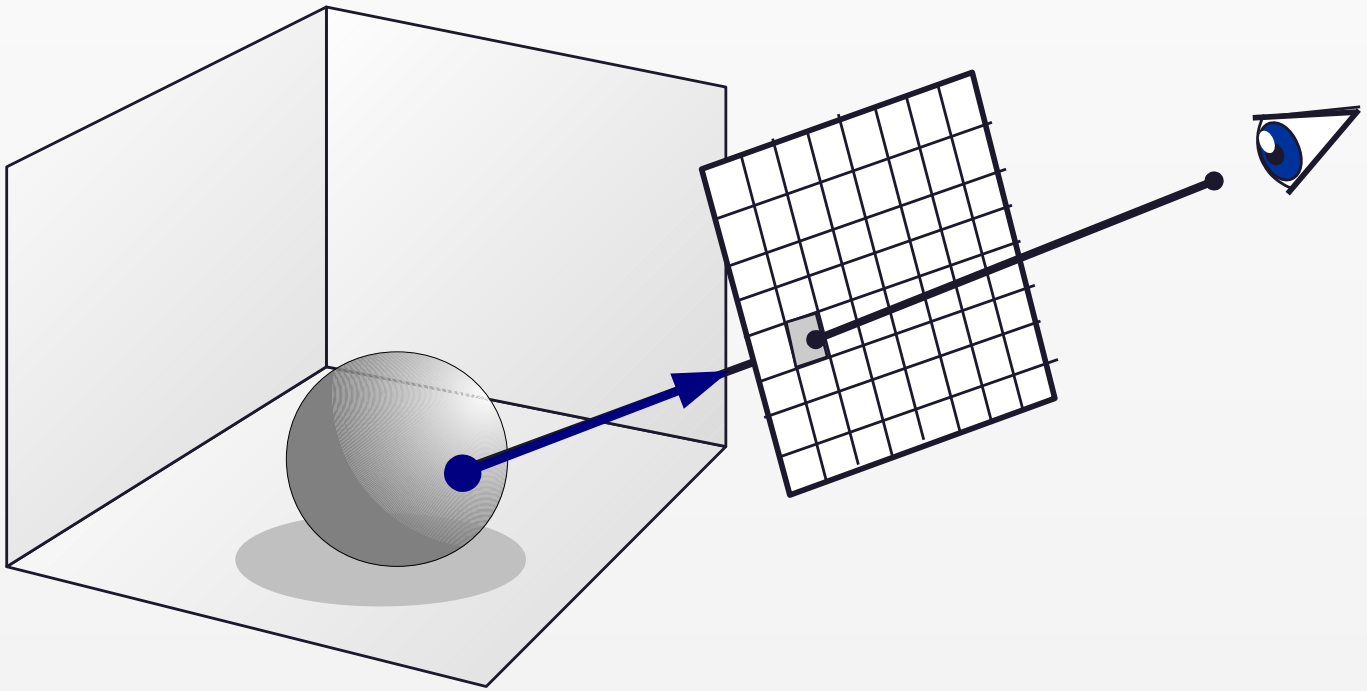
La técnica de *Ray-Tracing*

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.

Generación de Rayos-primarios

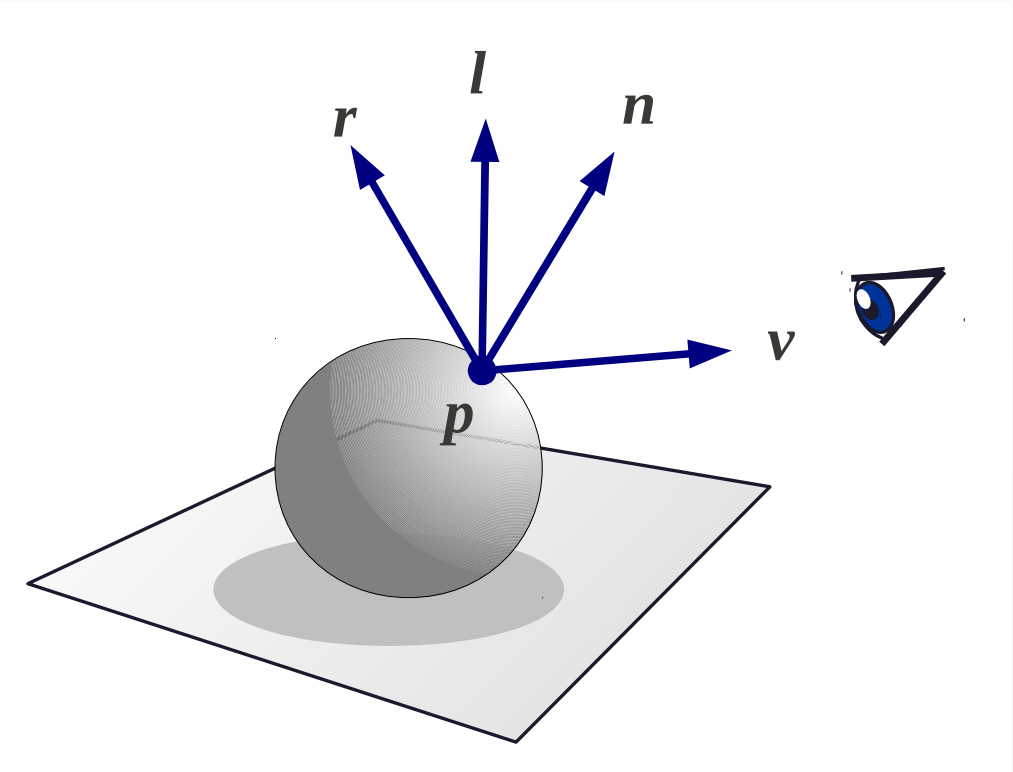
Los pixels se procesan secuencialmente, en cada uno se crea un rayo (llamado *rayo primario* o *rayo de cámara*) y se determina el primer objeto visible por Ray-Casting:



Subsección 2.2
Evaluación del MIL

Evaluación del MIL

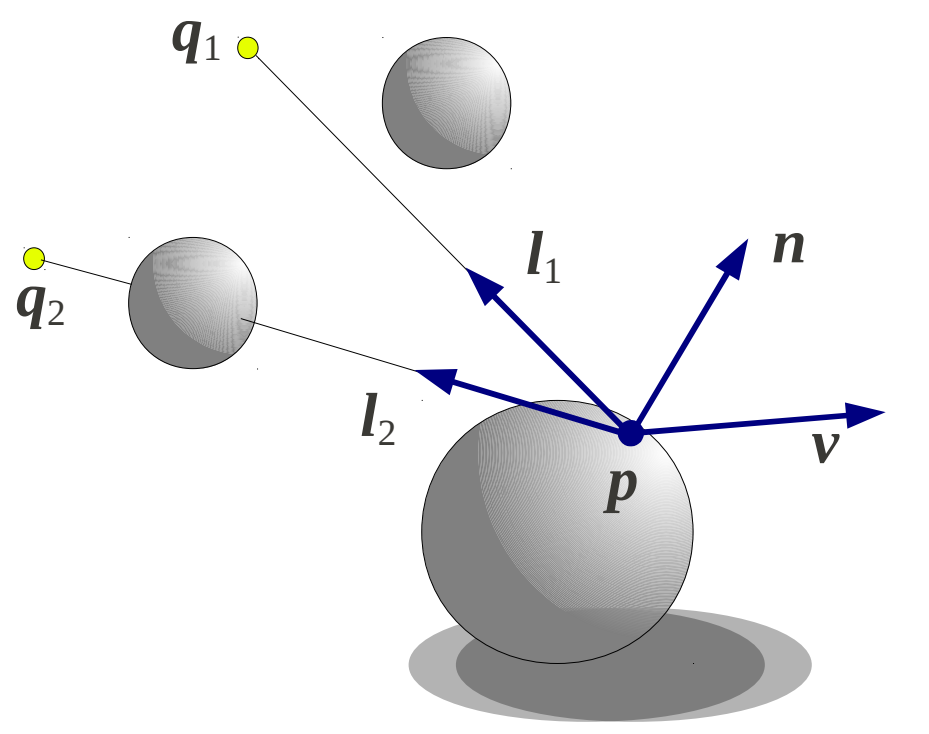
Una vez se conoce el punto p, se obtienen la normal n, y los parámetros del MIL, que es evaluado:



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

Sombras arrojadas

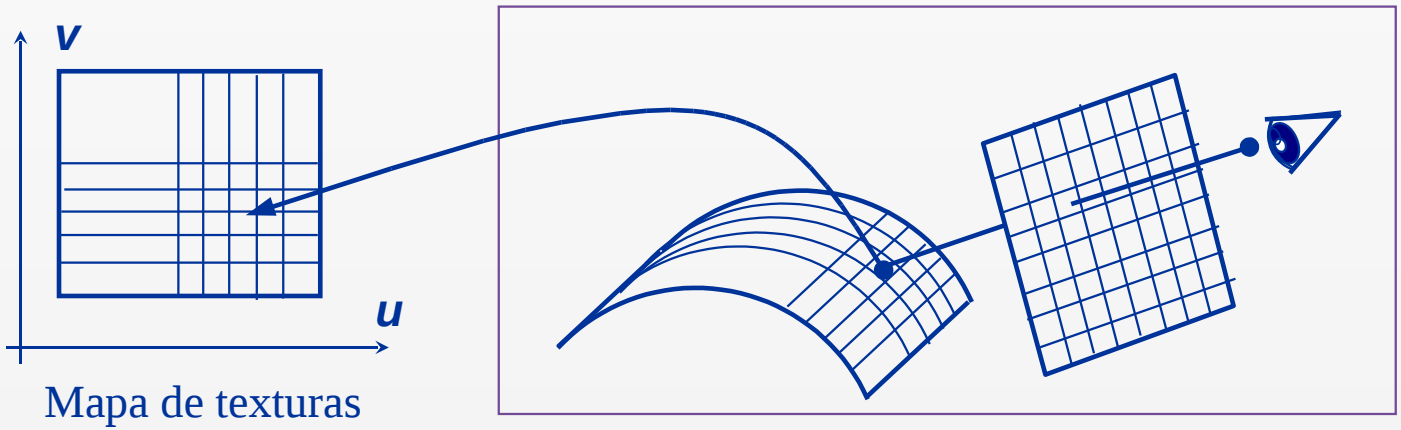
Este método permite incorporar sombras arrojadas, usando Ray-Casting para visibilidad. Se comprueba si un segmento de recta desde p hasta (o hacia) la fuente interseca algún objeto de la escena, es decir, se evalua $V(p, q_i)$



Detalles de las superficies

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

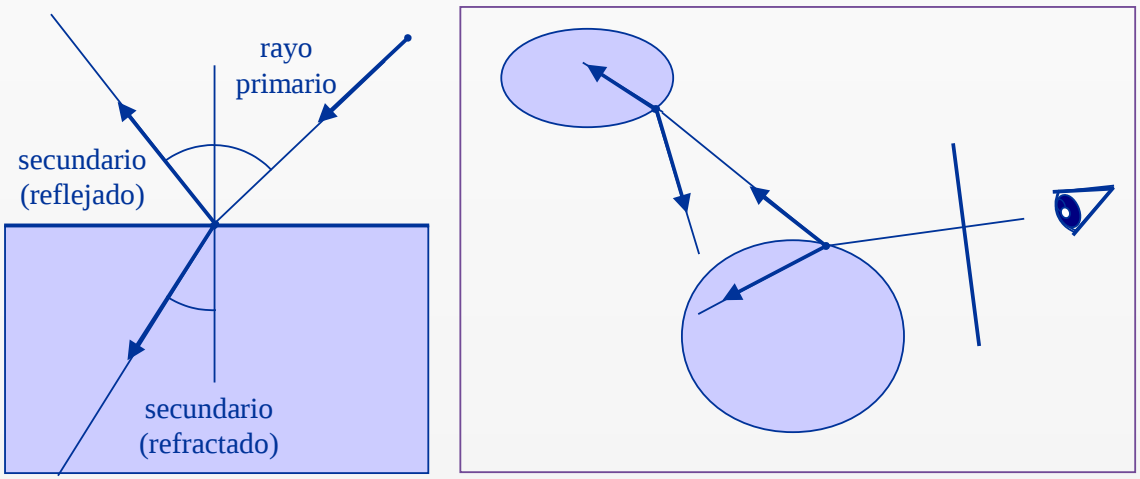
- A partir de p se obtienen las coordenadas (u, v) en el espacio de la textura
- A partir de (u, v) se consulta la textura o texturas asociadas al objeto.



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.

Subsección 2.3
Esquema del algoritmo

El procedimiento principal de Ray-Tracing

El pseudocódigo del algoritmo puede quedar así:

```
Sea o = posicion del observador, en coords. del mundo

Para cada pixel (i,j) de la imagen
   q := punto central (en CM) del pixel (i,j)
   u := vector desde o hasta q normalizado
   L := RayTracing(o,u,1 )
   I[i,j] := L
```

La función **RayTracing** es recursiva, devuelve un color, y tiene un parámetro que sirve para que la recursión no se haga infinita

La función **RayTracing**

Función **RayTracing**(punto o, vector u, entero n)

```
Si n > max
     devolver color negro

A := primer objeto visible desde o en la dir. u
Si no existe ningun objeto
     devolver color de fondo corr. a u

p := punto de A intersecado
n := vector normal a A en p
(u,v) := coordenadas de textura de p

I := EvaluaMIL( A, p, n, -u, u,v,n )
Devolver I
```

La función **EvaluaMIL**

Función **EvaluaMIL**(objeto A, punto $\boldsymbol{p}$, vectores $\boldsymbol{n}$,$\boldsymbol{v}$, entero n)

```
Para cada fuente de luz puntual con posicion q_i
    s_i := V(p, q_i)

Obtener todos los atributos del
    objeto A en p  (k_a, k_d, k_s, k_t, etc....)
I := resultado de evaluar el MIL en p
Si k_t > 0 entonces
    t := vector refractado respecto de v
    I := I + k_t * RayTracing( p, t, n+1 )
Si k_ps > 0 entonces
    r := vector reflejado respecto de v
    I := I + k_ps * RayTracing( p, r, n+1 )

devolver I
```

Ejemplos (1)

Fin de la presentación.