Efectos de iluminación

Normal mapping y modelos alternativos de iluminación

flickr.com/photos/119580457@N06

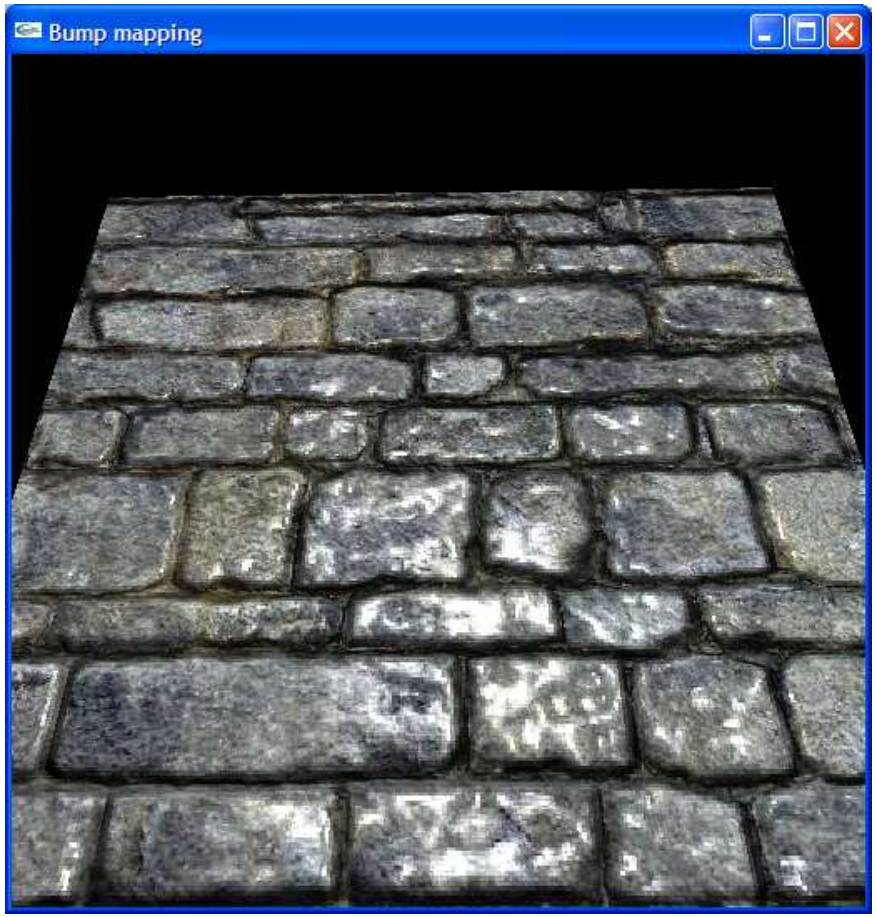
Bibliografía:

- Superbiblia, 7ª ed. 582-599
- Real Time Rendering 4th ed., 6.7, 6.8, 10.4, 11.3

Índice

- *Normal mapping*
  - *Bump Mapping*
  - *Parallax Mapping*
- Modelos alternativos de iluminación
  - Iluminación semiesférica
  - Mapas de entorno
  - Iluminación basada en imagen
  - Oclusión ambiental

Bump mapping



Bump mapping

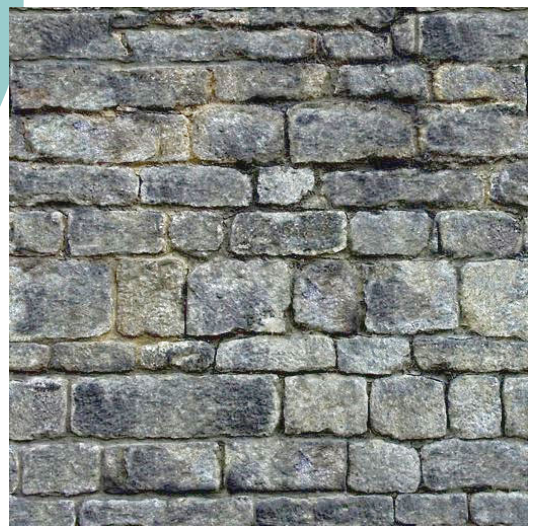
- Es una técnica muy utilizada para dar realismo a las superficies sin necesidad de añadir geometría
- Consiste en, a la hora de calcular la iluminación, perturbar la normal según cierto patrón para simular rugosidades
  - El patrón puede ser procedural o venir dado en una textura

Bump mapping

- La geometría de la superficie realmente no se modifica, por lo que en las siluetas se ve el truco
  - Por ello, el *bump mapping* sólo se usa para añadir pequeños detalles, no para grandes cambios de forma
- Es una técnica que trabaja a nivel de fragmento

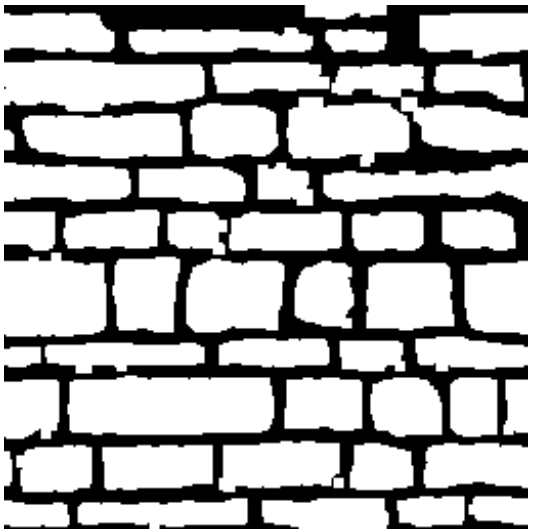
nvidia.com/object/tessellation.html

Bump mapping



Textura base

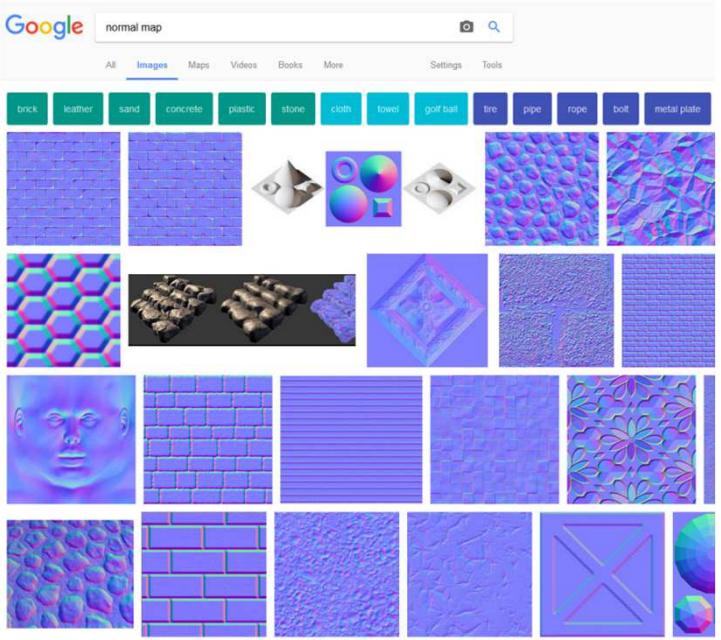
Mapa de normales



Mapa de brillo

Bump mapping

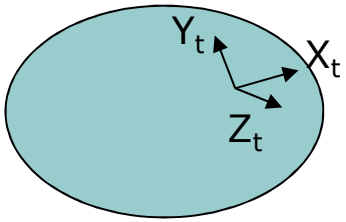
- El mapa de normales codifica una normal en cada texel
- Para convertir un color en una normal hay que escalar el rango de valores
  - Color: [0..1, 0..1, 0..1]
  - Normal: [-1..1, -1..1, -1..1]
- Por ejemplo, el color 0.5, 0.5, 1.0 se corresponde con la normal:
  - [0, 0, 1]

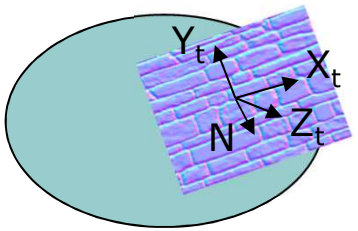


Búsqueda de "normal map" en Google Images

Bump mapping

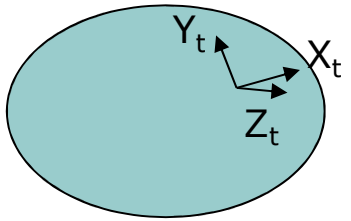
- Para calcular la iluminación por vértice, normalmente se trabaja en el espacio de la cámara
- Para aplicar el bump mapping, vamos a utilizar la normal almacenada en la textura
  - Consideramos que dicha normal está definida en el espacio tangencial
  - En el espacio tangencial, el origen está en el punto donde se desea calcular la iluminación, y la normal geométrica define la dirección del eje Z





Bump mapping

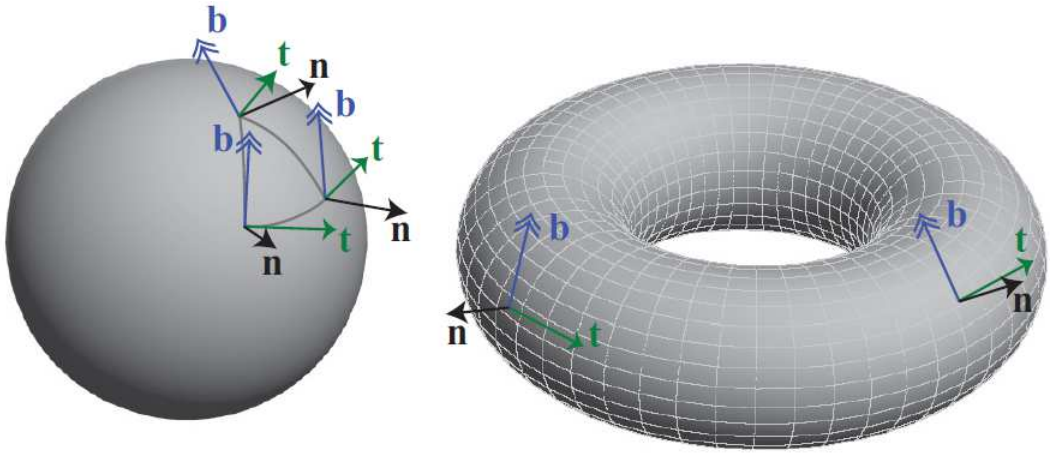
- Por lo tanto, para hacer los cálculos de iluminación, habrá que llevar los vectores de iluminación y de la vista al espacio tangencial del vértice (dicho espacio cambia para cada vértice)
- Para definir la rotación necesaria para llevar dichos vectores al espacio tangencial, debemos definir el sistema ortonormal:



donde, Z_t es la normal (sin perturbar) al vértice, pero X_t e Y_t no están determinados.

Bump mapping

- Así, para cada vértice debemos añadir un atributo adicional:
  - el vector tangente (T) a la superficie
  - los vectores tangentes deben estar calculados consistentemente en toda la superficie:



RTR4, Fig.6.32

Bump mapping

- Para calcular la rotación que lleva un vector del espacio de la cámara al tangencial:
  1. Transformar N y T al espacio de la cámara
     1. N con la normalMatrix y T con la modelviewMatrix
  2. Calcular el otro vector tangente (*bitangente*, o *binormal*) a la superficie (y perpendicular a N y T):

     $B = N \times T$
  3. Crear la matriz de rotación (T, B y N son unitarios):

$$R = \begin{pmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{pmatrix}$$

  4. $L_{Tangencial} = R \cdot L_{Cámara}$

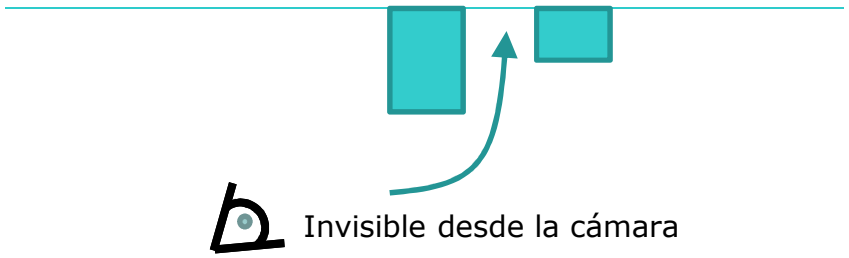
Bump mapping

- Vertex shader
  - Definir L y V como *out*
  - Calcular L y V en el espacio tangencial (¡no normalizar!)
  - Propagar la coordenada de textura al shader de fragmento

- Fragment shader
  - Leer la normal perturbada desde la textura (coordenadas entre [0,0,0] y [1,1,1]), escalarla (entre [-1,-1,-1] y [1,1,1]) y normalizarla
  - Normalizar los vectores interpolados L y V
  - Aplicar la ecuación de iluminación de Phong

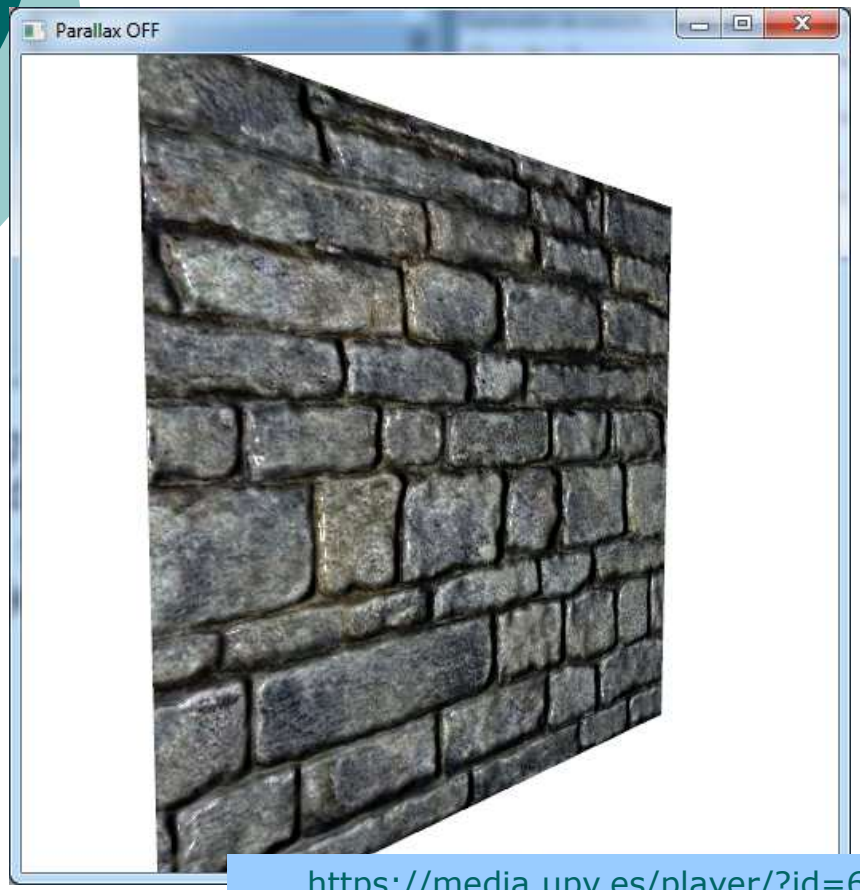
Parallax Mapping

- El problema del *bump mapping* es que, al fin y al cabo, estamos simulando salientes en una superficie, pero dichos salientes no ocultan otras porciones de la superficie
- El paralaje ocurre cuando las posiciones relativas entre dos objetos cambian cuando el observador se mueve
- La técnica del parallax mapping usa un mapa de alturas para aproximar qué se debería ver en un pixel a la altura de la posición encontrada
- Es una técnica sencilla que da buenos resultados y se usa mucho en la actualidad



Parallax Mapping

Bump mapping



Bump mapping + parallax mapping

https://media.upv.es/player/?id=6af796db-b0f7-4863-b9b6-e4983cb06912

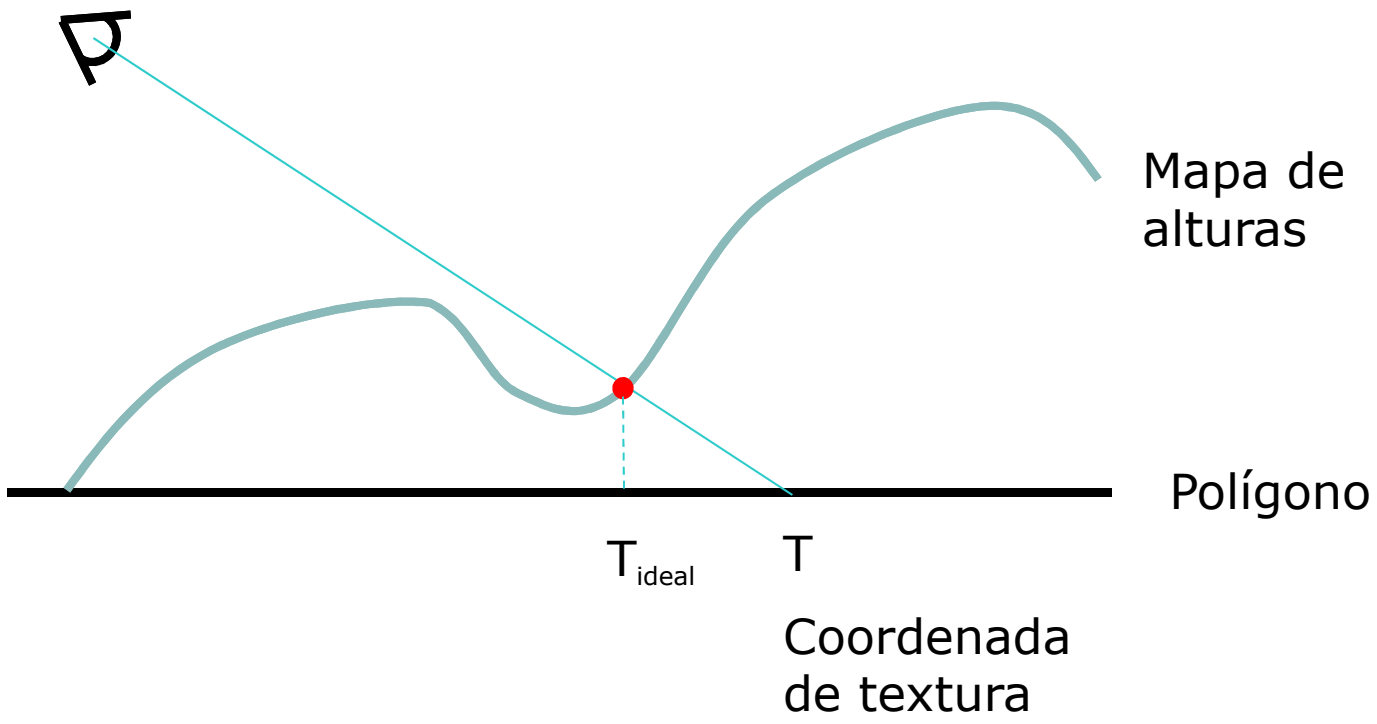
Parallax Mapping

- El parallax mapping necesita una textura adicional con el mapa de alturas de la superficie:

Textura en blanco y negro, con valores entre 0 (altura mínima) y 1 (altura máxima).

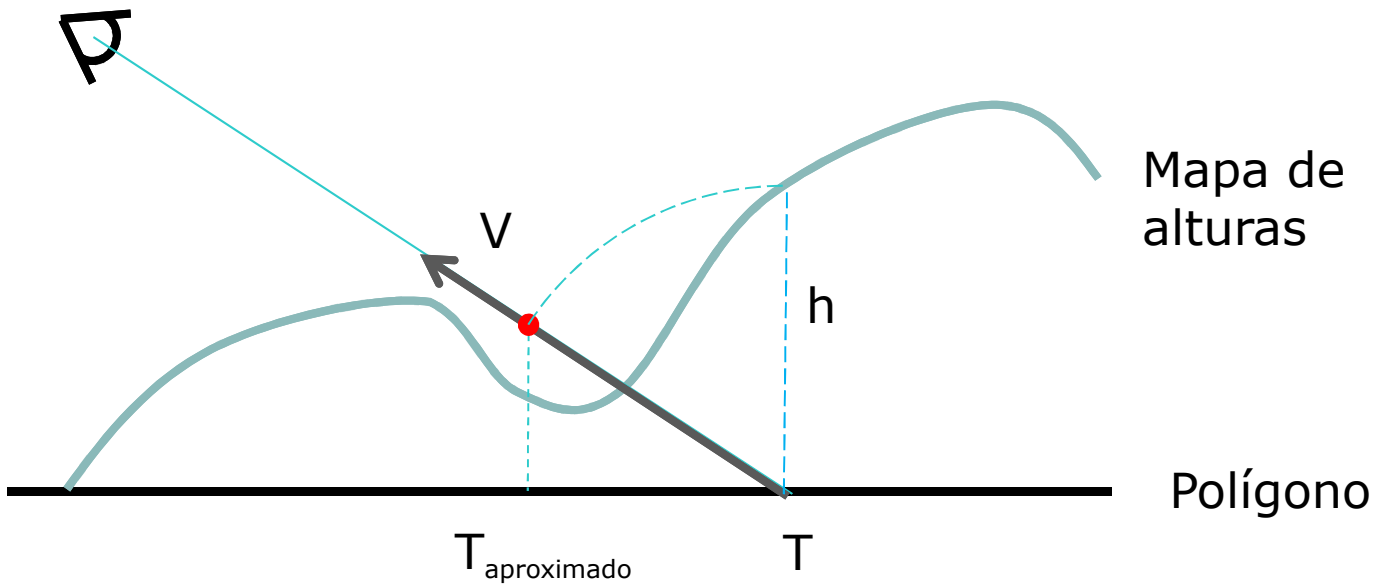
Parallax Mapping

- Objetivo:



Parallax Mapping

- Aproximación (no obtiene el resultado exacto):

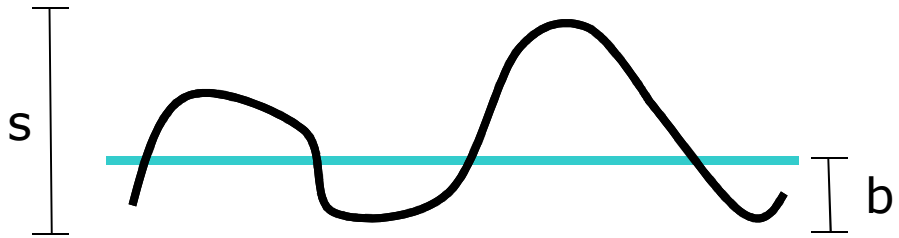


Parallax Mapping

- Cómo implementarlo:
  - Primero, ajustar h:

$$h = h_t \cdot s - b$$

  - donde:
    - h_t: altura dada por el mapa (entre 0 y 1)
    - s: escala. Define el rango de alturas del mapa
    - b: bias. Establece la altura del polígono con respecto al mapa de alturas



Parallax Mapping

- Después, calcular la nueva coordenada de textura:

$$T_{ajustada} = T + h \cdot V_{xy}$$

- donde:
  - T: coordenada de textura interpolada para el fragmento
  - V_{xy}: coordenadas *x,y* del vector vista en el espacio tangencial
- Por último, usar la coordenada de textura ajustada en el cálculo del *bump mapping* (igual que antes)

Modelos alternativos de iluminación

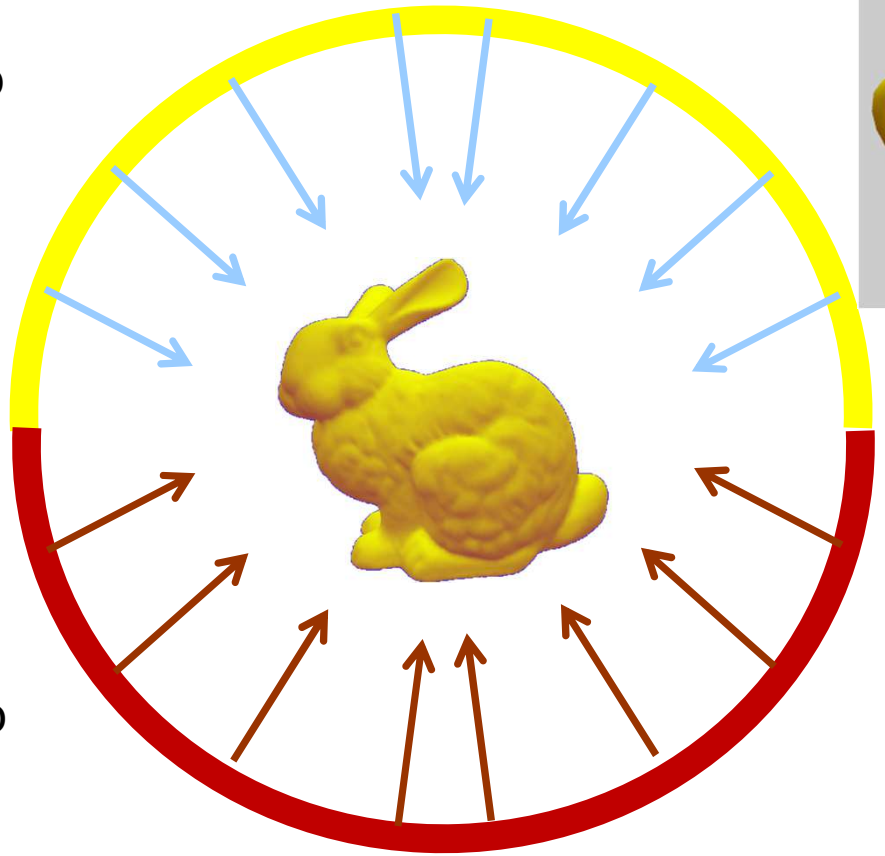
- La iluminación tradicional de OpenGL es poco realista: un objeto sólo recibe luz de las fuentes, las primitivas que no están orientadas hacia una luz reciben un término ambiental constante, etc.
- Dada la flexibilidad de las GPU programables, se pueden implementar otros modelos de iluminación

Iluminación semiesférica

- Este modelo supone que el objeto está iluminado por una superficie esférica que lo envuelve
- Dicha superficie está dividida en dos: el cielo (iluminado) y el suelo (no iluminado)
- Cada zona define un color
- Mejora el cálculo de la componente ambiental, y se puede integrar en el cálculo de iluminación "normal" de OpenGL

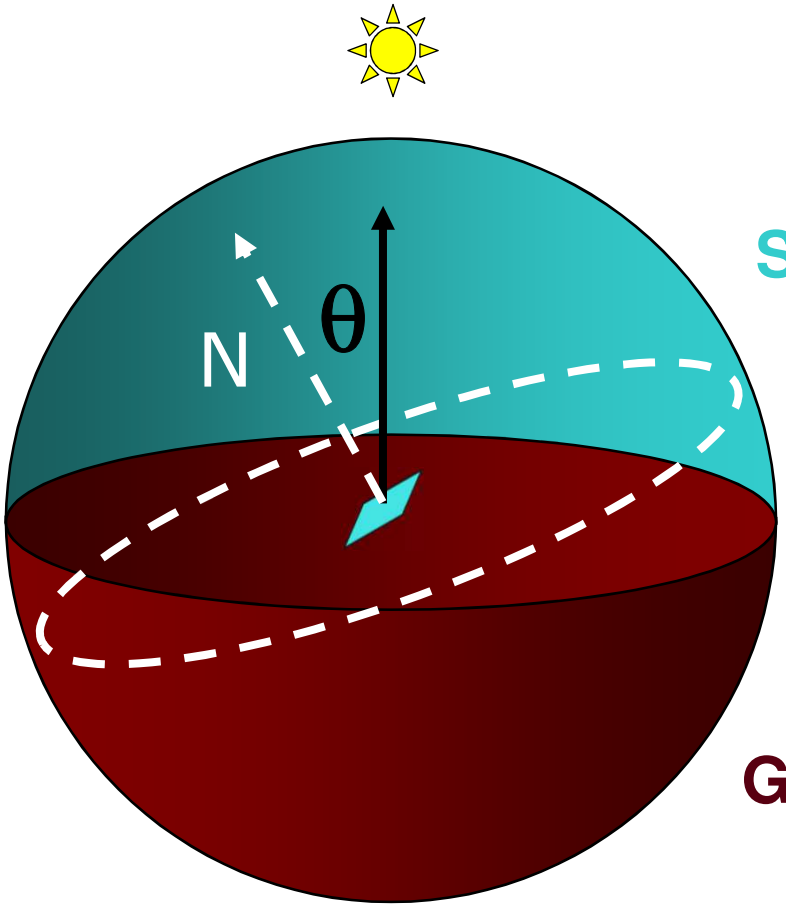
Iluminación semiesférica

Hemisferio iluminado



Hemisferio de suelo

Iluminación semiesférica

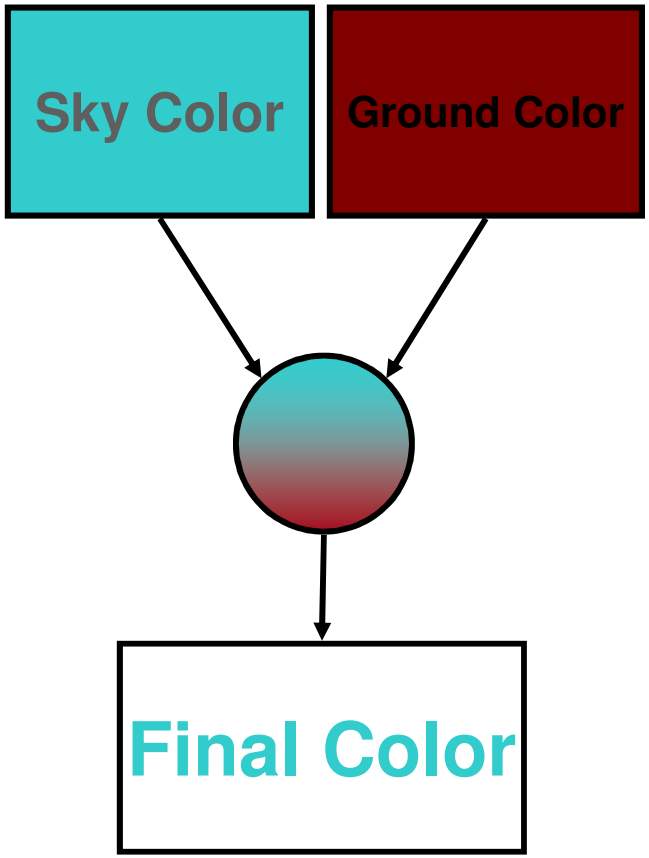


Sky Color

Ground Color

http://www.microsoft.com/corpevents/gdc2001/slides/Per_Pixel_Lighting.ppt

Iluminación semiesférica



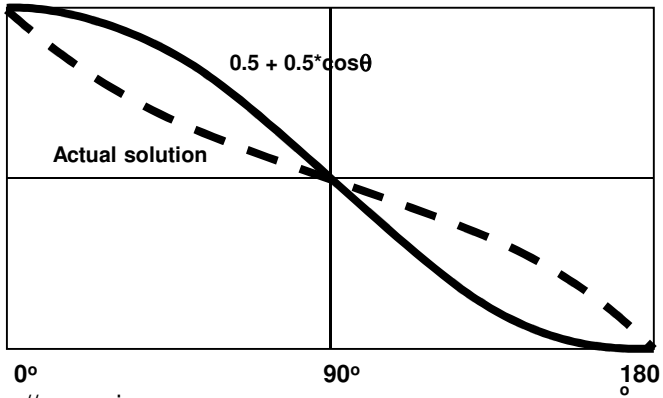
$$\text{Color} = \alpha \cdot \text{SkyColor} + (1-\alpha) \cdot \text{GndColor}$$

La fórmula para calcular α :

$$\begin{cases} \alpha = 1 - 0.5 \cdot \sin(\theta) & si\ \theta \leq 90 \\ \alpha = 0.5 \cdot \sin(\theta) & si\ \theta > 90 \end{cases}$$

Pero se puede aproximar por:

$$\alpha = 0.5 + 0.5 \cdot \cos(\theta)$$



Iluminación semiesférica

ej8-1/shader.vert

```
#version 420
$GLMatrices
in vec4 position;
in vec3 normal;

// Posición de la fuente en el espacio de la cámara
uniform vec3 lightpos;
uniform vec4 SkyColor;
uniform vec4 GroundColor;
out vec4 fragColor;

void main() {
  vec3 ecPosition = (modelviewMatrix * position).xyz;
  vec3 tnorm = normalize(normalMatrix * normal);
  vec3 lightVec = normalize(lightpos - ecPosition);
  float costheta = dot(tnorm, lightVec);
  float a = 0.5 + 0.5 * costheta;

  fragColor = mix(GroundColor, SkyColor, a);
  gl_Position =  modelviewprojMatrix * position;
}
```

```
mix(x,y,a)=x*(1.0-a)+y*a
```

Iluminación semiesférica

ej8-1/shader.frag

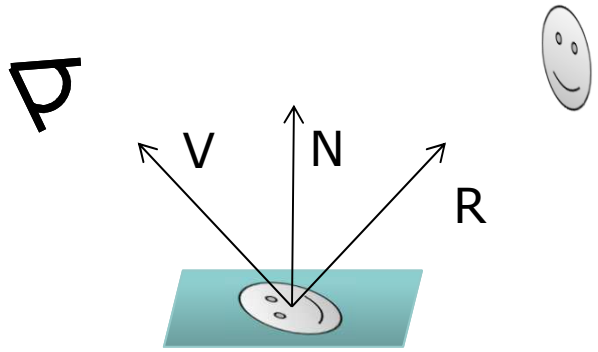
```
#version 420

in vec4 fragColor;
out vec4 finalColor;

void main() {
  finalColor = fragColor;
}
```

Iluminación semiesférica

- Ventajas:
  - Da mejores resultados que la iluminación por defecto de OpenGL, ¡sin luces!
- Problema:
  - No tiene en cuenta auto-oclusiones
  - El color del suelo/cielo es constante

Mapas de entorno

- Permiten modelar eficientemente entornos reales con iluminación compleja
- Puede simular reflejos en objetos brillantes, sin usar trazado de rayos
- Los reflejos se aproximan usando el vector reflexión y una textura

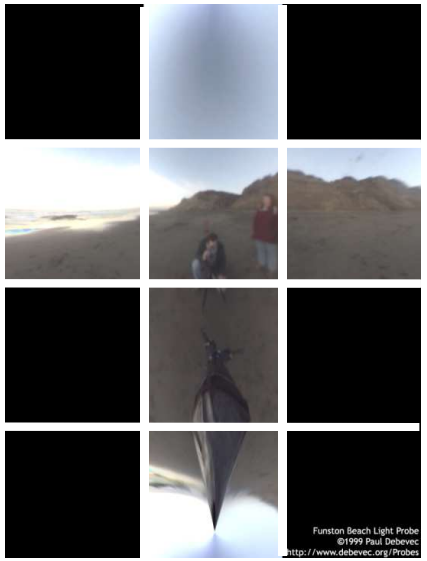


Mapas de entorno

- En la tubería fija de OpenGL se podían usar dos tipos de mapas de entorno:
  - Esféricos (*Sphere mapping*). Descartado en OpenGL 3
  - Cúbicos (*Cube mapping*)
- La diferencia radica en el tipo de textura utilizada:

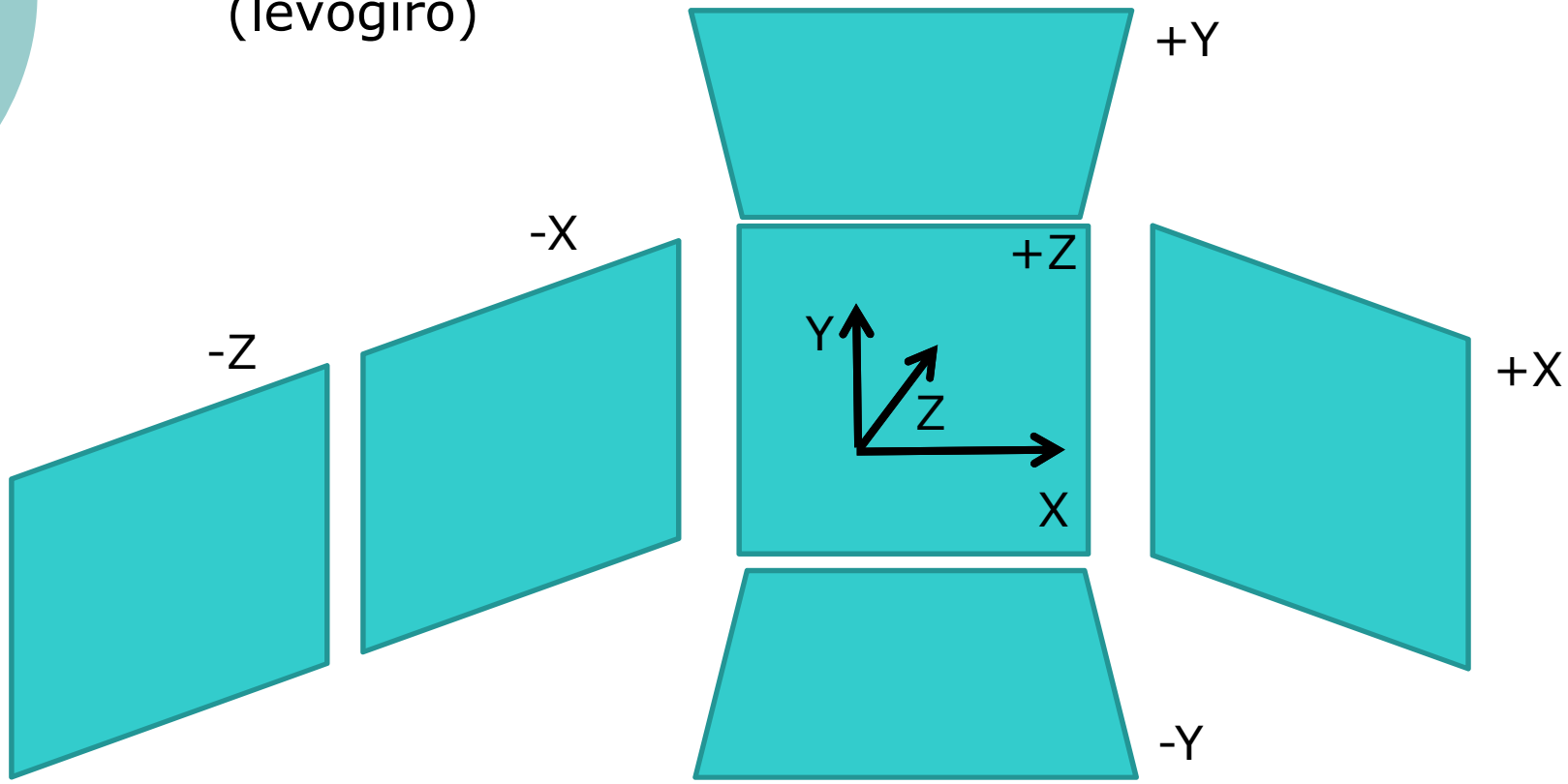
Mapas de entorno
Cúbicos

- En un mapa de entorno cúbico, hay 6 texturas cuadradas que representan las caras de un cubo que envuelve el objeto que se desea dibujar
- Cada cara representa una dirección
- Puedes construir mapas cúbicos con:
  - CubeMapGen de AMD o Gimp (con un plugin) para construir los mapas cúbicos en formato DDS
  - Mantener cada una de las 6 texturas en un fichero aparte
- Tienes un montón de ejemplos en: http://www.humus.name
- ¡Cuidado! OpenGL usa un sistema de coordenadas distinto al habitual para definir las direcciones del mapa de entorno (lo heredó de RenderMan)



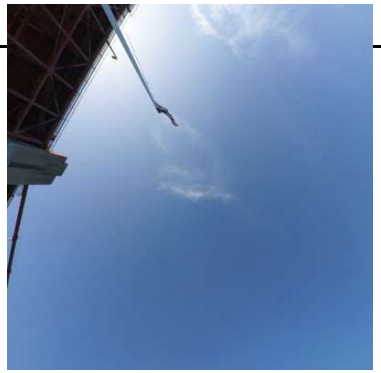
Mapas de entorno
Cúbicos

- Sistema de coordenadas para los mapas cúbicos (levógiro)



Mapas de entorno

Cúbicos



posy

negz

negx

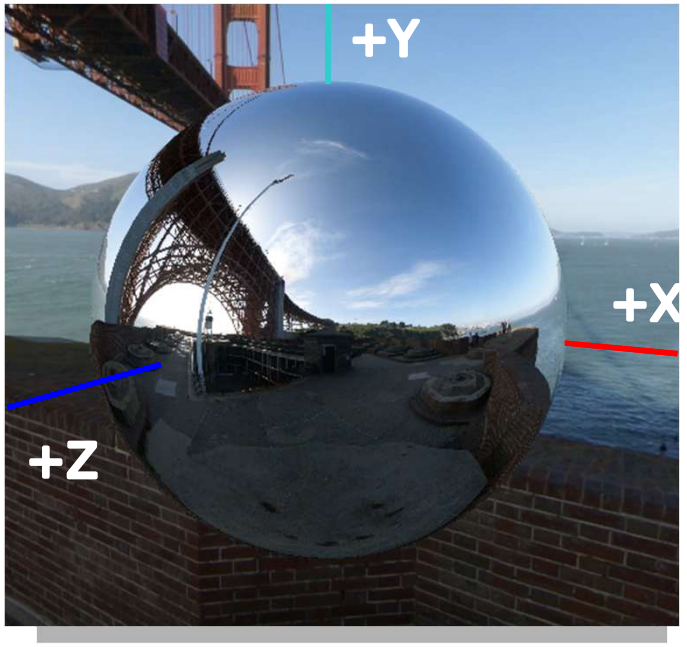
posz

posx

negy

Mapas de entorno
Cúbicos

- Fíjate en la diferencia entre el sistema de coordenadas de OpenGL y el del cubo



Mapas de entorno

Cúbicos

- Para calcular el texel que corresponde a un punto del objeto se usa un vector 3D (r_x, r_y, r_z), normalmente el vector reflexión:
  - Se usa la textura correspondiente a la componente máxima en valor absoluto, y su signo
  - Las otras dos coordenadas se dividen por la coordenada mayor para calcular la posición del *texel* dentro de la textura, y se normaliza a un valor entre 0 y 1
- Ejemplo:
  - *R=(-0.78, 0.3, -0.5)*
  - Se selecciona la textura –X (*0.78=max(|-0.78|, |0.3|, |-0.5|)*)
  - Las coordenadas que calcula OpenGL para acceder al texel son:

$$s = \frac{1}{2}\left(\frac{s_c}{|m_a|}+1\right) \qquad t = \frac{1}{2}\left(\frac{t_c}{|m_a|}+1\right)$$

Mapas de entorno
Cúbicos

- donde:

Major Axis Direction	Target	s_c	t_c	m_a
$+r_x$	TEXTURE_CUBE_MAP_POSITIVE_X	$-r_z$	$-r_y$	r_x
$-r_x$	TEXTURE_CUBE_MAP_NEGATIVE_X	r_z	$-r_y$	r_x
$+r_y$	TEXTURE_CUBE_MAP_POSITIVE_Y	r_x	r_z	r_y
$-r_y$	TEXTURE_CUBE_MAP_NEGATIVE_Y	r_x	$-r_z$	r_y
$+r_z$	TEXTURE_CUBE_MAP_POSITIVE_Z	r_x	$-r_y$	r_z
$-r_z$	TEXTURE_CUBE_MAP_NEGATIVE_Z	$-r_x$	$-r_y$	r_z

- Ejemplo:
  - *R=(-0.78, 0.3, -0.5)*
  - Por lo tanto: $s_c = -0.5, t_c = -0.3, m_a = -0.78$
  - y:

$$s = \frac{1}{2}\left(\frac{-0.5}{|-0.78|}+1\right) = 0.18 \qquad t = \frac{1}{2}\left(\frac{-0.3}{|-0.78|}+1\right) = 0.31$$

Mapas de entorno
Cúbicos

- En la aplicación:
  - Cargar las 6 texturas:
  - `glTexImage2D(GL_TEXTURE_CUBE_MAP_{POSITIVE_X | POSITIVE_Y | POSITIVE_Z | NEGATIVE_X | NEGATIVE_Y | NEGATIVE_Z}, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *texels)`
    - ¡Usa las direcciones definidas por el mapa de entorno!
  - `glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_{S,T,R}, <modo-repeticion>);`
  - `glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_{MIN|MAG}_FILTER, <filtro>);`

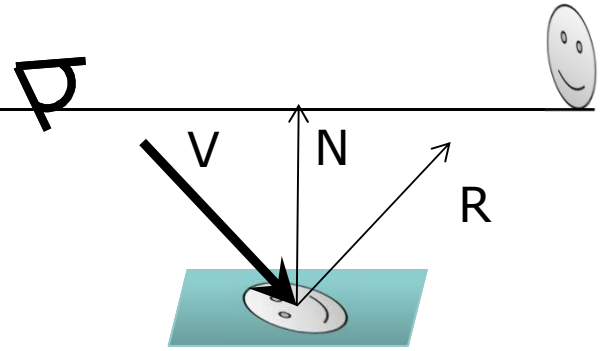
Mapas de entorno
Cúbicos

- Si suponemos que el mapa de entorno está alineado con el sistema de coordenadas del mundo, deberemos calcular el vector reflejado en dicho sistema.
- Para obtener el texel en el shader:

```
uniform samplerCube envmap;
vec4 envColor = texture(envmap, R);
```

Mapas de entorno

Cúbicos



reflect(V,N) → R

ej8-2\reflect.vert

```
#version 420
$GLMatrices
// Indica si debemos trabajar en un
// espacio levógiro o no
uniform bool levo = true;
in vec3 normal;
in vec4 position;
// Vector en la dirección del reflejo
out vec3 reflectDir;
void main(void)
{
  vec3 N = normalize(normalMatrix * normal);
  vec4 V = modelviewMatrix * position;
  // Para una matriz de rotación, su inversa y su transpuesta
  // son iguales
  reflectDir = transpose(mat3(viewMatrix)) * reflect(V.xyz, N);
  if (levo) reflectDir *= vec3(1.0, 1.0, -1.0);
  gl_Position = projMatrix * V;
}
```

Mapas de entorno
Cúbicos

ej8-2\reflect.frag

```
#version 420

uniform samplerCube envmap;

in vec3 reflectDir;
out vec4 finalColor;

void main(void)
{
  // El color final viene dado por el color de la textura
  // en la dirección del reflejo
  finalColor = texture(envmap, reflectDir);
}
```

Mapas de entorno

Otros tipos

- Esféricos
  - Dado el vector de reflexión (r_x, r_y, r_z):

$$m = 2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}$$

$$s = \frac{r_x}{m} + \frac{1}{2}$$

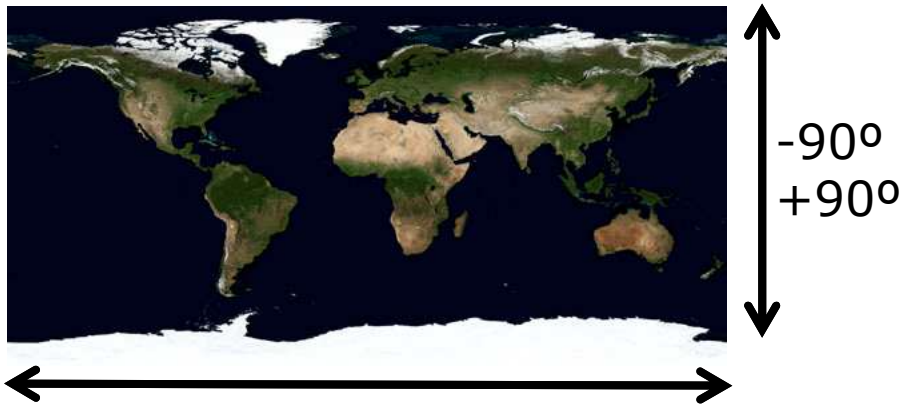
y

$$t = \frac{r_y}{m} + \frac{1}{2}$$

Mapas de entorno

Otros tipos

- Mapa equirectangular o de latitud-longitud
  - Hay que calcular la latitud/longitud correspondiente
  - Latitud: ángulo entre el vector de reflexión y el plano XZ
  - Longitud: se proyecta el vector reflexión en el plano XZ *(r_x, 0, r_z)* y se calcula su ángulo con el eje X
  - Se mapean los ángulos al rango [0, 1]



Iluminación basada en imagen

Image-based lighting

- Es difícil simular la iluminación de ciertas escenas de interior
- Por ello, la iluminación basada en imagen usa la luz real capturada mediante fotografías para iluminar objetos sintéticos
- En www.debevec.org puedes encontrar información y herramientas para capturar la iluminación de una escena (HDRShop)

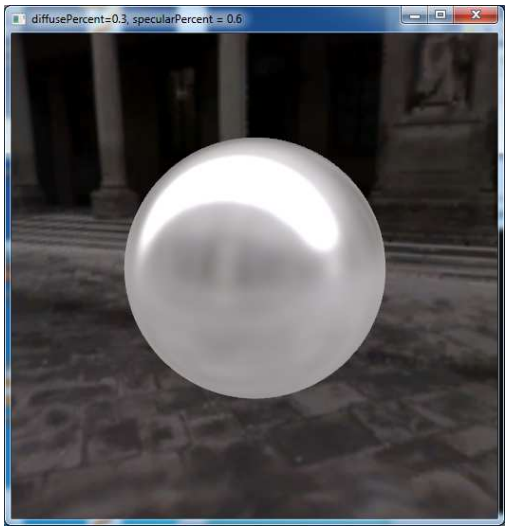
http://www.debevec.org

Iluminación basada en imagen

- Pasos:
  1. Capturar la luz de la escena (HDRI)
  2. Adaptar la imagen anterior para usarla en OpenGL (*environment mapping*)
  3. Situar los objetos sintéticos dentro del entorno capturado
  4. Visualizar los objetos

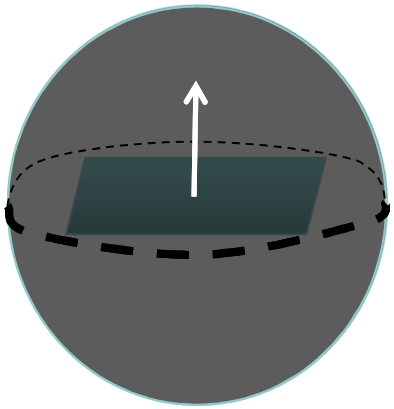
Iluminación basada en imagen

- Las imágenes de alto rango dinámico (HDR) almacenan un *float* por canal (en vez de un *byte*)
- Esto permite capturar más fielmente la iluminación de un entorno real
- Para iluminar un objeto usando un mapa HDR separamos las componentes en dos:
  - Difusa
  - Especular



Iluminación basada en imagen

- Componente difusa:
  - Un objeto difuso refleja luz de todos los puntos de la semiesfera del entorno visible
  - Sin embargo, no se puede acceder a la mitad del mapa de entorno para iluminar un pixel



Iluminación basada en imagen

- Componente difusa:
  - Solución: precalcular un mapa difuso a partir del mapa de entorno (mapa de irradiancia difusa)
  - Cada texel de este mapa contiene información de la luz de una semiesfera en una dirección dada
  - Como no hay altas frecuencias, se pueden usar mapas pequeños

Uffizi Gallery en http://www.debevec.org

Iluminación basada en imagen

- Componente especular
  - Calcular un mapa de irradiancia especular
  - Si el objeto es un espejo, usar el mapa inicial, si está pulido pero no tanto (exponente de Phong < ∞) usar un mapa precalculado:

HDRShop

Exp.
Phong 50

Exp.
Phong 100

Iluminación basada en imagen

ej8-3\reflect2.vert

```
#version 420
$GLMatrices
in vec3 normal;
in vec4 position;
// Vector en la dirección del reflejo
out vec3 reflectDir;
// Vector normal
out vec3 normalDir;
void main(void)
{
  vec3 N = normalize(normalMatrix * normal);
  vec4 V = modelviewMatrix * position;
  // Para una matriz de rotación, su inversa y su transpuesta
  // son iguales
  reflectDir = transpose(mat3(viewMatrix)) * reflect(V.xyz, N);
  normalDir = transpose(mat3(viewMatrix)) * N;
  reflectDir *= vec3(1.0, 1.0, -1.0);
  normalDir *= vec3(1.0, 1.0, -1.0);
  gl_Position = projMatrix * V;
}
```

Iluminación basada en imagen

ej8-3.frag

```
#version 420

uniform vec3 baseColor;
uniform float specularPercent;
uniform float diffusePercent;
uniform samplerCube specularMap;
uniform samplerCube diffuseMap;

in vec3 reflectDir;
in vec3 normalDir;
out vec4 finalColor;

void main(void) {
  vec4 diffColor = texture(diffuseMap, normalDir);
  vec4 specColor = texture(specularMap, reflectDir);
  vec4 color = mix(baseColor, diffColor*baseColor, diffusePercent);
  color = mix(color, specColor + color, specularPercent);
  finalColor = vec4(color.xyz, 1.0);
}
```

Iluminación basada en imagen

Armónicos esféricos (Spherical Harmonics)

- Dado que la componente difusa no tiene altas frecuencias, se puede comprimir aún más mediante un análisis frecuencial
- Los armónicos esféricos permiten codificar la información de una imagen esférica en el dominio de la frecuencia (~Fourier)
- Así, podemos representar un mapa de irradiancia difusa mediante una función matemática
- La ecuación propuesta por Ramamoorthi y Hanrahan tiene sólo 9 coeficientes, y permite almacenar cualquier distribución de luz real con un error promedio menor del 3%

Iluminación basada en imagen

Armónicos esféricos

- La ecuación que modela la reflexión difusa es:

$$Diffuse = c_1 L_{22}\left(x^2 - y^2\right) + c_3 L_{20} z^2 + c_4 L_{00} - c_5 L_{20} + 2c_1\left(L_{2-2}xy + L_{21}xz + L_{2-1}yz\right) + 2c_2\left(L_{11}x + L_{1-1}y + L_{10}z\right)$$

donde:

$c_1 = 0.429043$

$c_2 = 0.511664$

$c_3 = 0.743125$

$c_4 = 0.886227$

$c_5 = 0.247708,$

los coeficientes L_{nm} (vectores 3d) se calculan a partir del mapa de irradiancias y (x, y, z) es la normal unitaria

Iluminación basada en imagen

Armónicos esféricos

- Ejemplo:

Uffizi Gallery en http://www.debevec.org

L_{00}	.32	.31	.35
L_{1-1}	.37	.37	.43
L_{10}	.0	.0	.0
L_{11}	-.01	-.01	-.01
L_{2-2}	-.02	-.02	-.03
L_{2-1}	-.01	-.01	-.01
L_{20}	-.28	-.28	-.32
L_{21}	.0	.0	.0
L_{22}	-.24	-.24	-.28

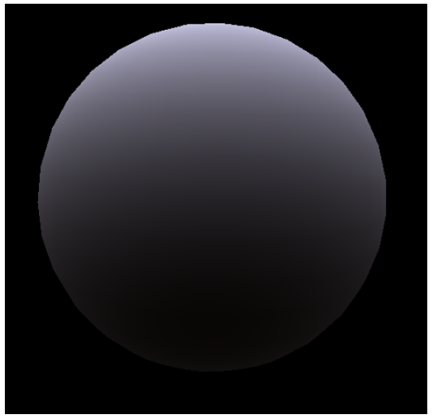
Coeficientes del mapa

Iluminación basada en imagen

Armónicos esféricos

- Ejemplo:

Mapa original



Modelos difusos iluminados con armónicos esféricos

Iluminación basada en imagen

Armónicos esféricos

- Ventajas:
  - No hace falta texturas
  - Se puede integrar el cálculo de iluminación tradicional de OpenGL
- Inconvenientes:
  - Al calcularse a partir de una imagen HDR, puede tener problemas de escalado

Oclusión ambiental

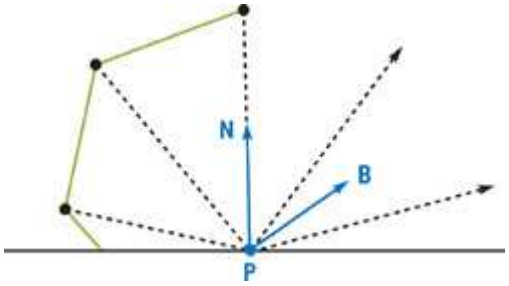
Ambient occlusion

- Uno de los problemas de la iluminación que usa OpenGL por defecto es que asume que la iluminación ambiental es constante
- En realidad, la luz que llega a una zona depende de los objetos que la rodean
- La oclusión ambiental precalcula un factor de oclusión (o accesibilidad) para cada punto de un modelo, que indica cómo de visible es dicho punto
- El factor de oclusión se usa para modular la componente ambiental

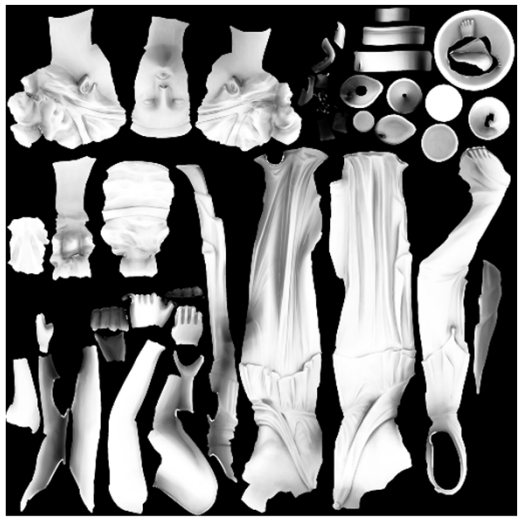
Oclusión ambiental

Ambient occlusion

- Para calcular el factor de oclusión, se calcula para cada punto del objeto qué parte de la semiesfera en la dirección de su normal es visible
- Dicho factor se precalcula, y se almacena o bien como un atributo para cada vértice, o bien como una textura para toda la superficie del objeto



GPU Gems, cap, 17. Disponible en Nvidia



Oclusión ambiental

Ambient occlusion

- Esta técnica se puede combinar con las anteriores para obtener mejores resultados
- Ventajas:
  - El mapa de oclusiones es independiente del punto de vista
  - Obtiene resultados muy efectivos con una técnica muy rápida (porque se precalcula)
- Desventajas:
  - Sólo se puede precalcular si el modelo es rígido. Si es animado, hay que recalcular cada vez que cambia. Buscar, por ejemplo Screen Space Ambient Occlusion (SSAO)
  - No tiene en cuenta que otros objetos puedan bloquear la iluminación del objeto