

# **Apunts-G.pdf**



**Arnau\_FIB**



**Gráficos**



**3º Grado en Ingeniería Informática**



**Facultad de Informática de Barcelona (FIB)  
Universidad Politécnica de Catalunya**

# **GRÀFICS**

*Resum per a l'examen final*

# INDEX

<b>Sistemes de coordenades i transformacions geomètriques</b>	<b>5</b>
Coordenades homogènies	5
Transformació de punts	5
Object Space	5
Modeling transform	5
World Space	6
Viewing transform	6
Eye Space	6
Projection transform	7
Clip Space	8
Perspective division	8
Normalized Device Space	8
Viewport transform & depth transform	9
Window space	9
Transformació vectors	9
Transformació normals	9
<b>Pipeline OpenGL</b>	<b>10</b>
Dibuix de primitives	10
Per-vertex operations	10
Primitive Assembly	10
Primitive Processing	10
Rasterització	10
Fragment processing ("shading")	10
Per-fragment operations ("raster operations")	11
Frame buffer operations	11
<b>Textures</b>	<b>12</b>
Mida d'una textura	12
Mapping	12
Opcions per a generar coord de textura	12
Mètodes per generar coords de textura	13
Plans S,T	13
Parametrització amb superfície auxiliar	14
Mapping esfèric	14
Mapping cilíndric	14
Creació d'una textura	14
Ús de textura al fragment shader	14
Filtrat	14
Magnification or minification?	15
Magnification filters	16
Minification filters	16

Wrapping	18
Combinació	18
Perspective-correct interpolation	18
Projective Texture Mapping	19
Aplicacions	20
Color mapping	20
Bump mapping / Normal mapping	20
Problema Bump mapping / Normal mapping	21
Parallax mapping	21
Problema Parallax mapping	22
Relief mapping	22
Displacement mapping	23
Bump mapping	23
Normal mapping	25
<b>Ombres</b>	<b>25</b>
Ombres per projecció sense stencil	26
Evitar problemes de Z-fighting	27
Stencil Buffer	28
Establir el test de comparació	29
Operacions a fer a stencil buffer segons el resultat del test	29
Ombres per projecció amb stencil	29
Matrius de projecció	30
Volums d'ombra	31
Frontface / Backface test	33
Shadow Mapping	34
<b>Reflexions especulars</b>	<b>36</b>
Reflexió difosa	36
Reflexió specular	36
Vector reflectit	37
Reflexions basada en objectes virtuals	37
Modelats	37
Reflectits sense stencil buffer	38
Reflectits amb stencil buffer	38
Textures dinàmiques	39
Matriu de reflexió	39
Environment mapping	40
Ús com a entorn (background)	40
Sphere mapping	41
Relació entre els vectors R i (u,v)	41
Càlcul del color donat R	42
Eye / world coordinates	42
Cube mapping	42
Implementació en GLSL	43

<b>Simulació d'objectes translúcids</b>	<b>44</b>
Llei de Snell	44
Equacions de Fresnel	45
Aproximació de Schlick	45
Alpha blending	45
Ordre de pintat dels polígons	46
<b>II·luminació local/global</b>	<b>47</b>
Radiometria	47
Flux radiant $\phi$	47
Irradiancia E	47
Intensitat I	48
Radiància L	48
BRDF / BTDF / BSDF	50
Equació general del rendering	50
Light Paths	50
<b>Ray tracing</b>	<b>51</b>
Ray tracing clàssic	51
Path tracing	51
Distributed ray tracing	51
Two-pass ray-tracing	51
Ray tracing classic	52
Intersecció raig triangle	52
<b>Funcions de GLSL</b>	<b>53</b>
Mix	53
Step	53
Smoothstep	53
Mod	53
Floor	54
Normalize	54
Lambert	54
Phong	54

# Sistemes de coordenades i transformacions geomètriques

## Coordenades homogènies

- Punts

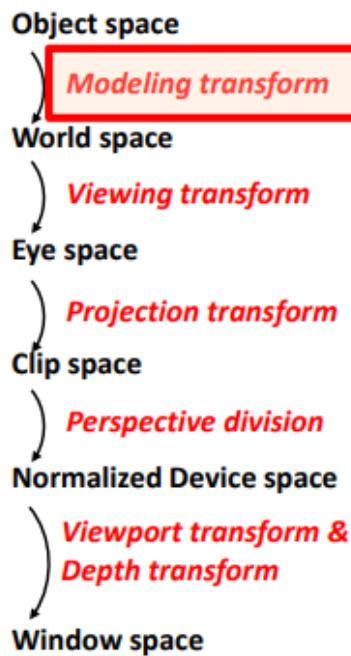
$$(x,y,z) \xrightarrow{\hspace{2cm}} (x,y,z,1)$$
$$(x/w, y/w, z/w) <----- (x,y,z,w) w \neq 0$$

$$(2,3,4,1) == (4,6,8,2)$$

- Vectors

$$(x,y,z) <-----> (x,y,z,0) w = 0$$
$$(2,3,4) == (2,3,4,0)$$

## Transformació de punts



## Object Space

- Sistema de coordenades utilitzat per modelar l'objecte
- W normalment serà:
  - 1.0 (PUNTS)
  - 0.0 (VECTORS)

## Modeling transform

Aquesta transformació situa cada instància d'un objecte en relació a l'escena.

### Matrius de transformació de Modelat

#### Modeling transform

Transformació de modelat

$$\text{glTranslatef}(t_x, t_y, t_z) \quad T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glScalef}(s_x, s_y, s_z) \quad T = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotatef}(a, x, y, z) \quad T = \begin{bmatrix} x^2d + c & xyd - zs & xzd + ys & 0 \\ yxd + zs & y^2d + c & yzd - xs & 0 \\ xzd - ys & yzd + xs & z^2d + c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

c=cos(a), s=sin(a), d=1-cos(a)

### World Space

- Sistema de coordenades utilitzat per representar l'escena.
- La transformació de modelat sovint preserva la component homogènia i per tant, w normalment serà:
  - 1.0 (PUNTS)
  - 0.0 (VECTORS)

### Viewing transform

Transformació de visualització.

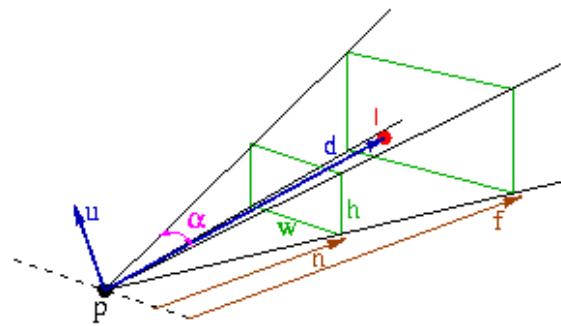
- Fa un canvi al sistema de referència de la càmera.
- Depèn de la posició i orientació de la càmera.
- Sovint es defineix amb lookAt(eye,target,up) o amb translacions i rotacions (angles Euler -> T(0,0,-d)\*Rz(-phi)\*Rx(theta)\*Ry(-psi)\*T(-VRP) ).

### Eye Space

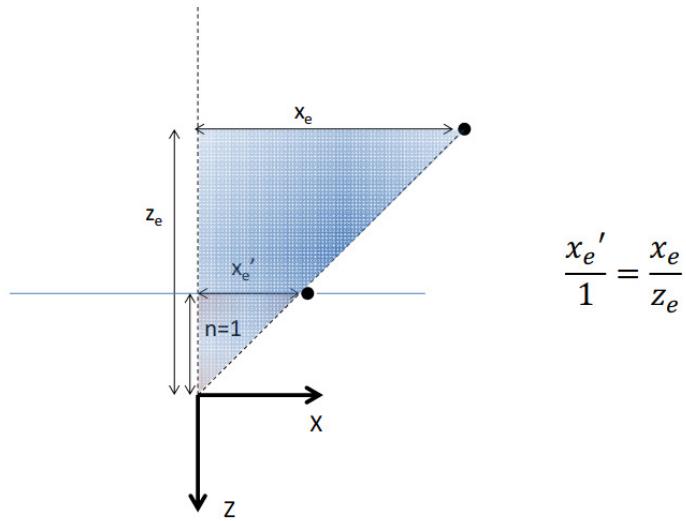
- Sistema de coordenades associat a la càmera.
- La transformació de visualització sovint preserva la component homogènia (1.0-punts, 0.0-vectors).
- Si la càmera és perspectiva, per tal de que el punt sigui visible Ze<0 o Ze<-Znear

## Projection transform

- Depèn de la forma de la piràmide de visió i per tant del tipus de càmera(perspectiva, axonomètrica)
- Sovint es defineix amb crides del tipus perspective, frustum, ortho.
- `perspective(fovy, aspect, near, far)`



Càmera perspectiva



# perspective(90, 1, 1, 11);

$$\begin{bmatrix} \cot \frac{\text{fovy}}{2} & 0 & 0 & 0 \\ 0 & \cot \frac{\text{fovy}}{2} & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2*n*f}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix} \xrightarrow{\substack{\text{fovy}=90, \text{aspect}=1 \\ n=1, f=11}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1.2 & -2.2 \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Pas eye space  $\rightarrow$  clip space

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1.2 & -2.2 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = \begin{bmatrix} x_e \\ y_e \\ -1.2z_e - 2.2 \\ -z_e \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1.2 & -2.2 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = \begin{bmatrix} x_e \\ y_e \\ \color{red}{-1} \\ \color{red}{1} \end{bmatrix}$$

Punt sobre el pla znear

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1.2 & -2.2 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = \begin{bmatrix} x_e \\ y_e \\ \color{red}{-11} \\ \color{red}{11} \end{bmatrix}$$

Punt sobre el pla zfar

Matriu de projecció sempre té un -1 a la posició [3,2].

## Clip Space

Sistema de coordenades de retallat

- Si el punt és interior al frustum:
  - $-w \leq x \leq w$
  - $-w \leq y \leq w$
  - $-w \leq z \leq w$
- Si la càmera és perspectiva, llavors:
  - $w = -Ze$

## Perspective division

Simplement és el pas de coordenades homogènies a 3D. Es divideix cada coordenada per la coord homogènia.

$$(x_c, y_c, z_c, w_c) \longrightarrow (x_c/w_c, y_c/w_c, z_c/w_c)$$

## Normalized Device Space

- Si un punt és interior al frustum:
  - $1 \leq x \leq 1$
  - $-1 \leq y \leq 1$
  - $-1 \leq z \leq 1$

- Els punts de Znear són -1 i els Zfar són 1

## Viewport transform & depth transform

- Viewport transformation
  - Transformació mon-dispositiu
  - Depèn del viewport amb glViewport
- Depth range transformation
  - Depèn de l'interval definit amb glDepthRange (per defecte [0,1])

## Window space

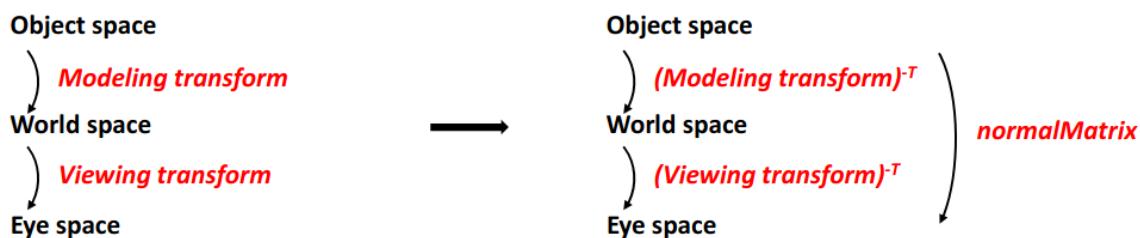
Coordenades de finestra/dispositiu

- Si el punt és interior al frustum:
  - $0 \leq x \leq w$
  - $0 \leq y \leq h$
  - $0 \leq z \leq 1$
- Els punts situats sobre znear tenen:
  - $z = 0$
- Els punts situats sobre zfar tenen:
  - $z = 1$
- Quan més endavant es generin fragments,  $gl_FragCoord.w = 1/w = -1/z$

## Transformació vectors

No és pràctic passar els vectors d'un espai de coordenades a un altre (perquè la component homogènia és 0). Per això, és millor passar dos punts del vector i calcular-ho després.

## Transformació normals



$\text{normalMatrix} = \text{transpose}(\text{inverse}(\text{modelViewMatrix}))$  (de la submat 3x3 de la mvM)

# Pipeline OpenGL

## 1. Dibuix de primitives

Les primitives (punt, línies, polígons...) es poden pintar:

- Vèrtex a vèrtex: glBegin, glVertex, glEnd (compatibility profile)
- Vertex Array Object (VAO): glDrawArrays, glDrawElements

## 2. Per-vertex operations

Es transformen els vèrtexs (modelview i projection)

Es transformen les normals (Amb la transposta de l'inversa de la submatriu 3x3 de la modelView) i es pot calcular la il·luminació del vèrtex.

## 3. Primitive Assembly

Els vèrtexs s'agrupen formant primitives.

Cada primitiva requereix un clipping diferent.

(GL\_POINTS, GL\_LINES, GL\_TRIANGLES)

## 4. Primitive Processing

1. Clipping (retallat) de la piràmide de visió. Si hi ha triangles que tenen vèrtexs dins i vèrtexs fora, se'n generen de nous.
2. Divisió de perspectiva: es divideix (x,y,z) per w
3. Viewport & Depth transform ---> window coordinates
4. Backface culling
  1. glEnable(GL\_CULL\_FACE)<- Es pot activar amb aquesta instrucció (per defecte desactivat)

## 5. Rasterització

1. Generació dels fragments corresponents a la primitiva retallada.
2. Cada fragment té diversos atributs:
  - Coordenades (window space)
  - Color (interpolat)
  - Coordenades de textura (interpolades)

## 6. Fragment processing ("shading")

Càcul del color del fragment (i.e. texture mapping)

## 7. Per-fragment operations ("raster operations")

Stencil test, Pixel ownership test (si pertany a la app o no, la calculadora pot estar per sobre de la nostra finestra opengl), Scissor test, Alpha test, Stencil test (vist més endavant, útil per les màscares), Depth test (test Z-buffer, s'escriu a gl\_FragDepth), Blending (veure mes endavant, alpha blending), Dithering, Logical Ops (glLogicOp)

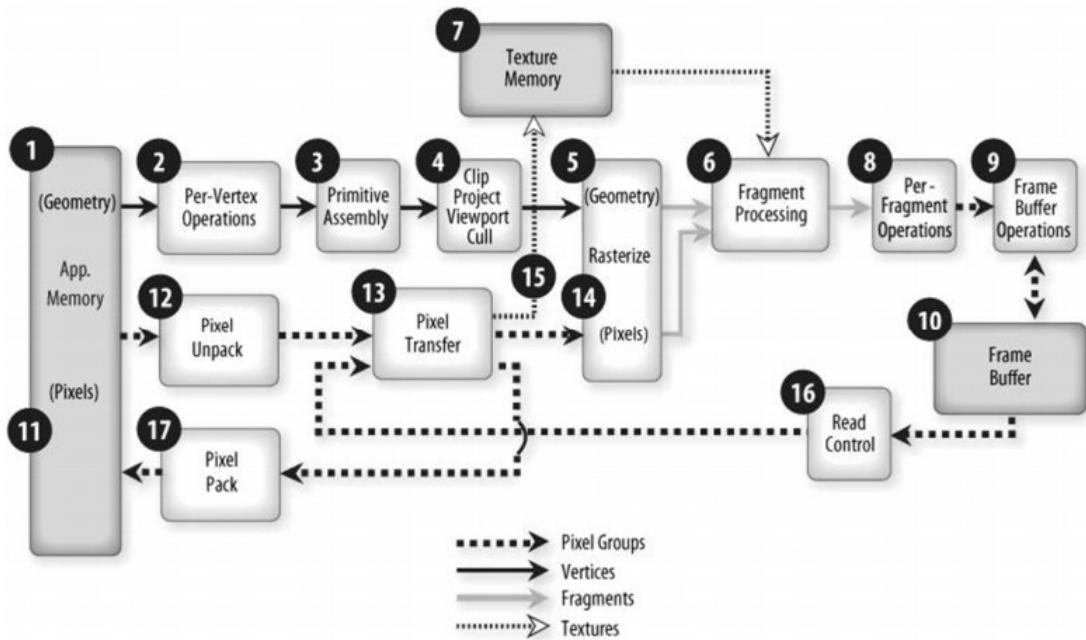
## 8. Frame buffer operations

Es modifiquen els buffers que s'hagin escollit amb glDrawBuffers, afectada per glColorMask, glDepthMask...

```
glColorMask(GL_TRUE,GL_TRUE,GL_TRUE,GL_TRUE)
```

```
//glColorMask(R,G,B,alpha) 
```

```
glDepthMask(GL_TRUE)
```



## Textures

Una textura és una taula de 1/2/3 dimensions, on cada cel·la emmagatzema una certa propietat amb 1/2/3/4 canals.

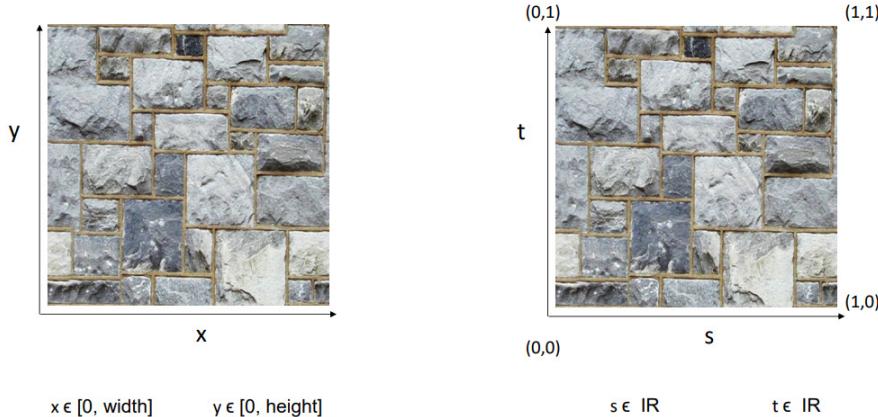
Habitualment les textures s'utilitzen al FS, tot i que també es poden usar en altres shaders.

- Kd (color map) -> (R,G,B)
- Ks (gloss map) ->(Rs,Gs,Bs) o simplificat -> Ks [0,1]
- Opacity (opacity map, alpha mask)
- Normal (normal map) -> 3 canals (nx,ny,nz)
- Desplaçament (bump mapping)
- Desplaçament (displacement mapping)

## Mida d'una textura

Número de texels en cada dimensió

Habitualment w, h són potència de 2.



## Mapping

- **Forward Mapping**-> funcions que ens diuen on ha d'anar cada texel en espai de pantalla (donat texel -> retornen pixel)
- **Inverse Mapping**-> funcions que ens diuen, donat un fragment, quina posició de la textura agafem el color. (donat pixel -> retornen texel)

Va millor Inverse Mapping en el paradigma d'OpenGL

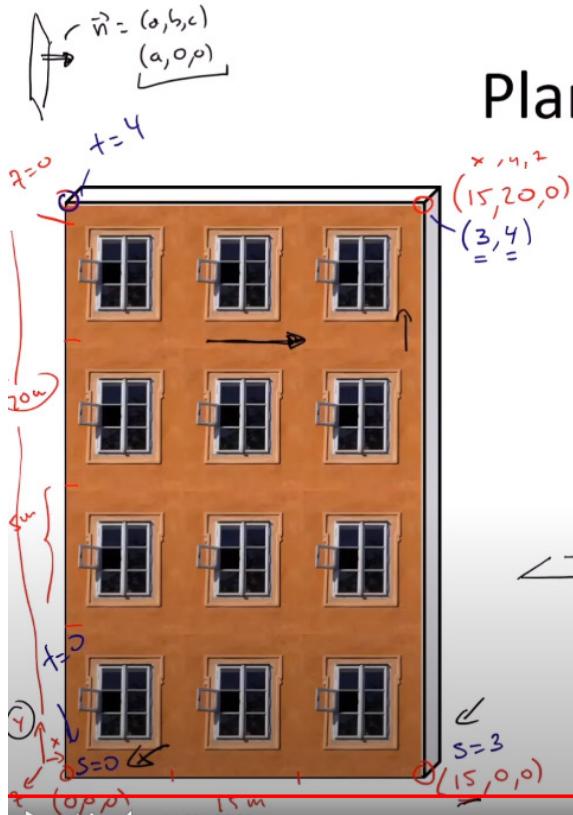
## Opcions per a generar coord de textura

- Modeling tool (Blender)
- Application (CPU)
- VS
- FS

# Mètodes per generar coords de textura

- Bàsics
    - Amb plans S,T
    - Amb superfície auxiliar (S-mapping, O-mapping)
  - Avançats (mesh parameterization)
    - Mesh partition
    - Area-preserving, Angle-preserving, Strech-preserving...

## Plans S,T



## Plans S,T (exemple)

$$S = ax + \underline{by} + \underline{cz} + d^{\cancel{w}} = 0$$

$x=0, S=0 \rightarrow 0 = a0 + d \rightarrow d = 0$

$x=15, S=3 \rightarrow \underline{3} = a\underline{15} \rightarrow a = \underline{3}/\underline{15} = 1/5$

Plan  $S \rightarrow (a, b, c, d) \rightarrow (\underline{1/5}, 0, 0, 0)$

$$\overline{p_{\text{lawo}}} \rightarrow (0, b, 0)$$

$$4 = 206 \rightarrow b = \frac{1}{5}$$

$s = ax+by+cz+dw \rightarrow (a,b,c,d).(x,y,z,w)$  dot product

$$x = 15, s = 3 \rightarrow 3 = a \cdot 15 \rightarrow a = 3/15 = \frac{1}{5}$$

d'igual manera es fa amb el pla T

## Parametrització amb superfície auxiliar

### Mapping esfèric

$\text{pas } (s,t) \rightarrow (\Theta, \Psi)$

$$\Theta = 2\pi s;$$

$$\Psi = \pi(t-0.5);$$

pas esfèriques  $\rightarrow (x,y,z)$

$$x = \sin(\Theta)\cos(\Psi);$$

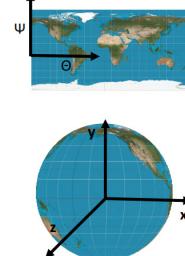
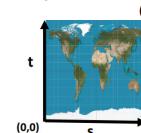
$$y = \sin(\Psi);$$

$$z = \cos(\Theta)\cos(\Psi);$$

### Exemple 1: Mapping esfèric

Input:  $s \in [0,1], t \in [0,1]$   
Output:  $x,y,z \in \text{esfera unitat}$   
 $\Theta = 2\pi s;$   
 $\Psi = \pi(t-0.5);$

// pas esfèriques  $\rightarrow (x,y,z)$   
 $x = \sin(\Theta)\cos(\Psi);$   
 $y = \sin(\Psi);$   
 $z = \cos(\Theta)\cos(\Psi);$



### Mapping cilíndric

$\text{pas } (s, t) \rightarrow (\Theta, h)$

$$\Theta = 2\pi s;$$

$$h = t;$$

pas cilíndriques  $\rightarrow (x,y,z)$

$$x = \sin(\Theta);$$

$$y = h;$$

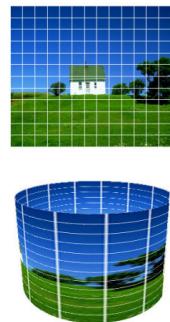
$$z = \cos(\Theta);$$

### Exemple 2: Mapping cilíndric

Input:  $s \in [0,1], t \in [0,1]$   
Output:  $x,y,z \in \text{cilindre } r=1 \text{ sobre pla XZ}$

// pas  $(s, t) \rightarrow (\Theta, h)$   
 $\Theta = 2\pi s;$   
 $h = t;$

// pas cilíndriques  $\rightarrow (x,y,z)$   
 $x = \sin(\Theta);$   
 $y = h;$   
 $z = \cos(\Theta);$



41

## Creació d'una textura

```
// Load Texture (once) QImage img0("fieldstone.png");
QImage T = img0.convertToFormat(QImage::Format_ARGB32);
glGenTextures( 1, &textureId0); glBindTexture(GL_TEXTURE_2D, textureId0);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, T.width(), T.height(), 0, GL_RGBA,
GL_UNSIGNED_BYTE, T.bits()); ...
// Bind textures, set uniforms...
g.glActiveTexture(GL_TEXTURE0);
g glBindTexture(GL_TEXTURE_2D, textureId0);

program->bind();
program->setUniformValue("colorMap", 0);
```

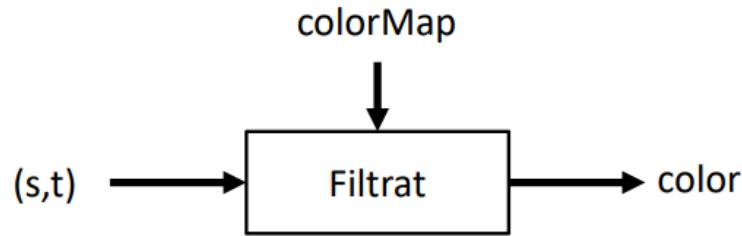
Important: Aquests dos números han de ser els mateixos

## Ús de textura al fragment shader

```
uniform sampler 2D colorMap;
in vec2 vtexCoord;
```

```
vec4 color = texture(colorMap, vtexCoord);
```

## Filtrat



Ideal: color d'un pixel -> color de la seva preimatge a la textura

- Magnification -> la preimatge és menor a un texel. Quan ens movem un pixel en la preimatge, en la textura ens movem menys d'un texel.
- Minification-> la preimatge és més gran que un texel. Quan ens movem un pixel en la preimatge, en la textura ens movem més d'un texel.

## Magnification or minification?

-Aproximació que fa OpenGL

Siguin  $s(x,y)$ ,  $t(x,y)$  les coordenades s,t del fragment (x,y).

Derivades parcials de  $s(x,y)$ :

Derivades parcials de  $s(x,y)$ :

$$\begin{aligned}\frac{\partial s}{\partial x} &\approx s(x+1, y) - s(x, y) \\ \frac{\partial s}{\partial y} &\approx s(x, y+1) - s(x, y)\end{aligned}$$

En GLSL es poden calcular amb  $dFdx$ ,  $dFdy$ :

$$\begin{aligned}\frac{\partial s}{\partial x} &\approx dFdx(\text{texCoord}.s) \\ \frac{\partial s}{\partial y} &\approx dFdy(\text{texCoord}.s)\end{aligned}$$

$dFdx$ ,  $dFdy$ .

Exemples:

- Mapping 1:1  
En aquest cas un píxel correspon a un texel

$$\delta u / \delta x = \delta v / \delta y = 1 \quad \delta v / \delta x = \delta u / \delta y = 0$$

- Magnification x2

$$\delta u / \delta x = \delta v / \delta y = 1/2 \quad \delta v / \delta x = \delta u / \delta y = 0$$

Per cada pixel en la imatge, en la textura ens movem  $\frac{1}{2}$  texel

- Minification x2

En aquest cas un píxel correspon a dos texels:

$$\delta u / \delta x = \delta v / \delta y = 2 \quad \delta v / \delta x = \delta u / \delta y = 0$$

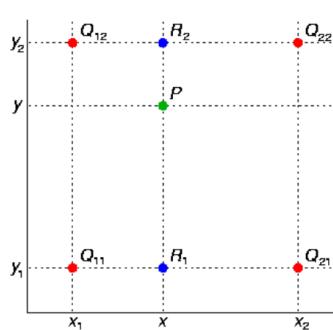
- Anisotòpic

Direcció H -> magnification

Direcció V -> minification

## Magnification filters

- GL\_NEAREST: nearest neighbor sampling  
El color d'aquesta mostra és el color del **veí més proper**.  
`glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);`
- GL\_LINEAR: interpolació lineal  
El color és una mitjana ponderada dels colors dels **quatre** veïns més propers.  
`glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);`



$$f(x, y) \approx \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y) + \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y_2 - y) + \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y - y_1) + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y - y_1).$$

## Minification filters

- Mipmapping

Idea bàsica: cada textura està representada amb diferents resolucions (level-of-details, **LOD**). Amb primitives texturades per les que cal un factor important de minification és molt significativa la millora.

En alguns casos el **LOD** més adient  $\lambda$  és fàcil de calcular.

Minification	$\partial u/\partial x, \partial v/\partial y$	LOD
1x	1	0
2x	2	1
4x	4	2
8x	8	3
$2^\lambda x$	$2^\lambda$	$\lambda$

En aquest cas  $2^\lambda = \partial u/\partial x$   
Per tant:  $\lambda = \log_2(\partial u/\partial x)$

En general:

- Es calcula un valor rho=f( $\partial u/\partial x, \partial v/\partial x, \partial u/\partial y, \partial v/\partial y$ )
- Es calcula el  $\lambda = \log_2(\rho)$

#### Without mipmapping

- GL\_NEAREST // Nearest neighbor sampling on LOD 0
- GL\_LINEAR // Bilinear interpolation on LOD 0

#### With mipmapping

- GL\_NEAREST\_MIPMAP\_NEAREST // Nearest neighbor sampling on LOD int( $\lambda$ )
- GL\_LINEAR\_MIPMAP\_NEAREST // Bilinear sampling on LOD int( $\lambda$ )
- GL\_NEAREST\_MIPMAP\_LINEAR //  $c_0$  = nearest neighbor on LOD int( $\lambda$ )  
//  $c_1$  = nearest neighbor on LOD int( $\lambda+1$ )  
// mix( $c_0, c_1, \text{fract}(\lambda)$ )
- GL\_LINEAR\_MIPMAP\_LINEAR //  $c_0$  = bilinear sampling on LOD int( $\lambda$ )  
//  $c_1$  = bilinear sampling on LOD int( $\lambda+1$ )  
// mix( $c_0, c_1, \text{fract}(\lambda)$ )

GL\_<samplingWithinTheLODs>\_MIPMAP\_<oneOrTwoLODs>

Per a un determinat fragment, les derivades parcials de les coordenades de textura, un cop multiplicades per la mida de la textura, tenen aquests valors:  
 $\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} = 4$ ;  $\frac{\partial u}{\partial y} = \frac{\partial v}{\partial x} = 0$ . Quin és el LoD més adient per a accedir a textura per a aquell fragment?

[Cast]

Trieu-ne una:

- 1
- 4
- No vull contestar la pregunta
- 3
- 2 ✓

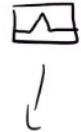
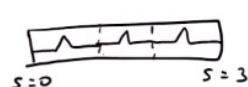
## Wrapping

Quins colors agafem quan acaba la textura? (MAGNIFICATION BILINEAL)

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, mode);  
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, mode);
```

$glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, mode)$

$GL_REPEAT \rightarrow s = \frac{\text{fract}(s)}{T}$



$GL_CLAMP_TO_EDGE$

$[0, 1]$



$\left[\frac{1}{2N}, 1 - \frac{1}{2N}\right]$

`GL_REPEAT, GL_CLAMP_TO_EDGE`

`GL_REPEAT` -> Només utilitza la part fraccionaria.

## Combinació

Mètodes habituals

- Replace:

```
fragColor = texColor;
```

Agafem directament el color de la textura, suposem que ja tenim la il·luminació calculada.

- Modulate:

```
fragColor = texColor * frontColor;
```

Incorporem il·luminació amb el frontColor.

- Decal

```
fragColor = mix(frontColor, texColor, texColor.a)
```

`texColor.a` ----> 0 -> frontColor

`texColor.a` ----> 1 -> texColor

## Perspective-correct interpolation

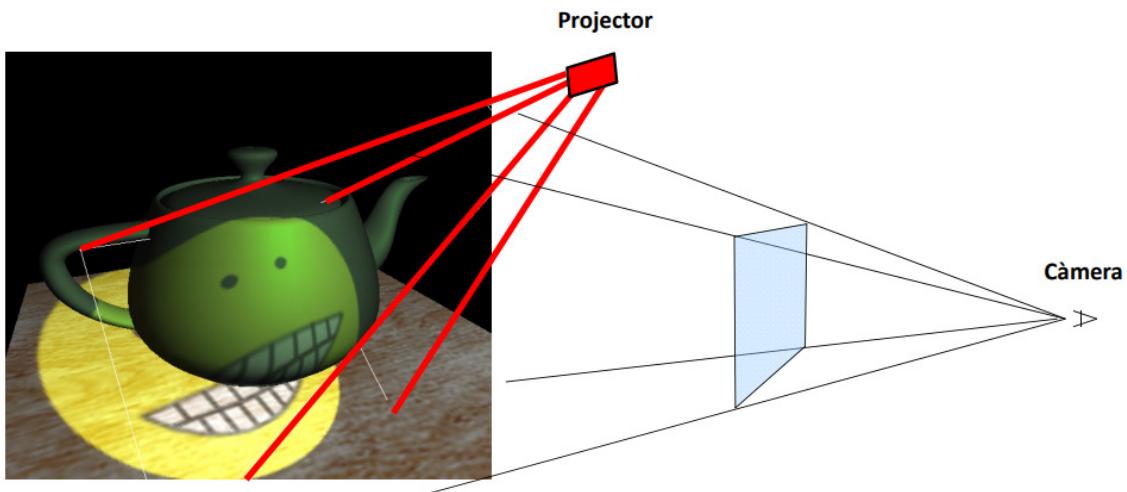
En **object space** les coords de textura varien linealment.

En **window space** les coords de textura no varien linealment degut a la divisió de perspectiva.

Solucions:

- Interpolem ( $s, t$ ) en object space (també valdria en world, eye i clip space, perquè són abans de la div de perspectiva)
- Interpolem ( $sw, tw, w$ ) en window space, obtenim un texel ( $s, t, q$ ); accedirem a la textura amb  $(s/q, t/q)$ . Recordeu que  $w = 1/wc = -1/ze$

## Projective Texture Mapping



La idea principal és “projectar” la textura.

VS-generació coords de textura

Passa el vèrtex de object space a window space (viewport 1x1) però sense aplicar la div de perspectiva.

Calcula  $(s,t,p,q)$  com  $(x,y,z,w)$  del resultat

$$(s,t,p,q) = T(0.5)*S(0.5)*Pp*Vp*M*(x,y,z,w)$$

FS-accés a textura

Usa  $(s/q, t/q)$  per a accedir a la textura (si no no interpolaria bé)

## Aplicacions

Tècnica	Info a la textura	Ús de la textura	Coords de textura	View parallax	Self-occlusion	Detailed silhouette	On s'aplica	
Color mapping	RGB 3	Kd del material	(s,t)	-	-	-	FS	
Bump mapping	D 1		Modificar la normal	(s,t)	N	N	N	FS
Normal mapping	Normal 3		Modificar la normal	(s,t)	N	N	N	FS
Parallax mapping	Normal + D 3+1 o 4	Modificar la normal	(s+d <sub>s</sub> , t+d <sub>t</sub> )	S	N	N	FS	
Relief mapping	Normal + D 3+1 o 4	Modificar la normal; descartar fragments	(s+d <sub>s</sub> , t+d <sub>t</sub> )	S	S	S	FS!!!	
Displacement mapping	D 1	Desplaçar els vèrtexs un cop subdividits els polígons.	(s,t)	S	S	S	CPU GS TCS+TES	

### Color mapping

Kd del material (RGB3)

S'aplica al FS

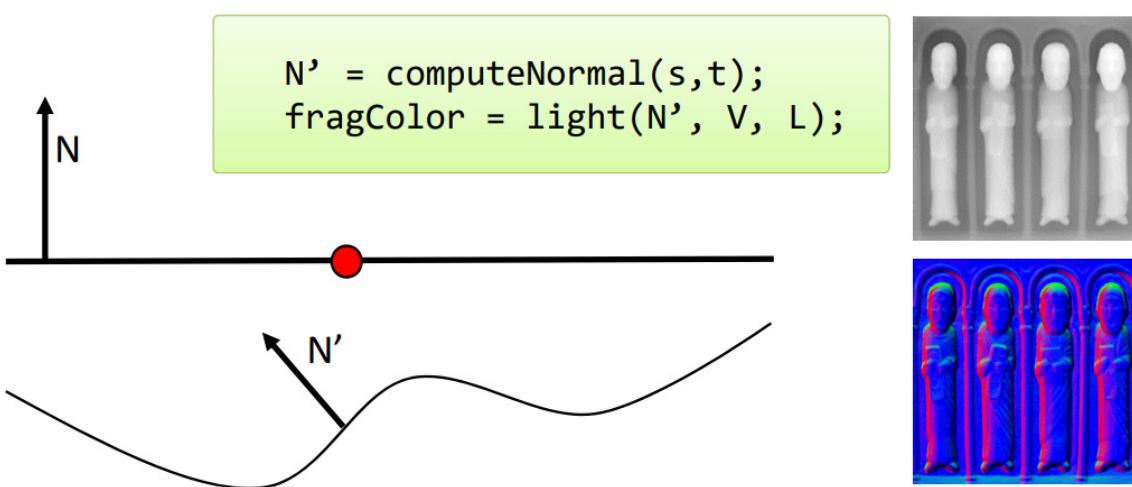
### Bump mapping / Normal mapping

Bump mapping -> codifiquem el desplaçament respecte a la superfície.

Normal mapping -> codifiquem la normal (3 valors).

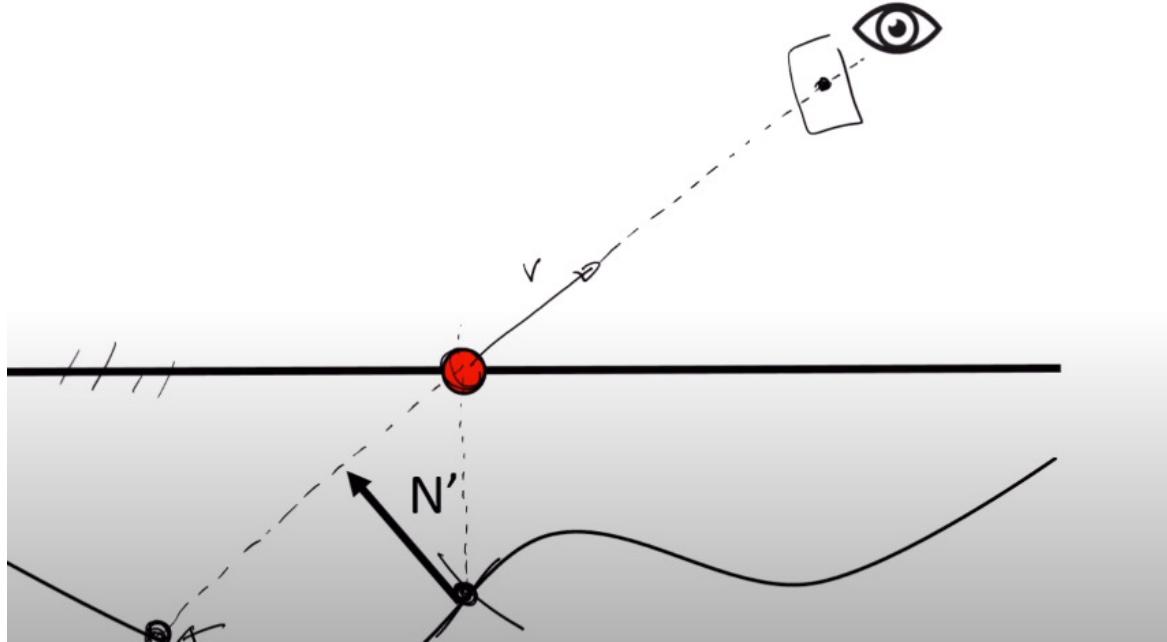
Textura 1 canal: guardem heightField

Textura 2 canals: guardem gradient de F(u,v): dF/du, dF/dv



## Problema Bump mapping / Normal mapping

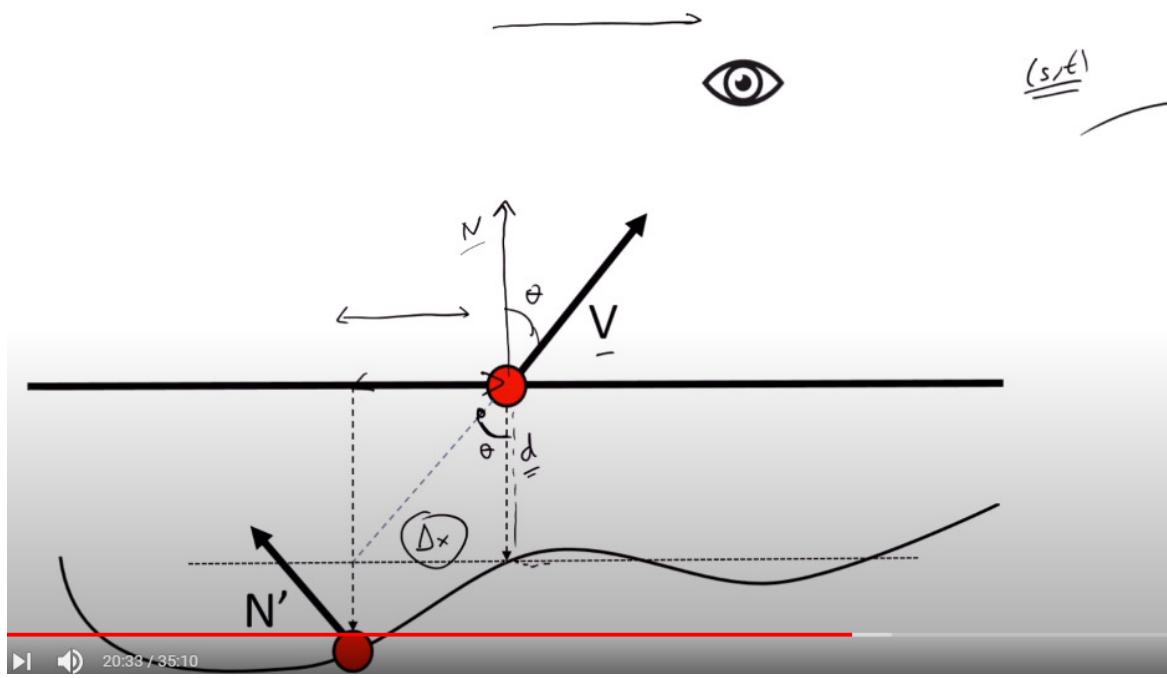
problema amb les vistes tangencials



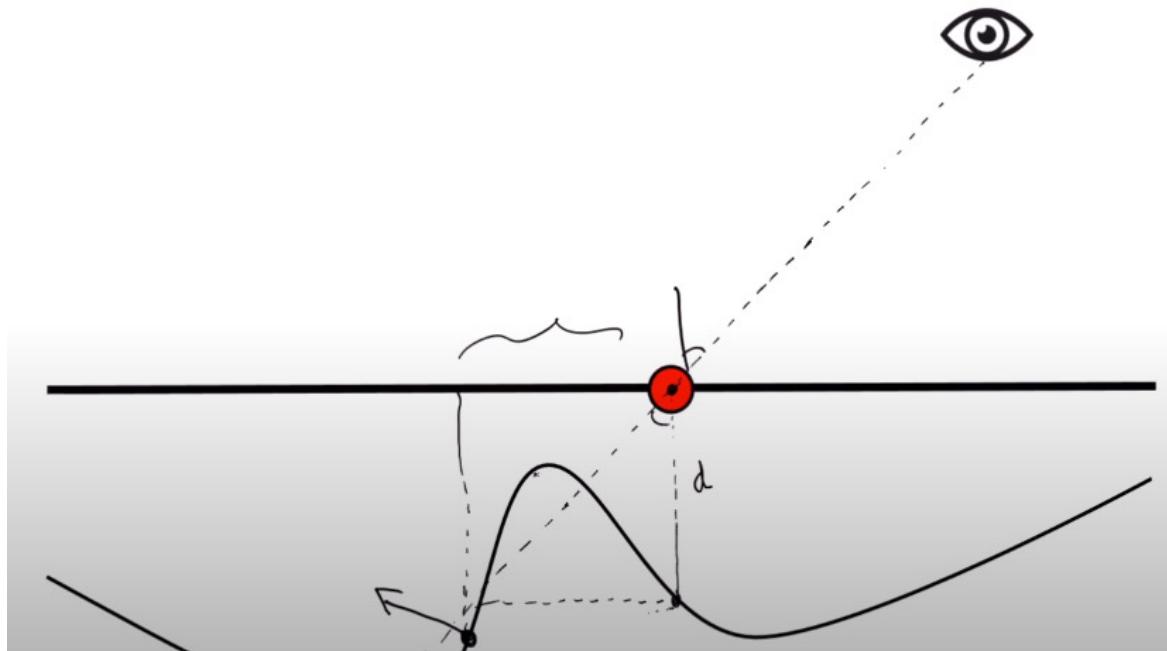
## Parallax mapping

Normal+Desplaçament. S'aplica al FS

Modifiquem la normal ( $s+ds$ ,  $t+dt$ )



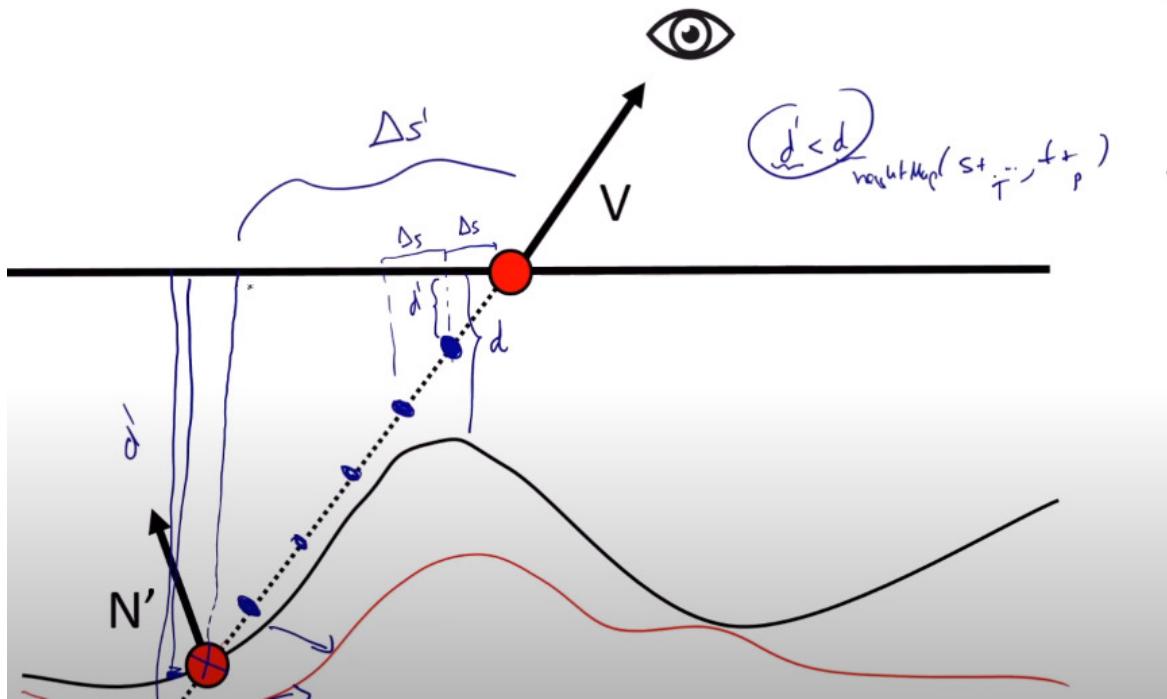
## Problema Parallax mapping



## Relief mapping

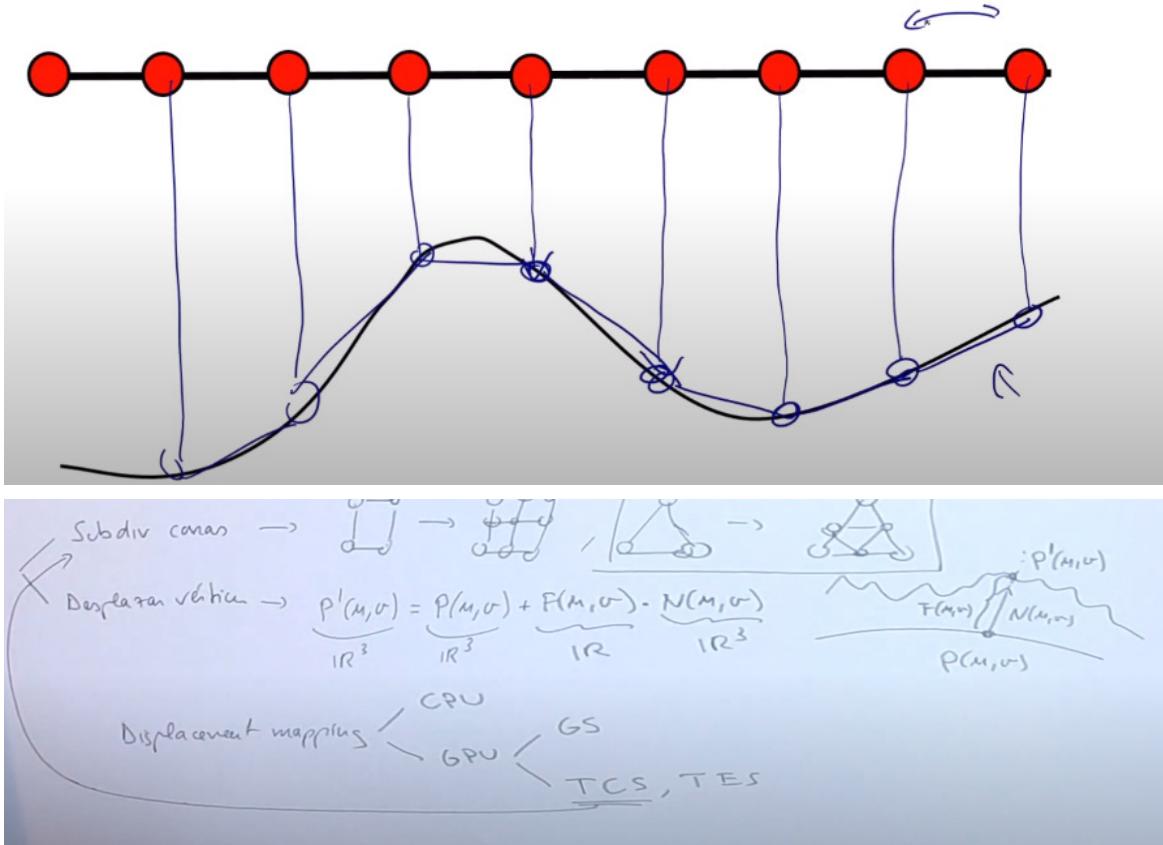
Normal+Desplaçament. S'aplica al FS.

Modificar la normal; descartar fragments. Detecta auto-occlusions, alt overhead.



## Displacement mapping

Desplaçament respecte a la superfície. Desplaçar els vèrtexs un cop subdividits els polígons.



Tessellation control / evaluation shaders

## Bump mapping

No modifiquem els vèrtexs (NO subdivisió, NO desplaçament), sinó la normal per la il·luminació.

$$\mathbf{N}'(u,v) = \frac{\partial \mathbf{P}'(u,v)}{\partial u} \times \frac{\partial \mathbf{P}'(u,v)}{\partial v}$$

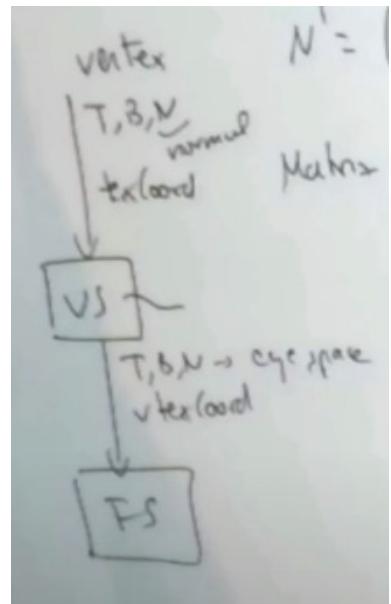
$$N' = N - \frac{\delta F}{\delta u} T - \frac{\delta F}{\delta v} B$$

(T,B,N) -> espai tangent

$$N' = \left( -\frac{\delta F}{\delta u}, -\frac{\delta F}{\delta v}, 1 \right)$$

Matriu per passar de tangent space a eye space

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 0 & N & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{eye space}} \circ \underbrace{\begin{bmatrix} N' \end{bmatrix}}_{\text{tangent}} = \underbrace{\begin{bmatrix} N' \end{bmatrix}}_{\text{eye space}}$$



BumpMap  $\rightarrow F(u, v)$  1 canal

$$\begin{aligned}
 \bar{F} &= \underline{\text{texture(bumpmap, (s, t))}} \cdot x \\
 \bar{F}_x &= " " \quad " \quad (s+E, t) \cdot x \\
 \bar{F}_y &= " " \quad " \quad (s, t+E) \cdot x \\
 \nabla \bar{F} &= (\bar{F}_x - \bar{F}, \bar{F}_y - \bar{F}) / \epsilon \cdot \underline{\text{scale}} \\
 N' &= (\nabla \bar{F} \cdot x, \nabla \bar{F} \cdot y, 1) \\
 \underline{\underline{N'}} &= \underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 0 & N & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{eye space}} \underbrace{\begin{bmatrix} N' \end{bmatrix}}_{\text{tangent}} = \underline{\underline{B}}
 \end{aligned}$$

$\text{fragColor} = \text{phong}(\underline{\underline{N'}}, \dots)$

## Normal mapping

Guarda la normal en tangent space.

Normal Map ->  $(-\frac{\delta F}{\delta u}, -\frac{\delta F}{\delta v}, 1)$  3 canals

$N' = \text{texture(normalMap, (s,t))}$

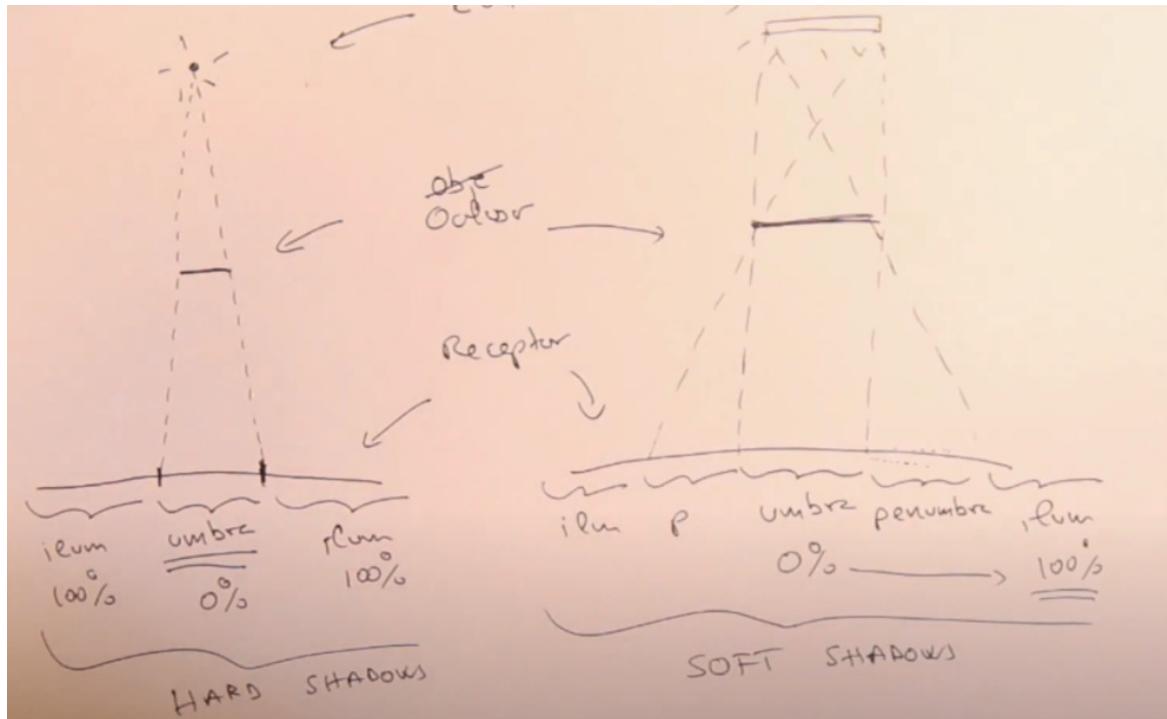
Gram-Schmidt

$N' = N$

$T' = T - (N \cdot T)N$

$B' = B - (N \cdot B)N - (T' \cdot B)T'$

## Ombres



### Propietats

- Si la llum és puntual -> no hi ha penombra
- Si augmenta la mida de la font de llum
  - Augmenta la penombra
  - Disminueix la umbra (pot ser nul·la)
- Si apropeix oclusor i receptor
  - Disminueix la penombra.

## Ombres per projecció sense stencil

Precondició:

- Receptor és pla.
- Llum puntual.

Passos:

1. Dibuixar receptor
2. Dibuixar l'emissor projectat (ombra)
3. Dibuixar emissor

```

// 1. Dibuixar receptor
dibuixa(receptor)

// 2. Dibuixar l'emissor projectat (ombra)
glDisable (GL_LIGHTING);
glDisable (GL_DEPTH_TEST);
glMatrixMode (GL_MODELVIEW);
glPushMatrix ();
glMultMatrixf (MatriuProjeccio);
dibuixa (oclusor);
glPopMatrix ();

// 3. Dibuixar emissor
glEnable (GL_LIGHTING);
glEnable (GL_DEPTH_TEST);
dibuixa (emissor);

```



Aquesta solució pot tenir problemes de Z-fighting

## Evitar problemes de Z-fighting

S'utilitza `glPolygonOffset( factor, units )`:

`glPolygonOffset( factor, units )`

- Efecte: abans del depth test, es modifica el valor de la z del fragment (per defecte en [0,1]), amb l'equació

$$z' = z + \partial z \cdot \text{factor} + r \cdot \text{units}$$

on

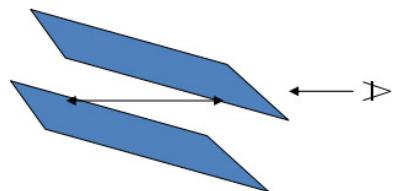
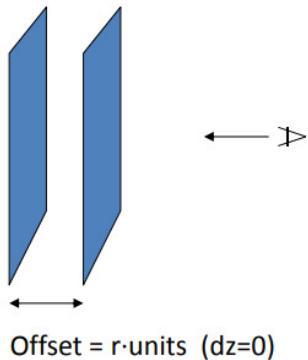
$$\partial z = \max( \partial z / \partial x, \partial z / \partial y )$$

$r$  = valor més petit tal que garantitza un offset  $> 0$

- El paràmetre *factor* permet introduir un **offset variable** (depén de la inclinació del polígon)
- El paràmetre *units* permet introduir un **offset constant**

Exemples:

$$\text{offset} = dz \cdot \text{factor} + r \cdot \text{units}$$



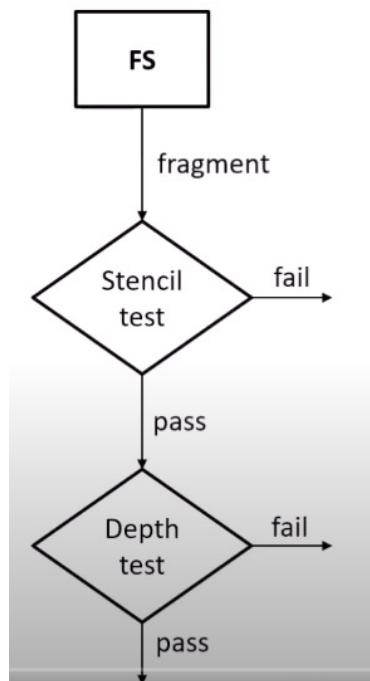
Valors típics: glPolygonOffset(1,1);

Offset positiu -> increment de la z (en window coordinates) -> es calcula la z com si estigués més lluny.

## Stencil Buffer

És una “màscara”. Stencil buffer guarda, per cada pixel, un enter entre 0.. $2^n-1$

- Demanar una finestra OpenGL amb stencil:
  - QGLFormat f;
  - f.setStencil(true);
  - QGLFormat::setDefaultFormat(f);
- Obtenir el núm. de bits del stencil:
  - glGetIntegerv(GL\_STENCIL\_BITS, &nbits);
- Esborrar stencil (no li afecta glStencilFunc(), sí glStencilMask()):
  - glClearStencil(0);
  - glClear(GL\_STENCIL\_BUFFER\_BIT);



### Establir el test de comparació

- glEnable(GL\_STENCIL\_TEST);
- glStencilFunc(comparació, valorRef, mask)
  - Comparació pot ser: GL\_NEVER, GL\_ALWAYS, GL\_LESS...
  - Ex: GL\_LESS: (valorRef & mask) < (valorStencil & mask)

### Operacions a fer a stencil buffer segons el resultat del test

- glStencilOp(fail, zfail, zpass)
  - fail -> op. a fer quan el fragment no passa el test de stencil
  - Zfail -> op. a fer quan passa stencil, pero no passa z-buffer
  - Zpass -> op. a fer quan passa stencil i passa z-buffer
- Cadascú dels paràmetres anteriors pot ser:
  - GL\_KEEP, GL\_ZERO, GL\_INCR, GL\_DECR, GL\_INVERT
  - GL\_REPLACE (usa valor referencia)

### Ombres per projecció amb stencil

Precondició:

- Receptor és pla.
- Llum puntual.

Passos:

1. Dibuixar el receptor al color buffer i al stencil buffer.
2. Dibuixar oclusor per netejar l'stencil a les zones a l'ombra.
3. Dibuixa la part fosca del receptor.
4. Dibuixa l'occlusor.

```

// 1. Dibuixa el receptor al color buffer i al stencil buffer
glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_ALWAYS, 1, 1);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
dibuixa(receptor);

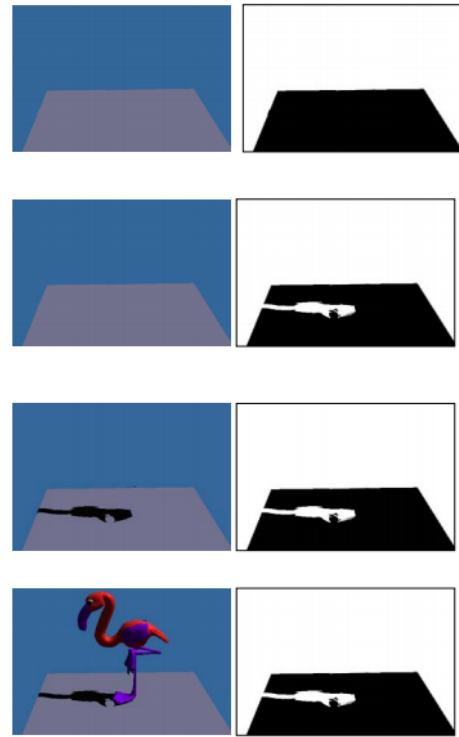
// 2. Dibuixa oclusor per netejar l'stencil a les zones a l'ombra
glDisable(GL_DEPTH_TEST);
glColorMask(GL_FALSE, ... GL_FALSE);
glStencilFunc(GL_EQUAL, 1, 1);
glStencilOp(GL_KEEP, GL_KEEP, GL_ZERO);
glPushMatrix(); glMultMatrixf(MatriuProjeccio);
dibuixa(oclusor);
glPopMatrix();

// 3. Dibuixa la part fosca del receptor
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LESS);
glColorMask(GL_TRUE, ... , GL_TRUE);
glDisable(GL_LIGHTING);
glStencilFunc(GL_EQUAL, 0, 1);
Dibuixa(receptor);

// 4. Dibuixa l'oclusor
glEnable(GL_LIGHTING);
glDepthFunc(GL_LESS);
glDisable(GL_STENCIL_TEST);
Dibuixa(oclusor);

```

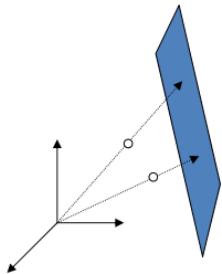
24



## Matrius de projecció

- Projecció respecte l'origen

Donats els coeficients ( $a, b, c, d$ ) d'un pla, la matriu és:

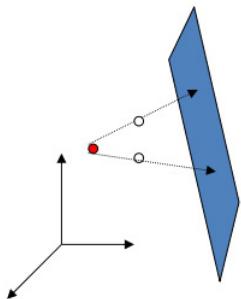


$$\begin{bmatrix} -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & -d & 0 \\ a & b & c & 0 \end{bmatrix}$$

- Projecció respecte punt  $(x, y, z)$

Donats els coeficients ( $a, b, c, d$ ) d'un pla, la matriu de projecció respecte un punt  $(x, y, z)$  és:

és:

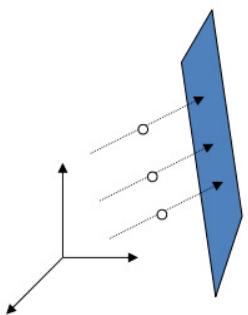


$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -(d+ax+by+cz) & 0 & 0 & 0 \\ 0 & -(d+ax+by+cz) & 0 & 0 \\ 0 & 0 & -(d+ax+by+cz) & 0 \\ a & b & c & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$-d - by - cz$	$xb$	$xc$	$xd$
$ya$	$-d - ax - cz$	$yc$	$yd$
$za$	$zb$	$-d - ax - by$	$zd$
$a$	$b$	$c$	$-ax - by - cz$

27

- Projecció en la direcció (x,y,z)  
Donats els coeficients (a,b,c,d) d'un pla, la matriu de projecció en la direcció del vector (x,y,z) és:



$$\begin{bmatrix} by + cz & -bx & -cx & -dx \\ -ay & ax + cz & -cy & -dy \\ -az & -bz & ax + by & -dz \\ 0 & 0 & 0 & ax + by + cz \end{bmatrix}$$

## Volums d'ombra

Precondició:

- Oclusor senzill
- Receptor arbitrari
- Llum puntual

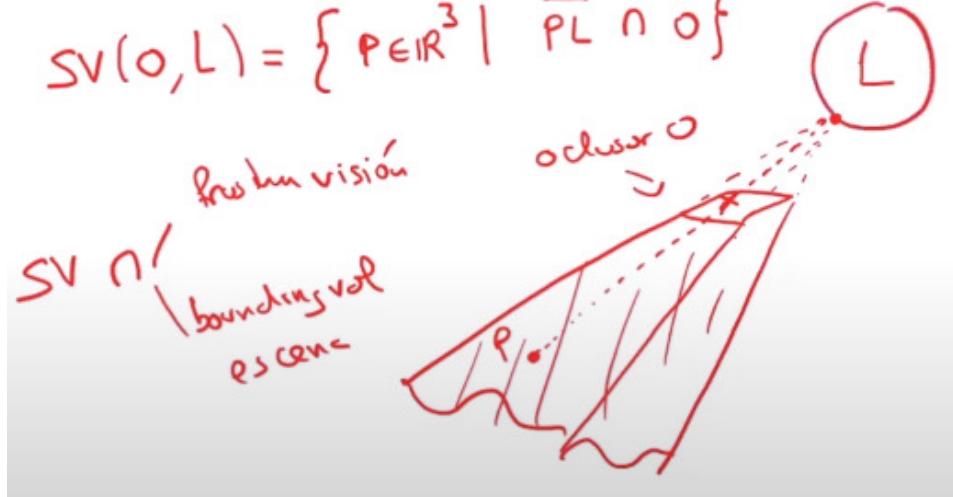
### PROBLEMA PUNT INTERIOR POLÍGON:

Llencem un raig en qualsevol direcció, si el número d'interseccions amb les arestes del polígon son:

- imparells, està dins;
- si és parell, està fora.

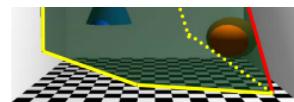
Shadow Volume  $\equiv$  SV

$$SV(O, L) = \{ P \in \mathbb{R}^3 \mid \overline{PL} \cap O \neq \emptyset \}$$

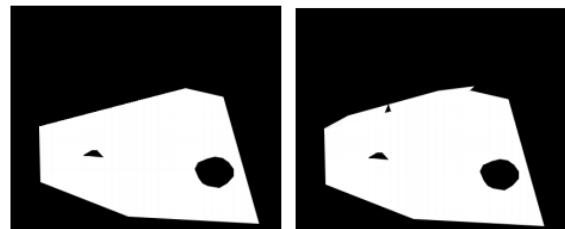


1. Dibuixa l'escena al z-buffer.
2. Dibuixa al stencil les cares frontals del volum.
3. Dibuixa al stencil les cares posteriors del volum.
4. Dibuixa al color buffer la part fosca de l'escena.
5. Dibuixem al color buffer la part clara de l'escena.
6. Restaura l'estat inicial.

```
// 1. Dibuixa l'escena al z-buffer
glColorMask(GL_FALSE, ..., GL_FALSE);
dibuixa(escena);
```



```
// 2. Dibuixa al stencil les cares frontals del volum
 glEnable(GL_STENCIL_TEST);
 glEnable(GL_DEPTH_TEST);
 glStencilFunc(GL_ALWAYS, 0, 0);
 glEnable(GL_CULL_FACE);
 glStencilOp(GL_KEEP, GL_KEEP, GL_INCR);
 glEnable(GL_CULL_FACE);
 dibuixa(volum_ombra);
```



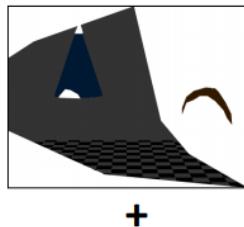
```
// 3. Dibuixa al stencil les cares posteriors del volum
 glStencilOp(GL_KEEP, GL_KEEP, GL_DECR);
 glEnable(GL_CULL_FACE(GL_FRONT));
 dibuixa(volum_ombra);
```



30

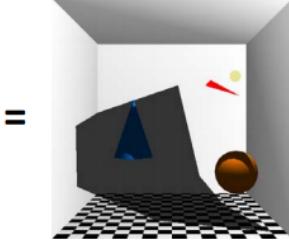
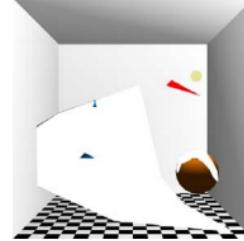
```
// 4. Dibuixa al color buffer la part fosca de l'escena
```

```
glDepthMask(GL_TRUE);
glColorMask(GL_TRUE, ... , GL_TRUE);
glCullFace(GL_BACK);
glDepthFunc(GL_LESS);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
glStencilFunc(GL_EQUAL, 1, 1);
glDisable(GL_LIGHTING);
dibuixa(escena);
```



```
// 5. Dibuixem al color buffer la part clara de l'escena
```

```
glStencilFunc(GL_EQUAL, 0, 1);
 glEnable(GL_LIGHTING);
dibuixa(escena);
```



```
// 6. Restaura l'estat inicial
```

```
glDepthFunc(GL_LESS);
glDisable(GL_STENCIL_TEST);
```

# front Face - # backFace > 0

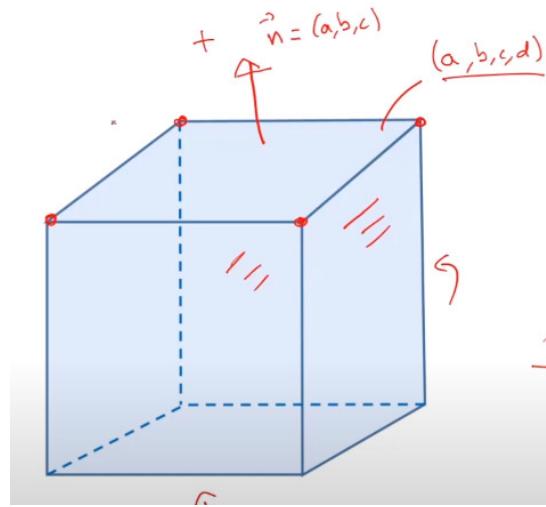
- P interior al SV
- P no il·luminat

## Frontface / Backface test

$$ax + by + cz + d = 0 \text{ (pla)}$$

$$\begin{aligned} \text{dist}(Q, \text{pla}) = aq + bq + cq + d &\longrightarrow \text{front face } (> 0) \\ &\longrightarrow \text{backface } (< 0) \end{aligned}$$

Q → llum, càmera.



## Shadow Mapping

Precondició:

- Oclusor arbitrari
- Receptor arbitrari

Passos:

1. Definir càmera situada a la font de llum.
2. Dibuixar l'escena.
3. Guardar el z-buffer en una textura.

### // Pas 1. Actualització del shadow map

#### // 1. Definir càmera situada a la font de llum

```
glViewport( 0, 0, SHADOW_MAP_WIDTH, SHADOW_MAP_HEIGHT );
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
gluPerspective( fov, ar, near, far); // de la càmera situada a la llum!
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
gluLookAt( lightPos, ..., lightTarget, ...., up,...);
```

#### // 2. Dibuixar l'escena

```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glPolygonOffset(1,1); glEnable(GL_POLYGON_OFFSET_FILL);
drawScene();
glDisable(GL_POLYGON_OFFSET_FILL);
```

#### // 3. Guardar el z-buffer en una textura

```
glBindTexture(GL_TEXTURE_2D, textureId);
glCopyTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, 0, 0, SHADOW_MAP_WIDTH,
SHADOW_MAP_HEIGHT);
```

<sup>45</sup> // Restaurar càmera i viewport

VS

$$\underbrace{T(0.5) \cdot S(0.5)}_{\text{light camera}} \cdot P \cdot V \cdot M \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} s \\ t \\ p \\ ? \end{bmatrix}$$

FS

shadowMap(s, t)

vertex  
(object space)

En el fragment shader obtenim que si:

- $\text{shadowmap}(s/q, t/q) \approx p/q$  està il·luminat.
- $\text{shadowmap}(s/q, t/q) < p/q$  està a l'ombra.

### // VS

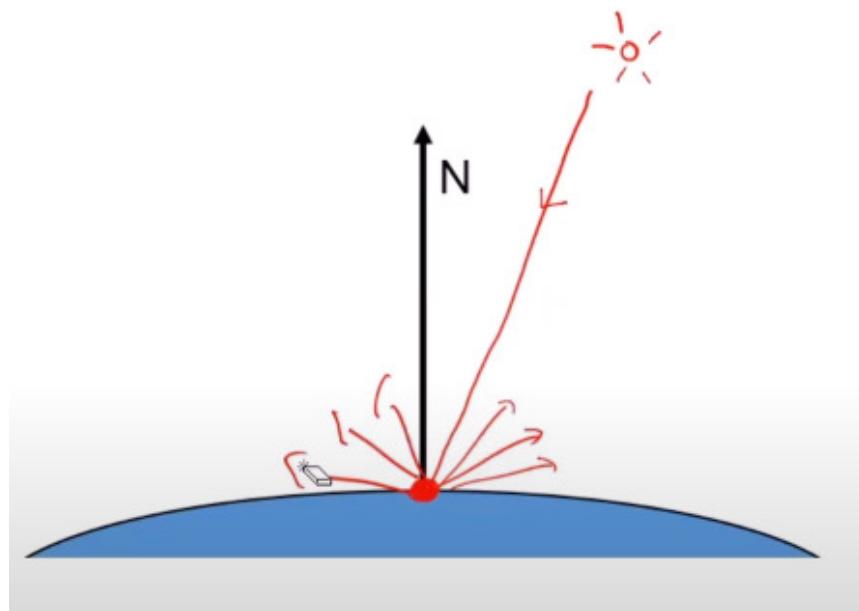
```
uniform mat4 lightMatrix;  
...  
  
void main()  
{  
    ...  
  
    gl_TexCoord[0] = lightMatrix * gl_Vertex;  
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;  
}
```

### // FS

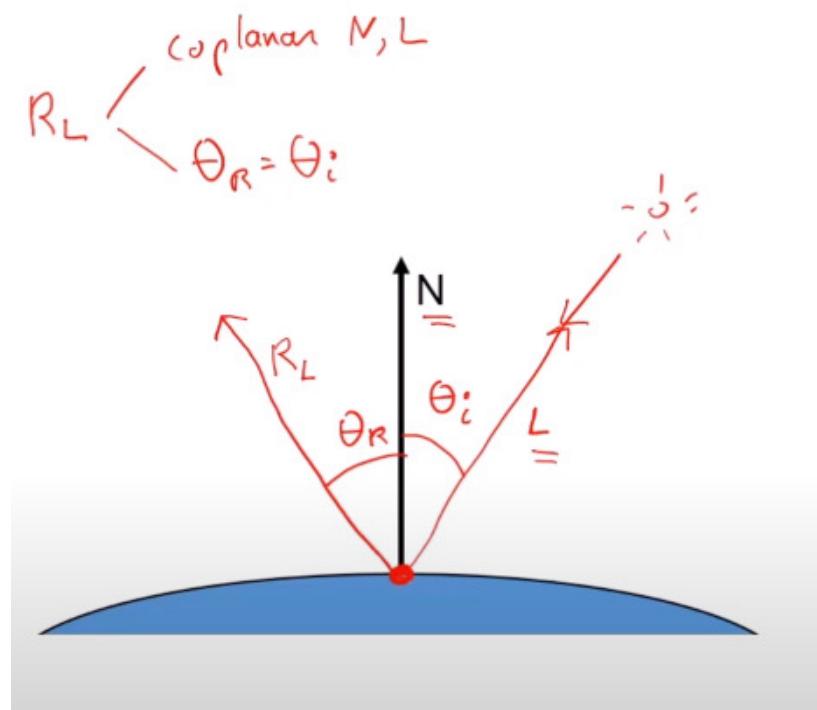
```
...  
  
vec2 st = gl_TexCoord[0].st / gl_TexCoord[0].q;  
float trueDepth = gl_TexCoord[0].p / gl_TexCoord[0].q;  
float storedDepth = texture2D(shadowMap, st).r;  
float bias = 0.01;  
if (trueDepth - bias <= storedDepth)  
    gl_FragColor = ... // iluminat  
else  
    gl_FragColor = ... // a l'ombra
```

## Reflexions especulars

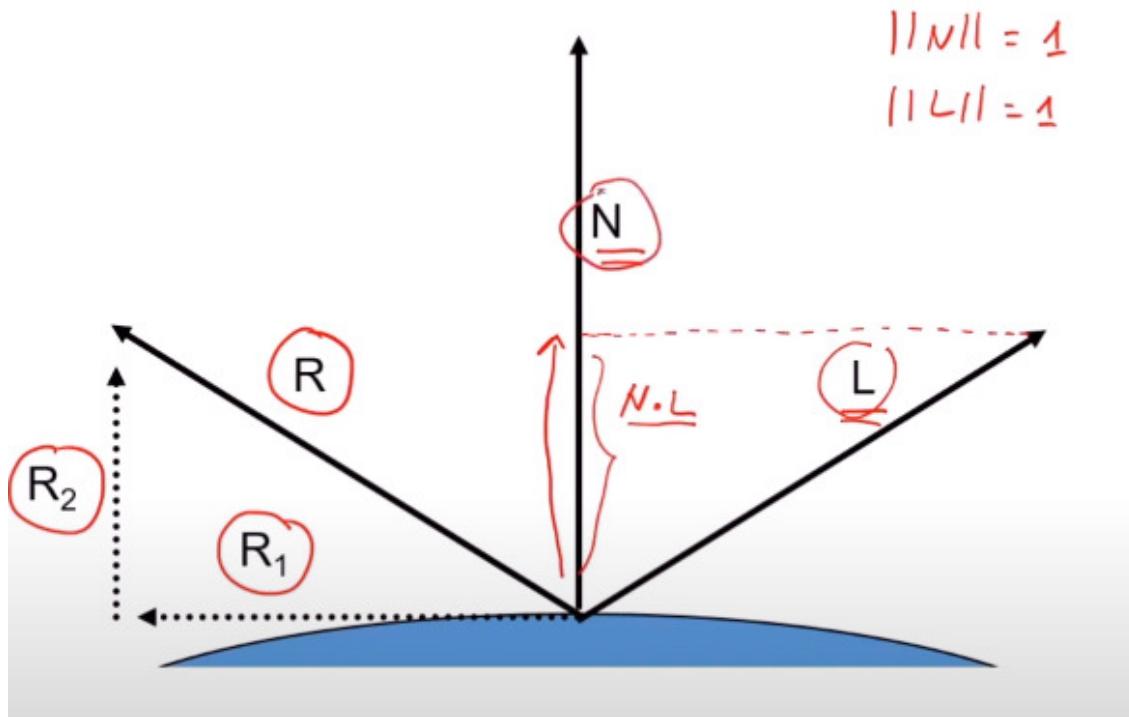
### Reflexió difosa



### Reflexió specular



## Vector reflectit



$$R = R_1 + R_2$$

$$R_1 = -L + R_2$$

$$R_2 = N(N \cdot L)$$

$$R = -L + R_2 + R_2 = -L + 2N(N \cdot L)$$

## Reflexions basada en objectes virtuals

Precondició: mirall pla

### Modelats

1. Dibuixar els objectes en posició virtual.
2. Dibuixar el mirall semitransparent.
3. Dibuixar els objectes en posició real.

Vèrtexs en CCW

Hem de reflexar la font de llum.

Assumeix que els objectes virtuals estan en el semiespai positiu del pla del mirall.

Assumeix que els objectes virtuals només es veuen a través del forat del mirall.

### // 1. Dibuixar els objectes en posició virtual

```
glPushMatrix();
glMultMatrix(matriu_simetria)
glLightfv(GL_LIGHT0, GL_POSITION, pos);
glCullFace(GL_FRONT);
dibuixar(escena);
glPopMatrix();
```

### // 2. Dibuixar el mirall semi-transparent

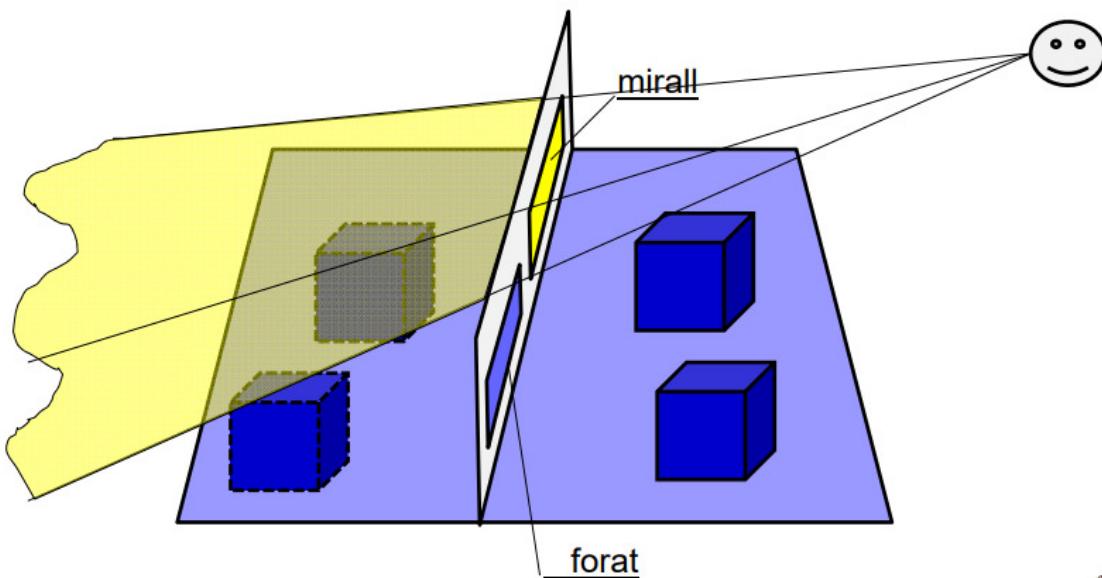
```
glLightfv(GL_LIGHT0, GL_POSITION, pos);
glCullFace(GL_BACK);
dibuixar(mirall);
```

### // 3. Dibuixar els objects en posició real

```
dibuixar(escena);
```

#### Reflectits sense stencil buffer

Dibuixar els objectes virtuals amb plans de retallat addicionals glClipPlane()



12

#### Reflectits amb stencil buffer

Usar stencil per limitar els objectes virtuals a la regió ocupada pel mirall.

1. Dibuixem el mirall a l'stencil buffer.
2. Dibuixem els objectes virtuals.
3. Dibuixem el mirall semitransparent
4. Dibuixem els objectes reals.

38

WUOLAH

```

// 1. Dibuixem el mirall a l'stencil buffer
glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_ALWAYS, 1, 1);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
glDepthMask(GL_FALSE); glColorMask(GL_FALSE...);
dibuixar(mirall);

// 2. Dibuixem els objectes virtuals
glDepthMask(GL_TRUE); glColorMask(GL_TRUE...);
glStencilFunc(GL_EQUAL, 1, 1);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
glPushMatrix(); glMultMatrix(matriu simetria)
glLightfv(GL_LIGHT0, GL_POSITION, pos);
glCullFace(GL_FRONT);
dibuixar(escena);
glPopMatrix();

// 3. Dibuixem el mirall semitransparent
glDisable(GL_STENCIL_TEST);
glLightfv(GL_LIGHT0, GL_POSITION, pos);
glCullFace(GL_BACK);
dibuixar(mirall);

```

## Textures dinàmiques

1. Dibuixar objecte en posició virtual.  
Crear una textura.
2. Dibuixar el mirall utilitzant la textura.  
`fragColor = texture(colorMap, gl_FragCoord.xy/SIZE)`
3. Dibuixar objecte en posició real.

```

GLdouble modelview[16];
glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
GLdouble projection[16]; glGetDoublev(GL_PROJECTION_MATRIX, projection);
GLint viewport[4] = {0, 0, 1, 1};
//glGetIntegerv(GL_VIEWPORT, viewport);

```

```

GLdouble s,t,r;
gluProject(x, y, z, modelview,
projection, viewport, &s, &t, &r)

```

### Matriu de reflexió

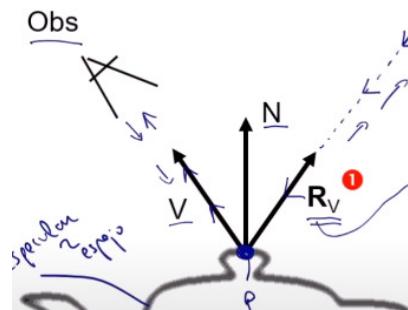
Matriu de reflexió respecte un pla (a,b,c,d):

$$\begin{bmatrix} 1 - 2a^2 & -2ba & -2ca & -2da \\ -2ba & 1 - 2b^2 & -2cb & -2db \\ -2ca & -2cb & 1 - 2c^2 & -2dc \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## Environment mapping

Funció (o textura) que, donada una direcció arbitrària  $R$ , ens retorna el color de l'entorn en direcció  $R$ .

color = environmentMap( $R$ )



**vs**

P, N en world space o eye space

**FS**

$V = \text{norm}(\text{Obs}-P)$

$R_v = 2(N \cdot V)N - V$

$R_v = \text{reflect}(-V, N)$  // Es una opció equivalent que ens dona glsl

fragColor = EnvMap( $R_v$ )

## Ús com a entorn (background)

Skymap

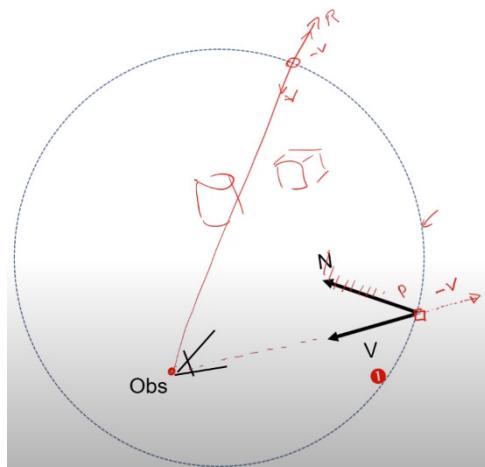
**vs**

P en world space o eye space

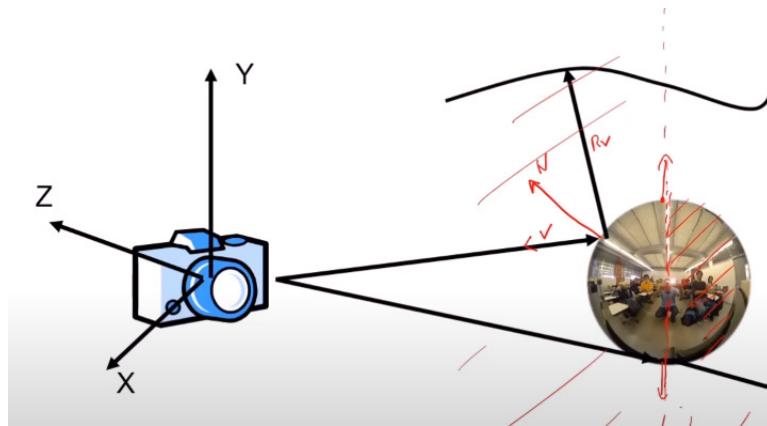
**FS**

$R = \text{norm}(P-\text{obs})$

fragColor = EnvMap( $R$ )

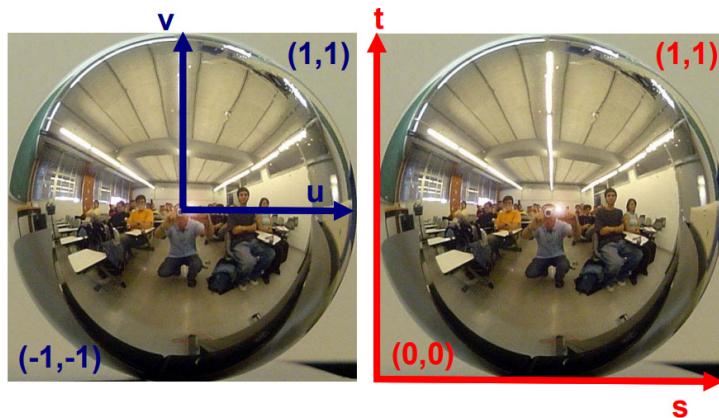


## Sphere mapping



Propietats:

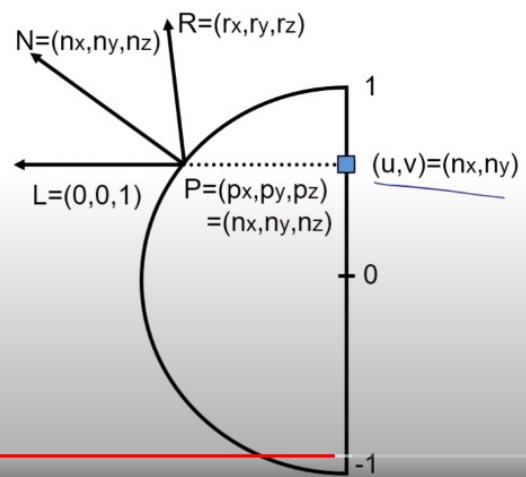
- De la textura només s'aprofita el cercle inscrit.
- Conté informació d'aproximadament tot l'entorn (totes direccions).
- Distorsió considerable a prop de la vora del cercle.



Relació entre els vectors  $R$  i  $(u, v)$

$$R = (2n_z n_x, 2n_z n_y, 2n_z^2 - 1)$$

$$\begin{aligned} n_x &= \frac{r_x}{r_z} = \sqrt{\frac{(r_z)^2 + 1}{2}} \\ n_y &= \frac{r_y}{r_z} = \frac{(r_x)}{2n_z} \rightarrow u \\ n_z &= \frac{r_z}{r_z} = \frac{(r_y)}{2n_z} \rightarrow v \end{aligned}$$



$R \rightarrow 1 \rightarrow P, N \rightarrow 2 \rightarrow (u, v) \rightarrow 3 \rightarrow (s, t)$

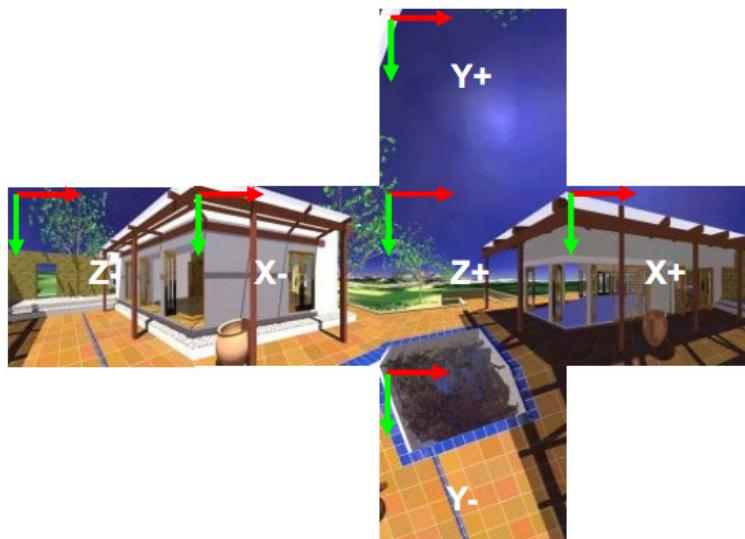
Càcul del color donat R

```
vec4 sampleSphereMap(sampler2D sampler, vec3 R)
{
    float z = sqrt((R.z+1.0)/2.0);
    vec2 st=vec2((R.x/(2.0*z)+1.0)/2.0,(R.y/(2.0*z)+1.0)/2.0);
    return texture2D(sampler, st);
}
```

### Eye / world coordinates

- Càcul amb vèrtex, normal en eye coords
  - L'entorn serà estàtic respecte la càmera.
  - L'objecte sempre reflecteix "la mateixa part" de l'entorn.
- Càcul amb vèrtex, normal en world coords
  - L'entorn serà dinàmic respecte la càmera.
  - L'objecte reflecteix diferents parts de l'entorn.

### Cube mapping



Així estan orientats els eixos (s,t) respecte cada imatge

Per tant aquestes són les sis textures que cal definir (origen de les coords (s,t) al pixel inferior esquerra):

1. Creació de les sis textures.
- ```
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X_EXT, ...);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X_EXT, ...);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y_EXT, ...);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y_EXT, ...);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z_EXT, ...);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z_EXT, ...);
```

2. Activació Cube mapping.

```
glEnable(GL_TEXTURE_CUBE_MAP_EXT);
```

3. Generació de coordenades de textura.

```
glTexGenfv(GL_S, GL_TEXTURE_GEN_MODE,GL_REFLECTION_MAP_EXT);
glTexGenfv(GL_T, GL_TEXTURE_GEN_MODE,GL_REFLECTION_MAP_EXT);
glTexGenfv(GL_R, GL_TEXTURE_GEN_MODE,GL_REFLECTION_MAP_EXT);
glEnable(GL_TEXTURE_GEN_S); glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);
```

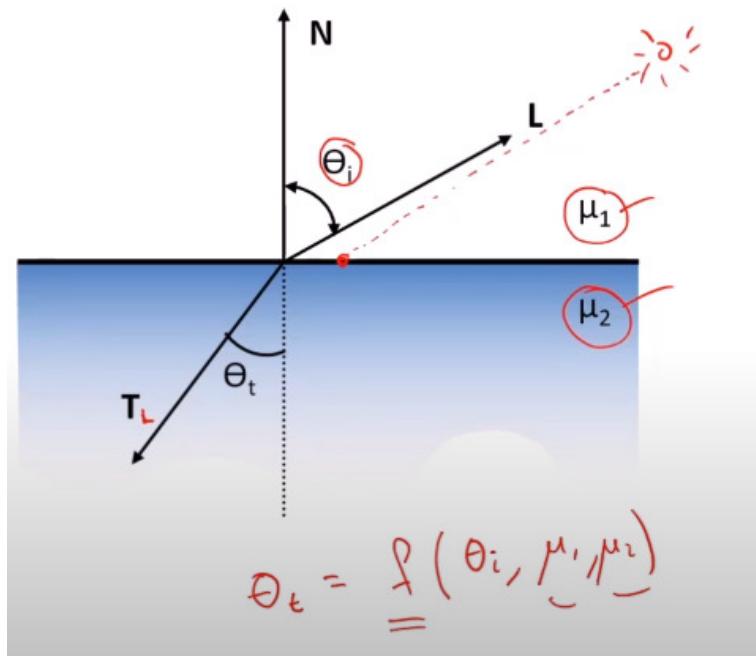
### Implementació en GLSL

```
uniform sampleCube samplerC;
...
vec3 R; (R = Rv = reflect(V,N))
...
vec4 color = textureCube(samplerC, R);
```

## Simulació d'objectes translúcids

### Llei de Snell

N, L, TL coplanars



$$\theta_t = \frac{\mu_1}{\mu_2} (\theta_i, \mu_1 / \mu_2)$$

$$\sin \theta_t = \frac{\mu_1}{\mu_2} \sin \theta_i = \mu \sin \theta i$$

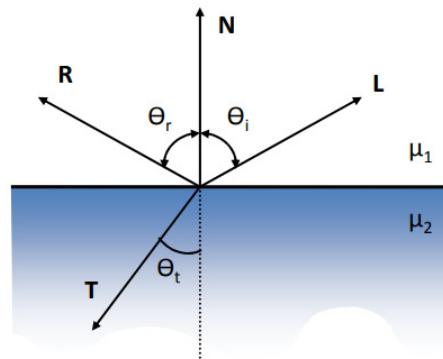
- + dens  $\rightarrow \mu_2 > \mu_1$   
TL "s'apropa" a N
- - dens  $\rightarrow \mu_2 < \mu_1$   
TL "s'allunya" a N

## Equacions de Fresnel

$$R = \frac{R_s + R_p}{2}$$

$$R_s = \left( \frac{\sin(\theta_t - \theta_i)}{\sin(\theta_t + \theta_i)} \right)^2$$

$$R_p = \left( \frac{\tan(\theta_t - \theta_i)}{\tan(\theta_t + \theta_i)} \right)^2$$



$T = 1-R$  -----> Probabilitat de transmissió

$R = (R_s+R_p)/2$  -----> Probabilitat de reflexió

## Aproximació de Schlick

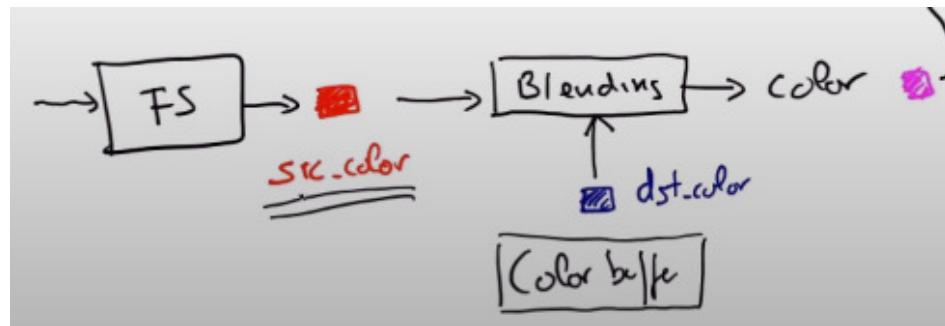
Evita l'ús de funcions trigonomètriques:

$$R = f + (1-f)(1 - L \cdot N)$$

$$f = \frac{(1-\mu)^2}{(1+\mu)^2}$$

## Alpha blending

RGBA-> Opacitat



```

glEnable(GL_BLEND)
color = s_factor*src_color + d_factor*dst_color
src_color <- (rs, gs, bs, as)
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
s_factor = as           d_factor = 1-as

```

Posibles valors:

- GL\_SRC\_ALPHA (el valor alpha del src\_color)
- GL\_ONE\_MINUS\_SRC\_ALPHA (1 menys el valor alpha del src\_color)
- GL\_ONE (1)
- ...

Ordre -> Back to front

## Ordre de pintat dels polígons

Opcions:

1. Dibuixar tot ordenat back\_to\_front  
depth\_test -> indef  
Rendiment molt dolent.
2. Ho fem de forma separada:
  - a. Dibuixar opacs (sense ordenar)
    - i. depth test OK!
    - ii. depth write OK!
  - b. Dibuixar transparents (ordenats)
    - i. depth test OK!
    - ii. depth write indiferent
3. Ho fem de forma separada:
  - a. Dibuixar opacs (sense ordenar)
    - i. depth test OK!
    - ii. depth write OK!
  - b. Dibuixar transparents (sense ordenar)
    - i. depth test OK!
    - ii. depth write NO!

Versió millorada del primer.

Versió més eficient. A vegades pot donar un color erroni.

## II·luminació local/global

Local -> llum directa. Només accepta camins  $L(D|S)E$

Global -> llum directa + indirecta. Idealment, accepta camins  $L(D|S)^*E$

### Radiometria

És la disciplina que estudia la mesura de les radiacions electromagnètiques.

| Sím.   | Radiomet.                    | Fotometria                   | Definició                                               | Ús                                                       |
|--------|------------------------------|------------------------------|---------------------------------------------------------|----------------------------------------------------------|
| $\Phi$ | Flux (W)                     | Flux (lm)                    | Energia que travessa una superficie per unitat de temps | Energia total que emet una font de llum                  |
| $E$    | Irradiancia ( $W/m^2$ )      | Iluminància ( $lux=lm/m^2$ ) | Flux per unitat d'àrea                                  | Llum que incideix en un punt, des de qualsevol direcció  |
| $I$    | Intensitat ( $W/sr$ )        | Intensitat ( $cd=lm/sr$ )    | Flux per unitat d'angle sòlid                           | Distribució direccional d'una llum puntual               |
| $L$    | Radiància $W/(sr \cdot m^2)$ | Luminància ( $cd/m^2$ )      | Flux per unitat d'àrea i unitat d'angle sòlid           | Energia que travessa un punt en una determinada direcció |

### Flux radiant $\phi$

Energia que travessa una superficie per unitat de temps.

Energia total que emet una font de llum.

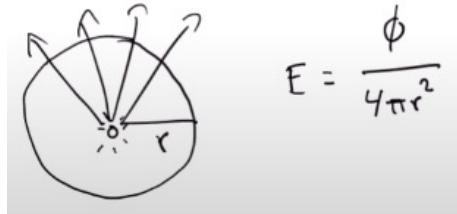
Les seves unitat són W (Watts) en radiometria o lm (lumens) en fotometria.

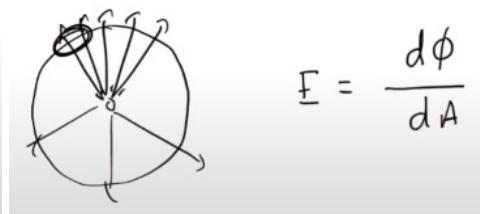
### Irradiancia E

Densitat de flux = flux per àrea.

Llum que incideix en un punt, des de qualsevol direcció.

Les seves unitat són  $W/m^2$  en radiometria o  $lm/m^2$  en fotometria.

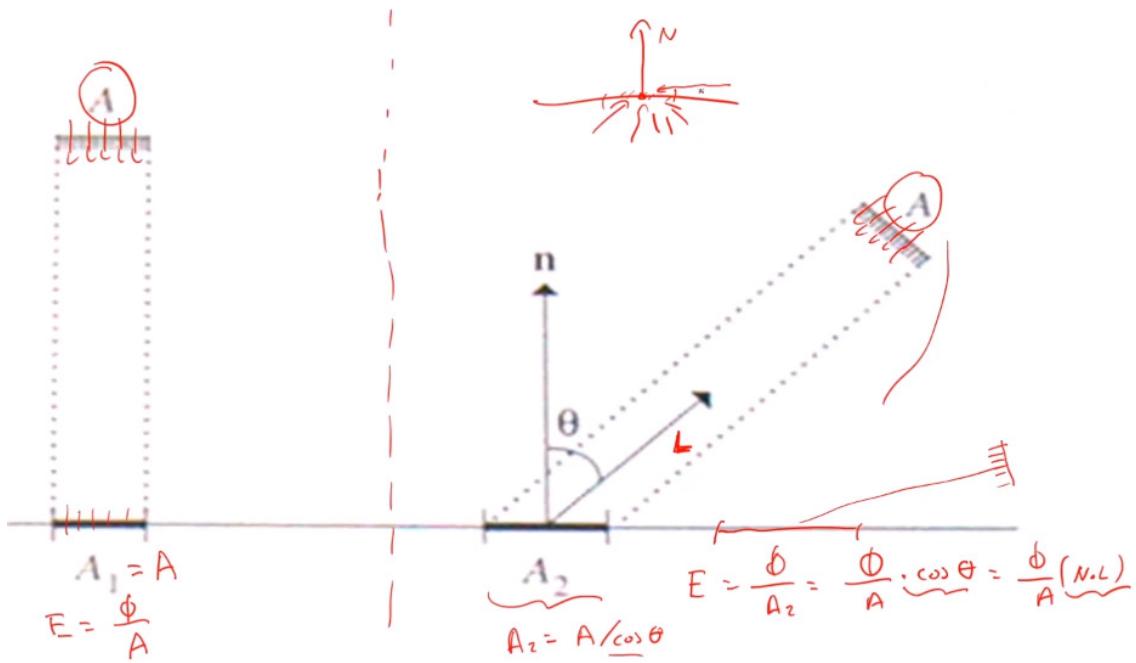


$$E = \frac{\phi}{4\pi r^2}$$


$$E = \frac{d\phi}{dA}$$

Decreix amb el quadrat de la distància.

**Llei de Lambert** -> és una relació empírica que relaciona l'absorció de llum amb les propietats del material travessat.



Com més tangencial, menor serà la contribució, ja que major serà el àrea.

### Intensitat I

Flux per unitat d'angle sòlid.

Distribució direccional d'una llum puntual.

Les seves unitat són W/sr en radiometria o cd=lm/sr (candela) en fotometria.

sr = esteroradiant.

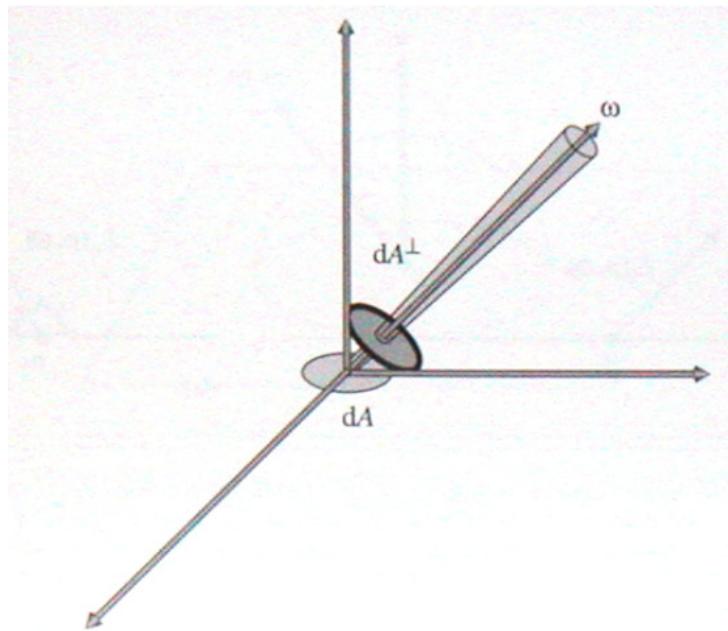
### Radiància L

Flux per unitat d'àrea i unitat d'angle sòlid.

Energia que travessa un punt en una determinada direcció.

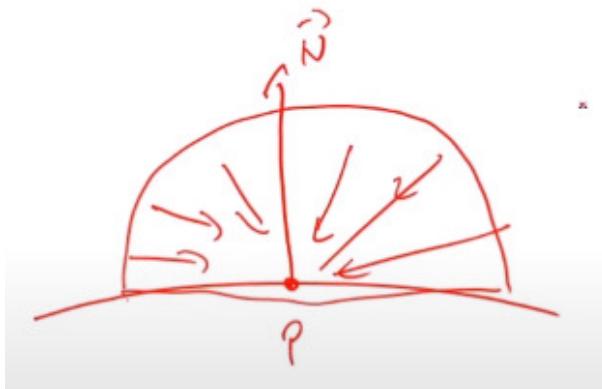
Les seves unitat són W/(sr\*m^2) en radiometria o cd/m^2 (candela) en fotometria.

$$L = \frac{d^2\Phi}{d\omega dA_{\perp}}$$



Tenim tres tipus de radiància:

- emitida  $L_e(p, w)$
- incidente  $L_i(p, w)$
- saliente  $L_o(p, w)$



L'energia d'aquesta semiesfera es pot calcular utilitzant aquesta fórmula:

$$E(p) = \int_{\Omega} L_i(p, \omega) \cos \theta \underline{d\omega}$$

Què és omega ( $\Omega$ )? La semiesfera centrada al punt x i alineada amb la normal n que representa totes les possibles direccions en les que pot arribar llum a x. (aqui, punt p)

## BRDF / BTDF / BSDF

$$f(p, w_{output}, w_{input})$$

Donat un punt, un vector d'entrada i un de sortida, quina és la quantitat de llum que es veu.

Propietats:

$$f_r(p, w_o, w_i) \geq 0$$

$$f_r(p, w_i, w_o) = f_r(p, w_o, w_i)$$

$$\int_{\Omega} f(p, w_i, w_o) \cdot \cos(\theta) dw \leq 1 \quad (\text{Conservació de l'energia})$$

## Equació general del rendering

$$L_o(p, \omega_o) = \underbrace{L_e(p, \omega_o)}_{\text{rad. emitida}} + \int_{S^2} f(p, \omega_o, \omega_i) \cdot \underbrace{L_i(p, \omega_i) \cos \theta}_{\text{rad. saliente}} \cdot dw_i$$

$L_o(p, \omega_o)$  -> Radiancia saliente de p en direcció  $\omega_o$ .

$L_e(p, \omega_o)$  -> Radiancia emitida de p en direcció  $\omega_o$ . Serà sempre 0 a no ser que p pertany-hi a una font de llum.

$S^2$  -> Totes les possibles direccions en una esfera.

## Light Paths

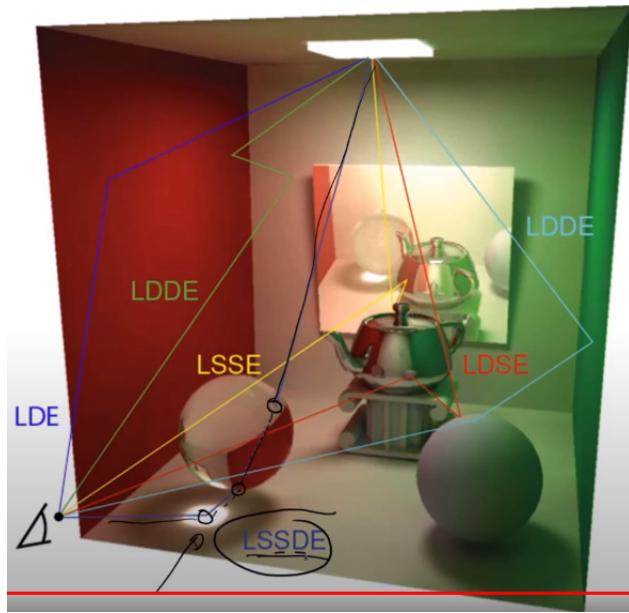
$$L(D | S)^* E$$

L -> llum

D -> difusa

S -> especular

E -> eye



## Ray tracing

Ray casting -> Per cada pixel enviem un raig i calculem les seves interseccions amb l'escena. És no recursiu, a diferència de Ray-tracing.

### Ray tracing clàssic

Es llancen raigs des de Eye.

Impactes en superfície: Hits

Cas recursiu?

- Material difús: No. Es calcula il·luminació local (light-ray/shadow-ray per cada font de llum).
- Material espectral: Sí. Poden ser translúcids (tenen 2 fills, el reflectit i el transmès) o miralls (només 1 fill). A cada hit, llencem els light-rays a totes les fonts de llum.

**Camins suportats:** LDS\*E, LS\*E

### Path tracing

Per cada pixel llancem r raigs.

Cal limitar profunditat. Per cada hit es calculen x rebots. Té problemes de soroll degut a la variància dels raigs rebotats.

**Camins suportats:** L (D | S)\* E

### Distributed ray tracing

Versió millorada de ray tracing classic. Per cada superficie espectral, es calculen varis raigs rebotats. En arribar a difusa, para el cas recursiu.

**Camins suportats:** LDS\*E, LS\*E

### Two-pass ray-tracing

#### Pas 1:

Raigs de llum des de la font fins la component difusa. Un cop arribes a la component difusa amb punt P, es guarda: hit(P,Energia).

#### Pas 2:

Fem camí normal de Eye fins a la component difusa. Quan arribem al punt P, consultem la info de hit(P,Energia) + la llum dels light-ray per cada font de llum

**Camins suportats:** LS\*DS\*E, LS\*E



## Funcions de GLSL

### Mix

mix — linearly interpolate between two values

**genType mix(genType x, genType y, genType a);**

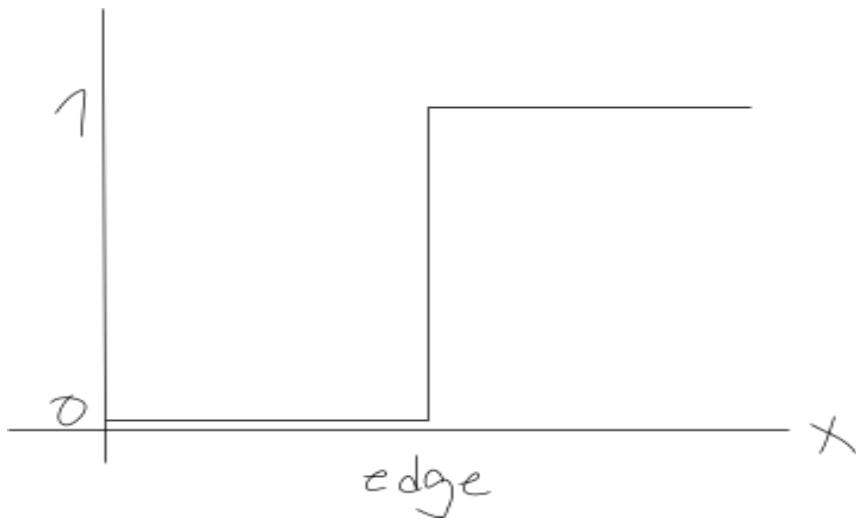
$$x \times (1-a) + y \times a$$

### Step

step generates a step function by comparing  $x$  to  $edge$ .

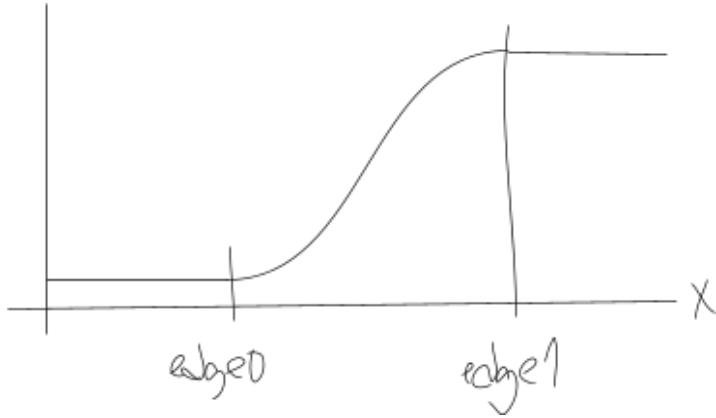
For element  $i$  of the return value, 0.0 is returned if  $x[i] < edge[i]$ , and 1.0 is returned otherwise.

**genType step(genType edge, genType x);**



### Smoothstep

**genType smoothstep(genType edge0, genType edge1, genType x);**



## Mod

mod returns the value of  $x$  modulo  $y$ . This is computed as  $x - y * \text{floor}(x/y)$ .

**genType mod(genType x, genType y);**

## Floor

floor — find the nearest integer less than or equal to the parameter

**genType floor(genType x);**

## Normalize

normalize — calculates the unit vector in the same direction as the original vector

serveix per calcular un vector de mida 1 (esfera de radi 1 centrat a l'origen)

**genType normalize(genType v);**

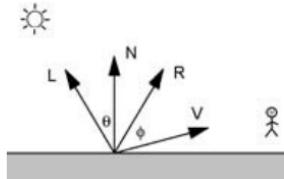
## Lambert

### Reflexión Difusa

#### Descripción

- Es característica de superficies rugosas, mates, sin brillo.
- Se modela fácilmente con la Ley de Lambert. Así, el brillo observado en un punto depende sólo del ángulo  $\theta$ ,  $0 \leq \theta \leq 90$ , entre la dirección a la fuente de luz  $L$  y la normal  $N$  en dicho punto.
- $L$  y  $N$  son vectores unitarios y  $k_d$ ,  $0 \leq k_d \leq 1$ , representa la parte de luz difusa reflejada por la superficie.

$$I_d = k_d L_d \cos \theta = k_d L_d (L \cdot N) \quad (2)$$



## Phong

$$C(N, V, L) = K_a I_a + K_d I_d (N \cdot L) + K_s I_s (R \cdot V)^s$$

```
vec4 light(vec3 N, vec3 V, vec3 L)
{
    N=normalize(N); V=normalize(V); L=normalize(L);
    vec3 R = normalize( 2.0*dot(N,L)*N-L );
    float NdotL = max( 0.0, dot( N,L ) );
    float RdotV = max( 0.0, dot( R,V ) );
    float Idiff = NdotL;
    float Ispec = 0;
    if (NdotL>0) Ispec=pow( RdotV, matShininess );
    return
        matAmbient * lightAmbient +
        matDiffuse * lightDiffuse * Idiff+
        matSpecular * lightSpecular * Ispec;
}
```