

Procés de visualització projectiu

C. Andujar

Sep 2020

SISTEMES DE COORDENADES I TRANSFORMACIONS GEOMÈTRIQUES

Paradigma projectiu simplificat



Informació geomètrica al pipeline

vertices.push_back(QVector3D(**0, 1, 0**));  Coordenades d'un vèrtex

...

normals.push_back(QVector3D(**0, 0, 1**));  Components de la normal

setUniformValue("lightPosition", QVector3D(**0, 0, 0**));

setUniformValue("lightDir", QVector3D(**0, 1, 0**));  Posició d'una llum

Direcció d'una llum

Informació geomètrica al pipeline

- Punts
- Vectors
 - Normals (perpendiculars a la superfície)
 - Resta de vectors

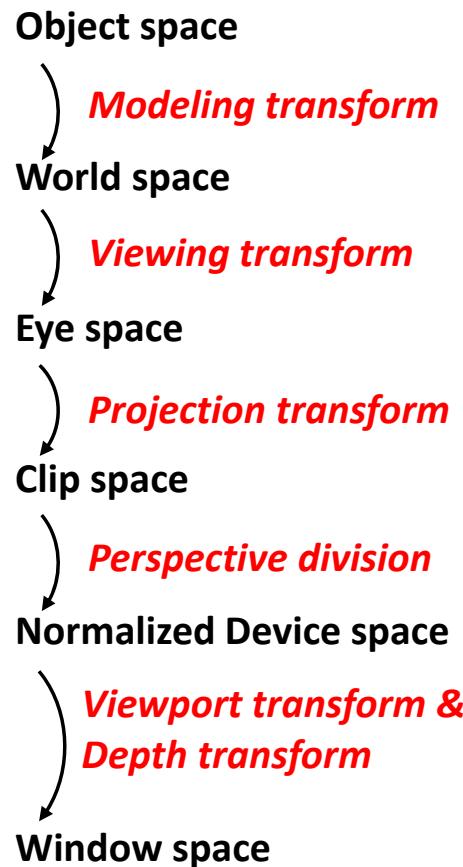
Coordenades homogènies

Punts

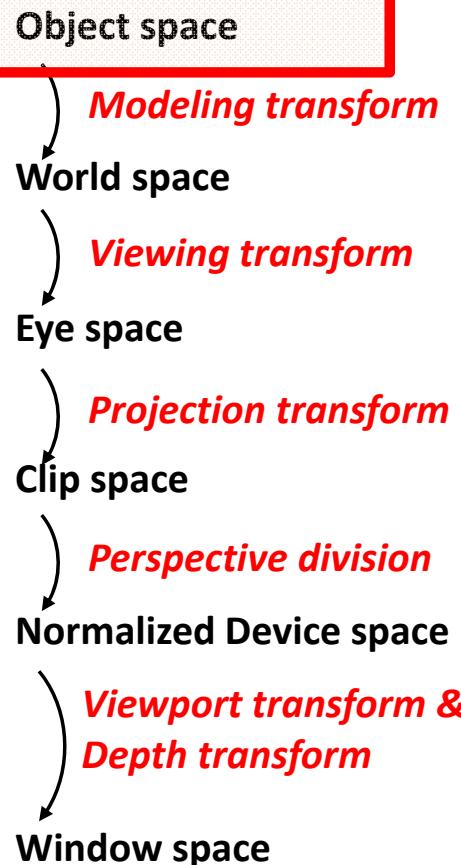
Vectors

TRANSFORMACIÓ DE PUNTS

Sistemes de coordenades



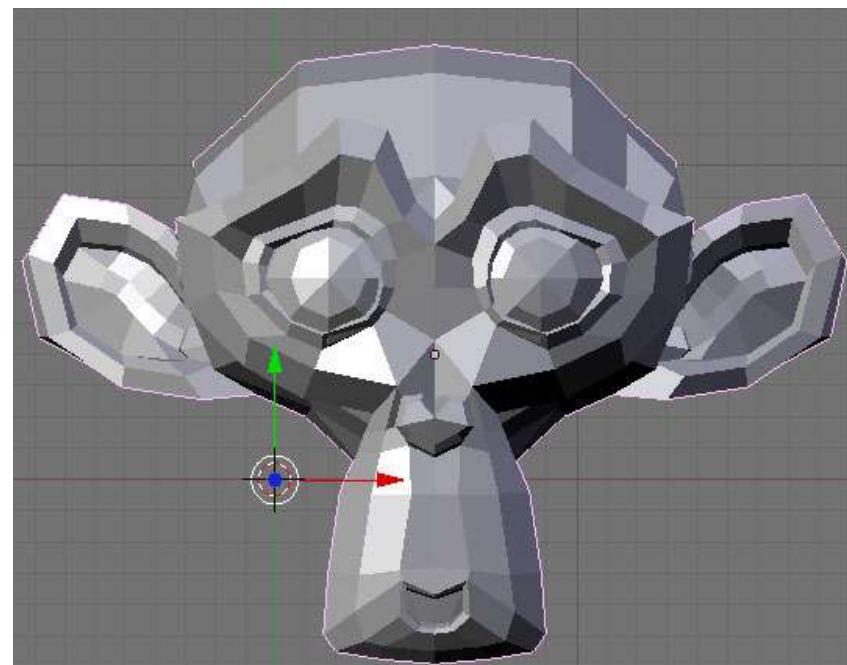
Sistemes de coordenades



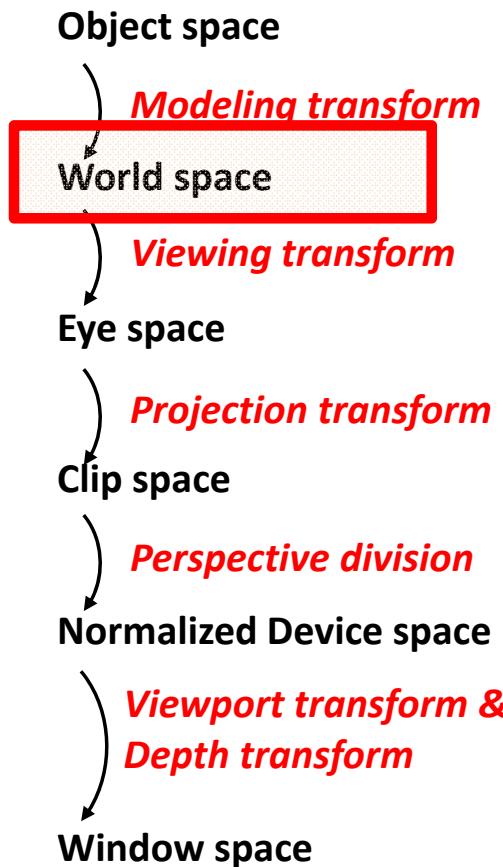
Object space (x_m, y_m, z_m, w_m)

Model space, SC del model, SC de l'objecte

- *SC utilitzat per modelar l'objecte.*
- w_m normalment serà 1.0 (punts)



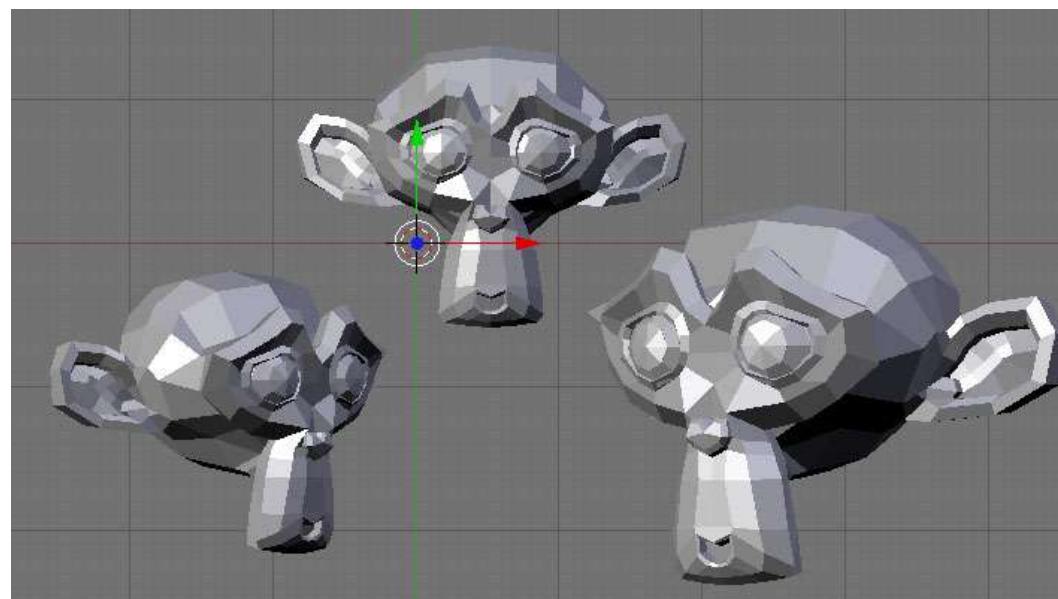
Sistemes de coordenades



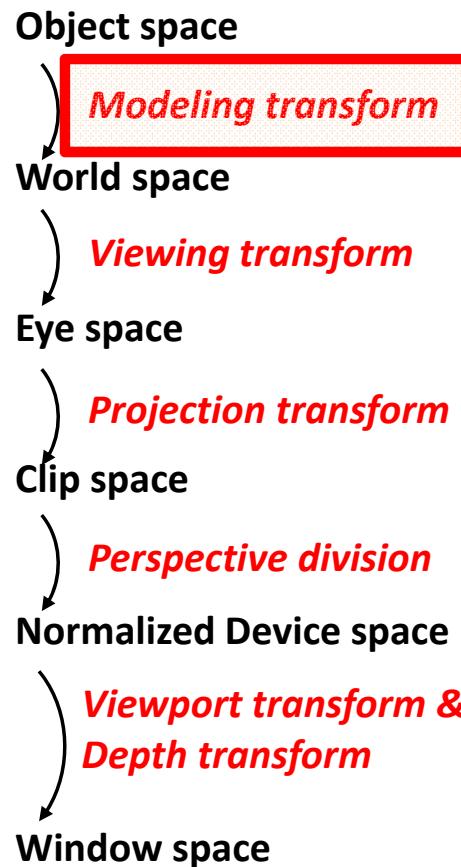
World space (x_a, y_a, z_a, w_a)

SC de mon, SC de l'aplicació

- SC utilitzat per representar l'escena
- La transformació de modelat sovint preserva la component homogènia i per tant, w_a normalment serà 1.0 (punts)



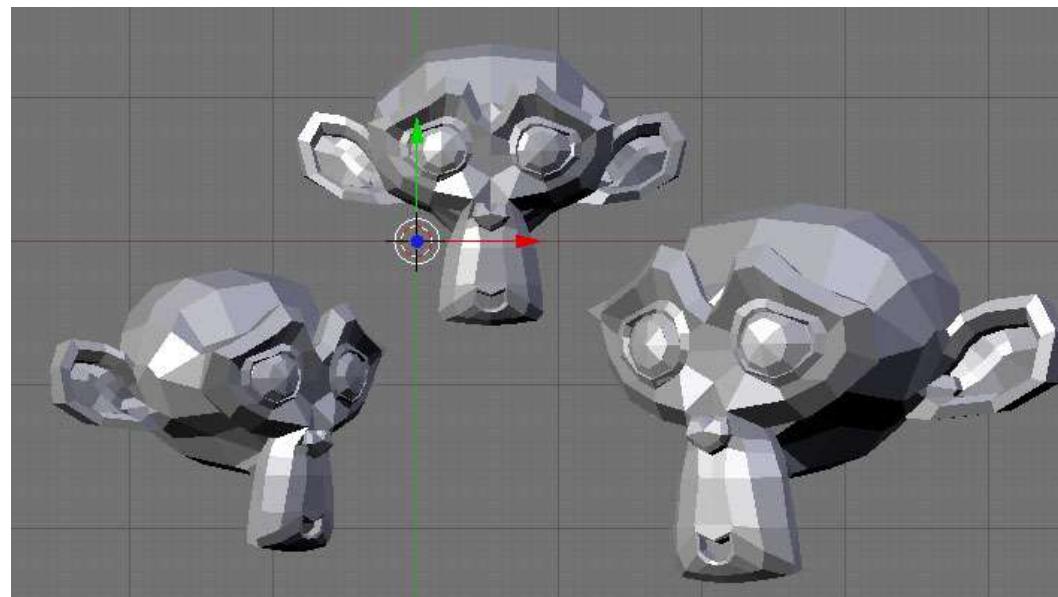
Sistemes de coordenades



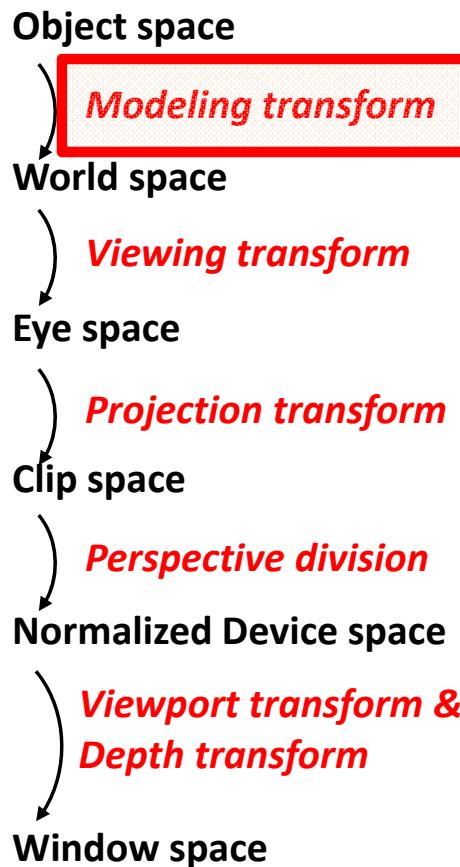
Modeling transform

Transformació de modelat

- Aquesta transformació situa cada instància d'un objecte en relació a l'escena.
- Sovint és la identitat, o una composició de translacions, rotacions i escalats



Sistemes de coordenades



Modeling transform
Transformació de modelat

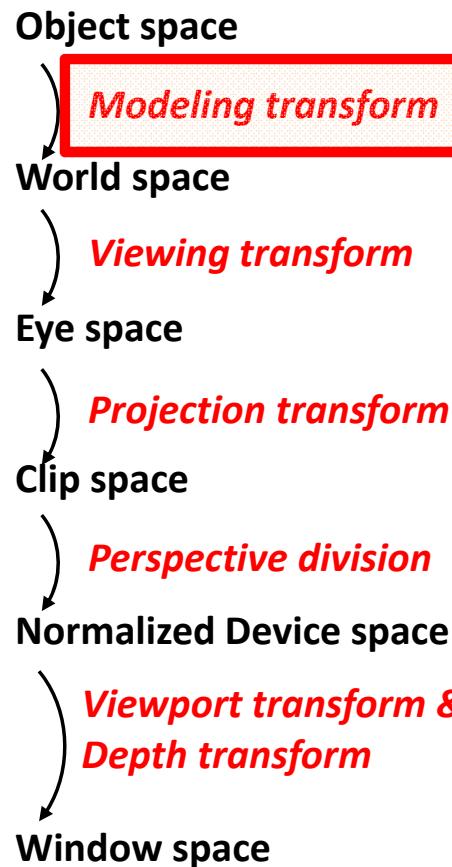
$$\text{Translate}(t_x, t_y, t_z) \quad T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

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

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

Sistemes de coordenades



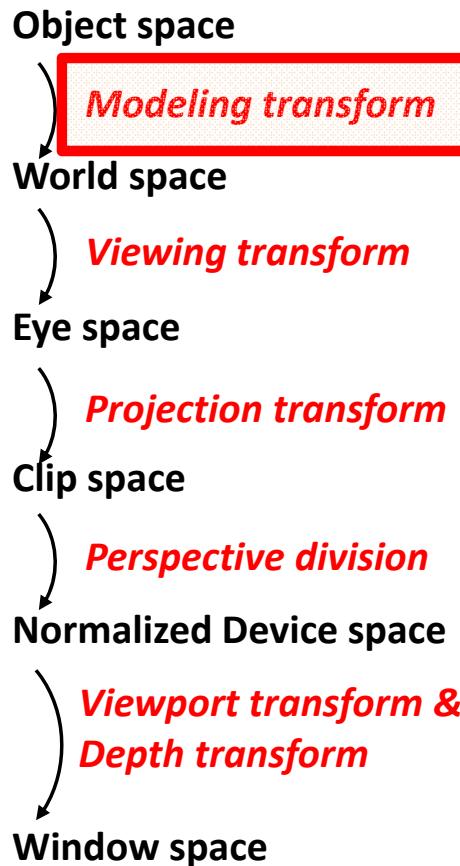
Modeling transform
Transformació de modelat

$$\text{glRotate}^*(\alpha, 1, 0, 0): \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate}^*(\alpha, 0, 1, 0): \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate}^*(\alpha, 0, 0, 1): \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

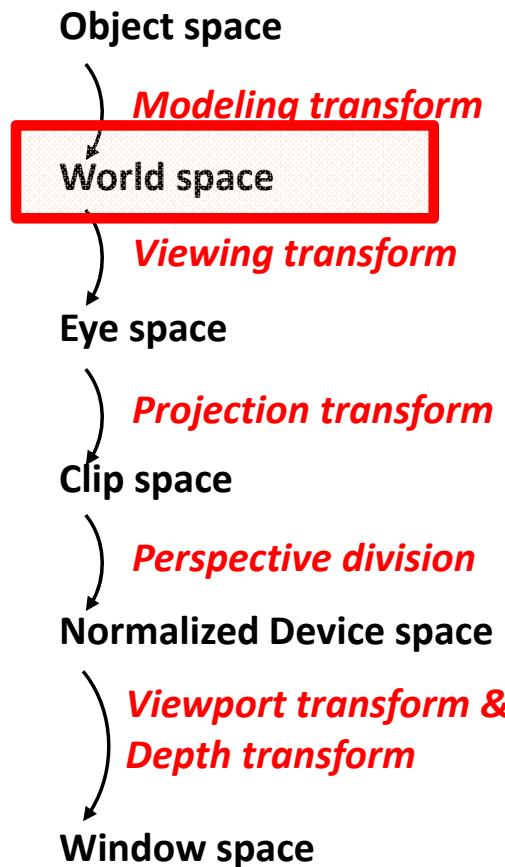
Sistemes de coordenades



- Punts: matriu 4×4 multiplicada pel punt (l'ordre és important!)

$$\begin{bmatrix} x_a \\ y_a \\ z_a \\ w_a \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix}$$

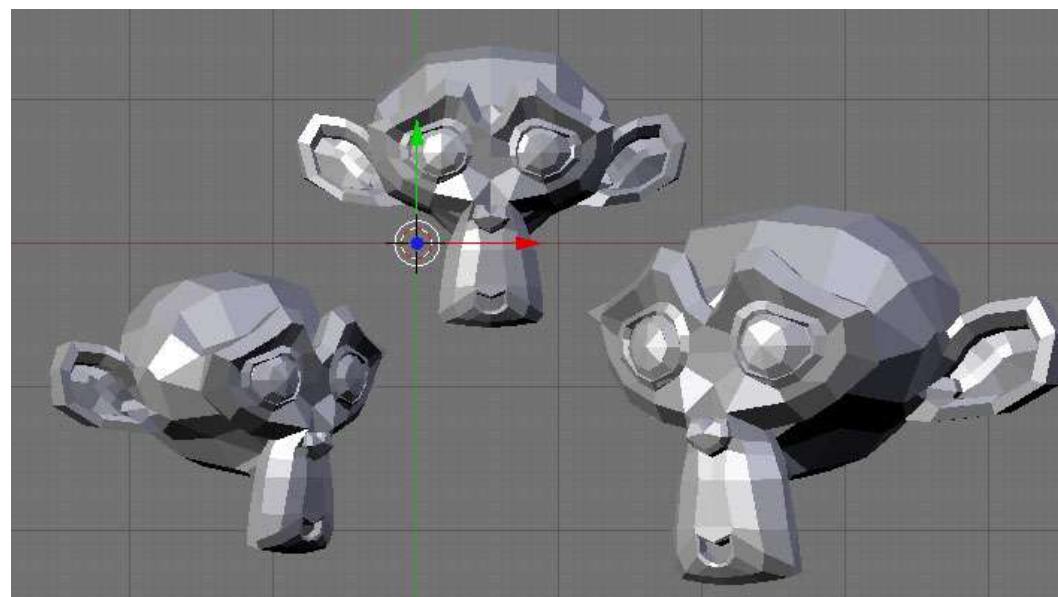
Sistemes de coordenades



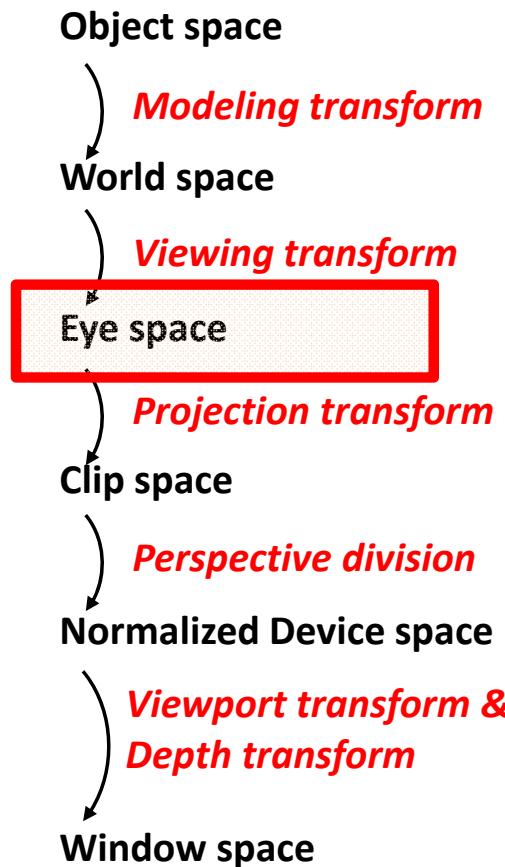
World space (x_a, y_a, z_a, w_a)

SC de mon, SC de l'aplicació

- *SC utilitzat per representar l'escena*
- *La transformació de modelat sovint preserva la component homogènia i per tant, w_a normalment serà 1.0 (punts)*



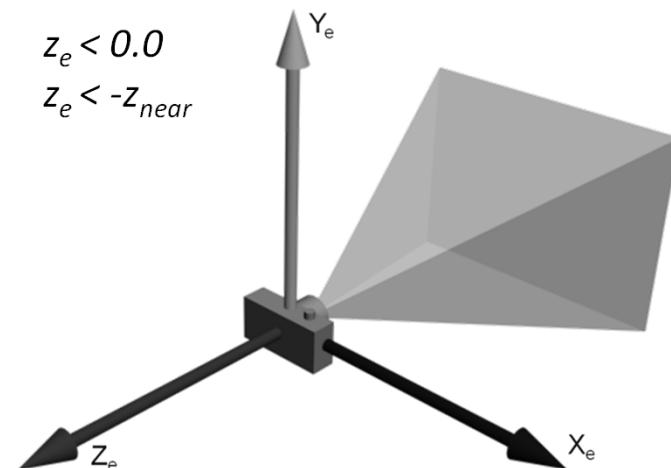
Sistemes de coordenades



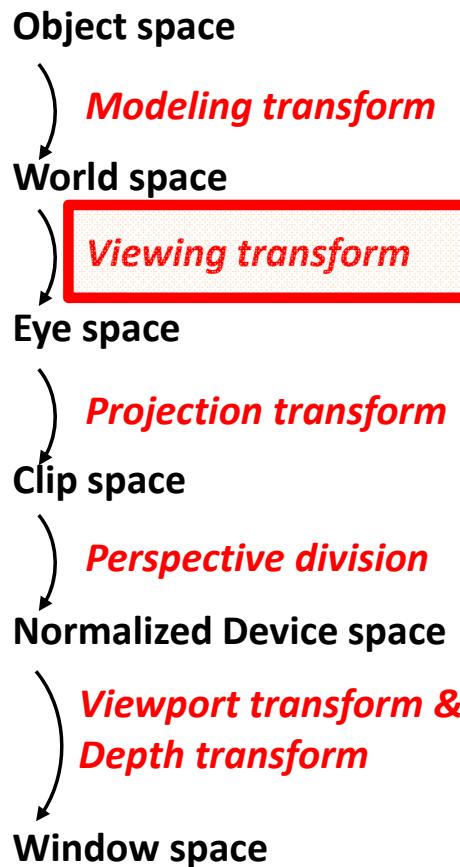
Eye space (x_e, y_e, z_e, w_e)

SC de l'observador, SC de la càmera

- *SC associat a la càmera*
- *La transformació de visualització sovint preserva la component homogènia i per tant, w_e normalment serà 1.0 (punts)*
- *Si la càmera és perspectiva, per tal que el punt sigui visible...*



Sistemes de coordenades

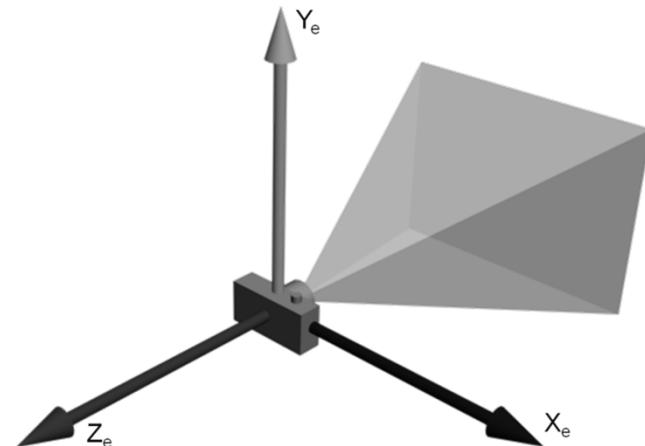


Viewing transform

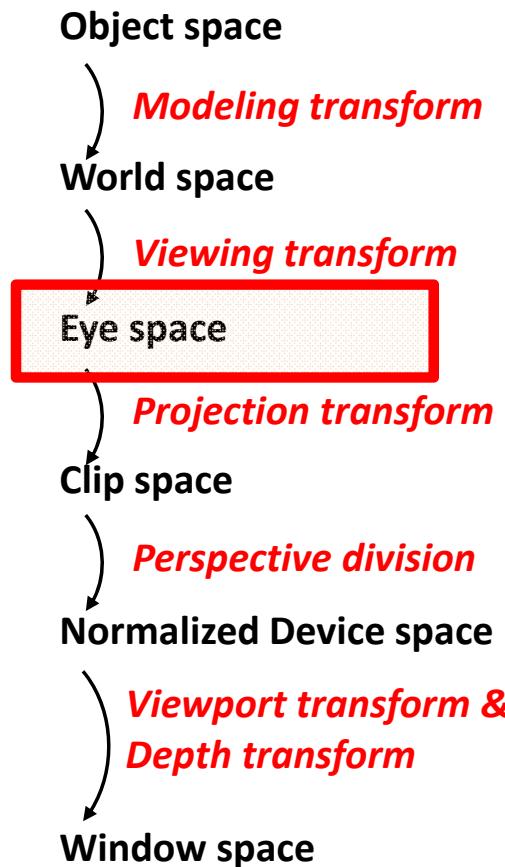
Transformació de visualització, transformació de càmera

- Fa un canvi al sistema de referència de la càmera.
- Depèn de la posició i orientació de la càmera
- Sovint es defineix amb crides tipus **lookAt**, o amb una composició de translacions i rotacions:

- *lookAt(eye, target, up)*
- $T(0,0,-d)*R_z(-\varphi)*R_x(\Theta)*R_y(-\Psi)*T(-VRP)$



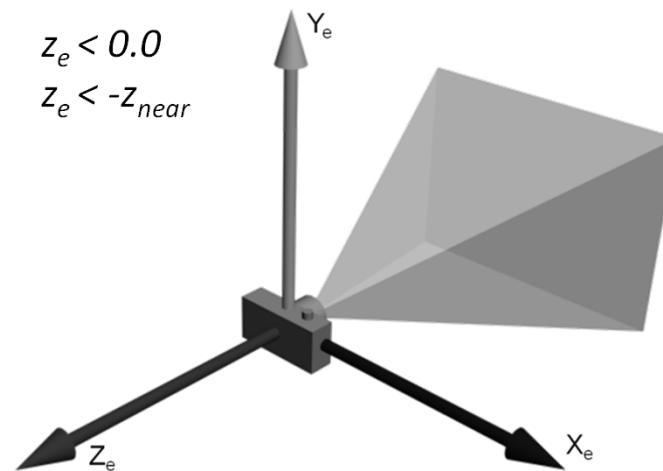
Sistemes de coordenades



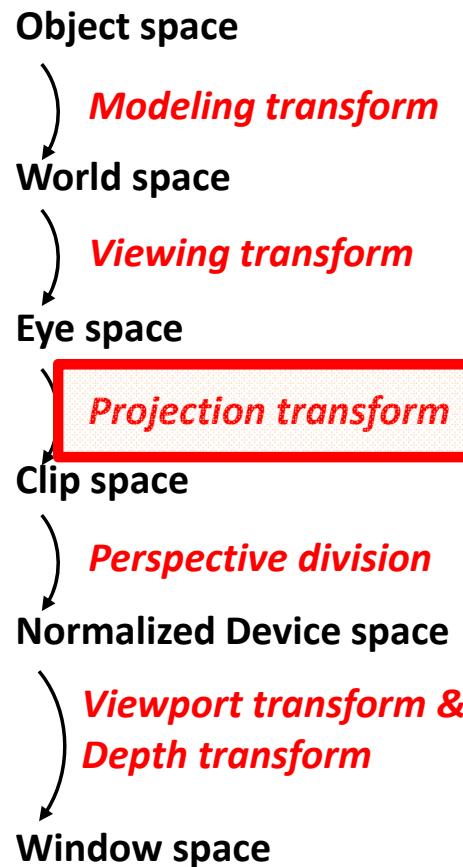
Eye space (x_e, y_e, z_e, w_e)

SC de l'observador, SC de la càmera

- *SC associat a la càmera*
- *La transformació de visualització sovint preserva la component homogènia i per tant, w_e normalment serà 1.0 (punts) o 0.0 (vectors)*
- *Si la càmera és perspectiva, per tal que el punt sigui visible...*



Sistemes de coordenades

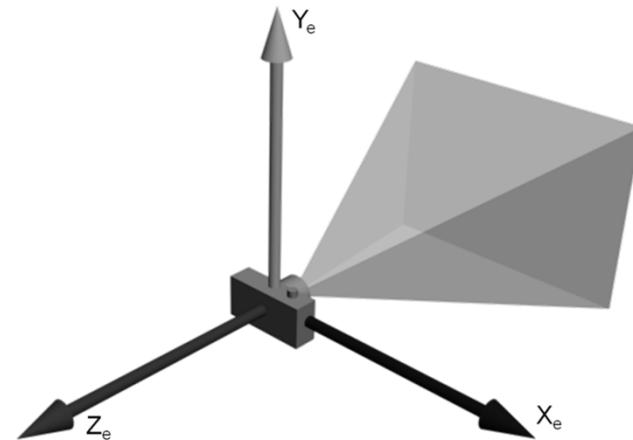


Projection transform

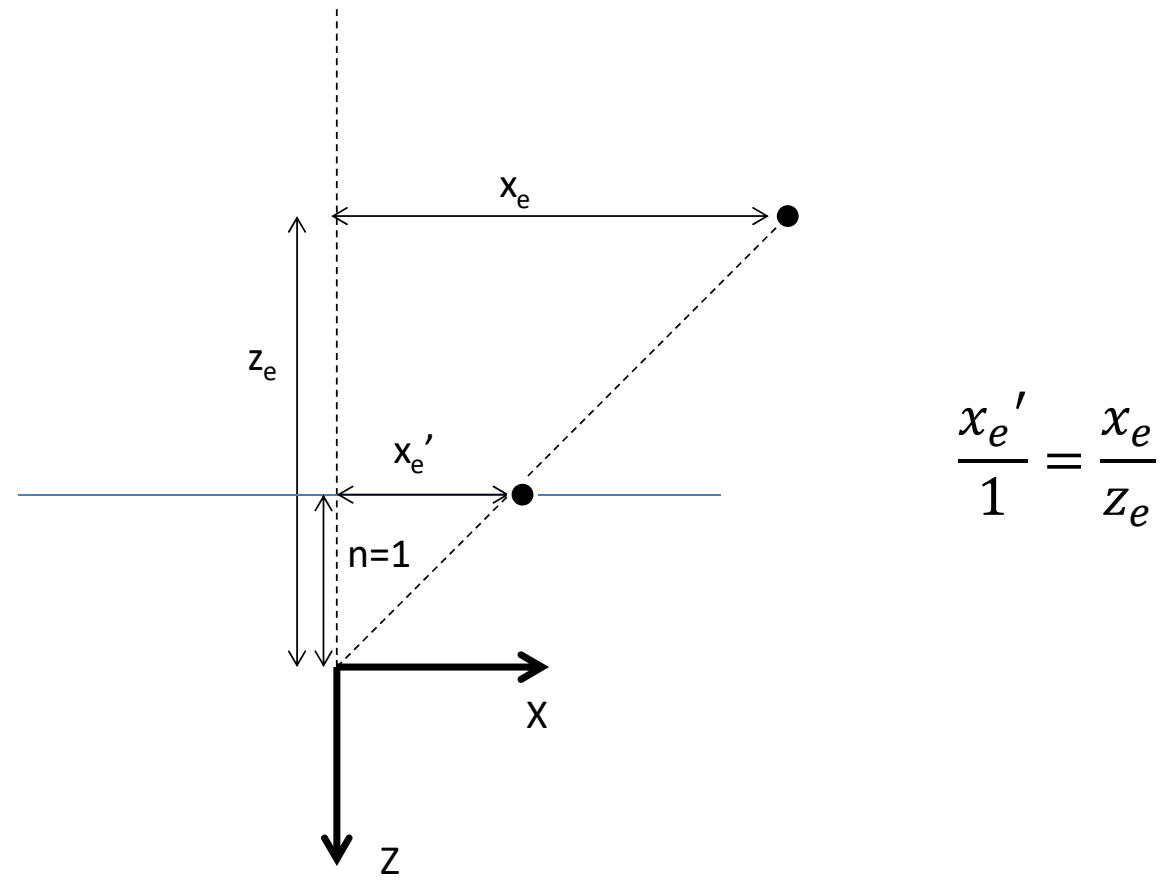
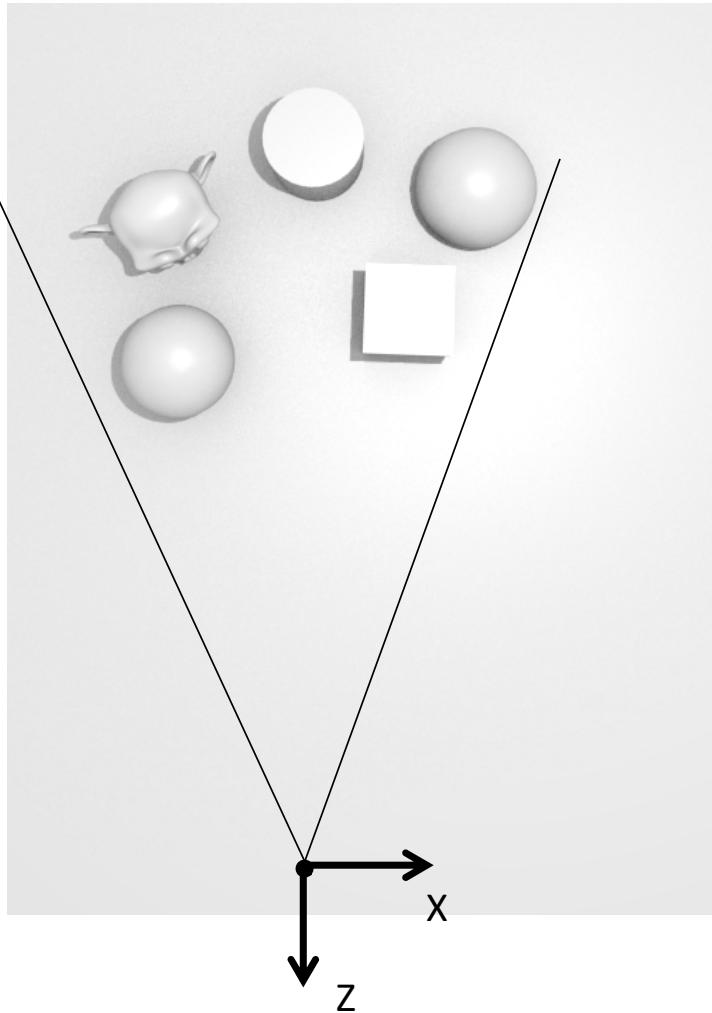
Transformació de projecció

- 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)`



Projecció i divisió per z_e



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

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

Pas eye space \rightarrow clip space

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

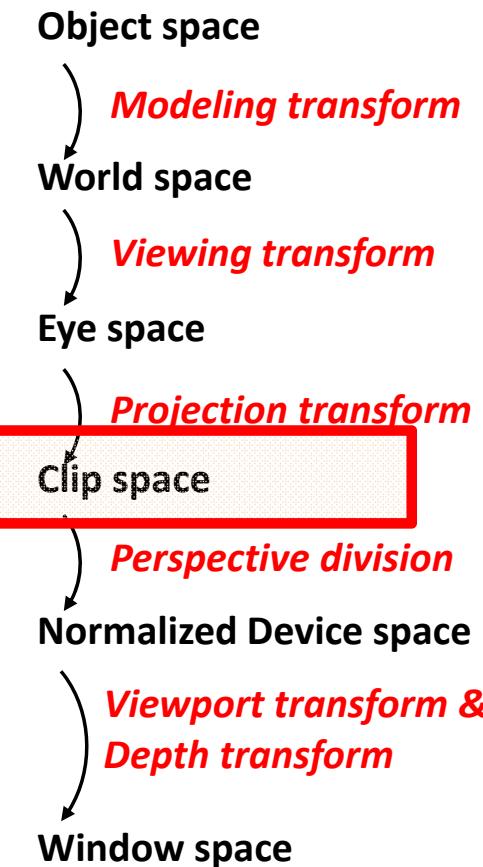
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1.2 & -2.2 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} x_e \\ y_e \\ -1 \\ 1 \end{bmatrix}$$

Punt sobre el pla znear

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

Punt sobre el pla zfar

Sistemes de coordenades



Clip space (x_c, y_c, z_c, w_c)

Coordenades de clipping, coordenades de retallat

- Si un punt és interior al frustum,

$$-w_c \leq x_c \leq w_c$$

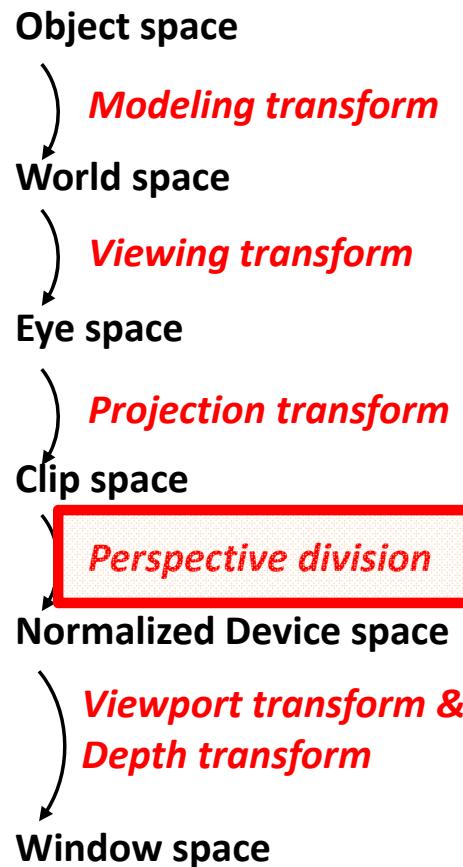
$$-w_c \leq y_c \leq w_c$$

$$-w_c \leq z_c \leq w_c$$

- Si la càmera és perspectiva, llavors

$$w_c = -z_e$$

Sistemes de coordenades



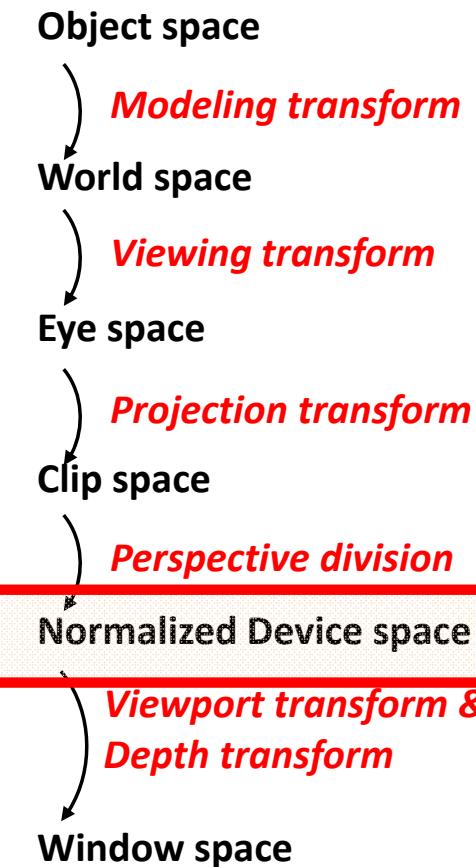
Perspective division

Divisió de perspectiva

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

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

Sistemes de coordenades



Normalized device space (x_n, y_n, z_n)

NDC, cordenades normalitzades

- Si un punt és interior al frustum,

$$-1 \leq x_n \leq 1$$

$$-1 \leq y_n \leq 1$$

$$-1 \leq z_n \leq 1$$

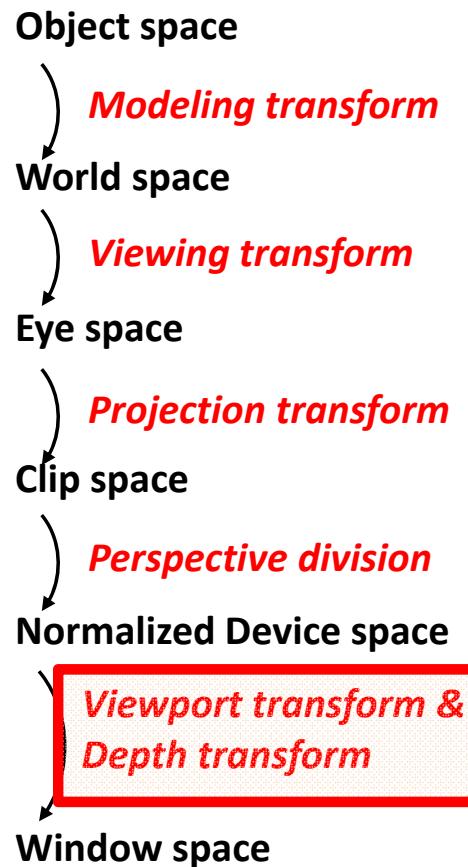
- Els punts situats sobre z_{near} tenen

$$z_n = -1$$

- Els punts situats sobre z_{far} tenen

$$z_f = +1$$

Sistemes de coordenades



Viewport transformation

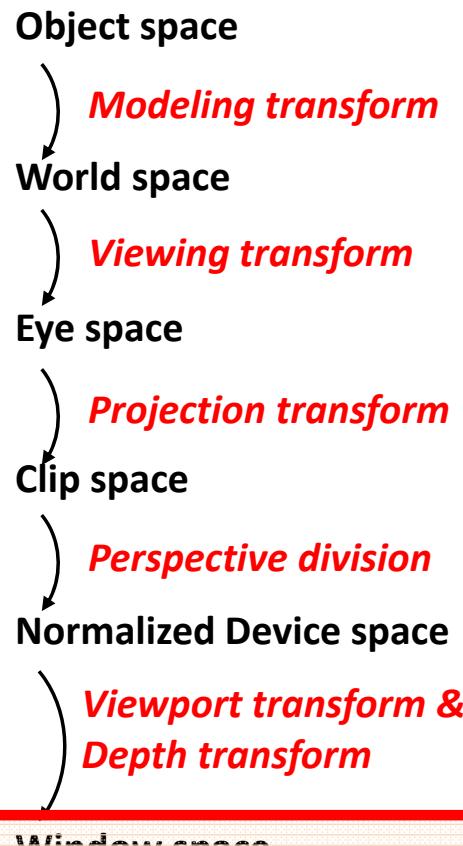
Transformació mon-dispositiu

- Depèn del viewport definit amb `glViewport`

Depth range transformation

- Depèn de l'interval definit amb `glDepthRange` (per defecte $[0,1]$)

Sistemes de coordenades



Window space (x_d, y_d, z_d)

Cordenades de finestra, coordenades de dispositiu

- Si un punt és interior al frustum,

$$0 \leq x_d \leq w$$

$$0 \leq y_d \leq h$$

$$0 \leq z_d \leq 1$$

- Els punts situats sobre znear tenen

$$z_d = 0$$

- Els punts situats sobre zfar tenen

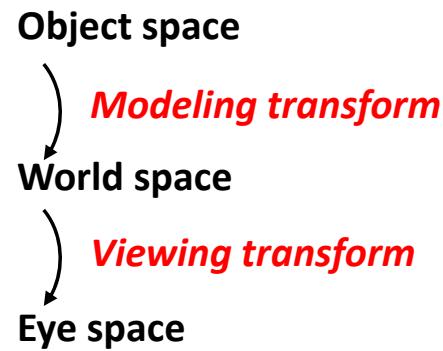
$$z_d = 1$$

- Quan més endavant es generin fragments,

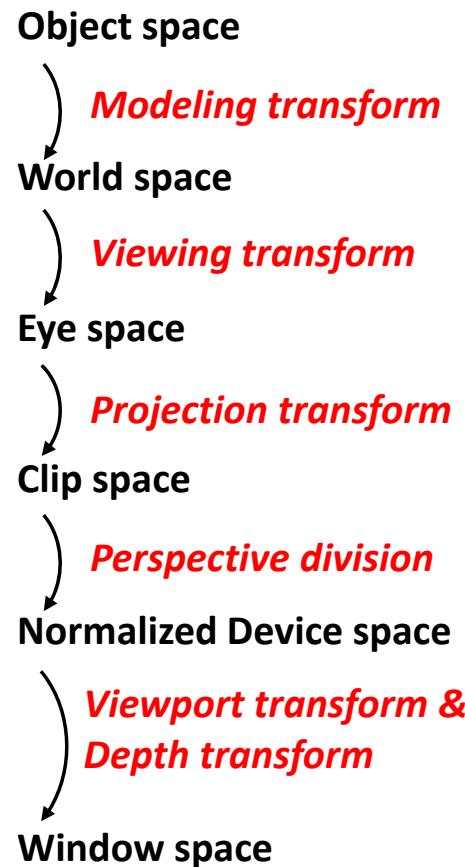
$$\text{gl_FragCoord.w} = 1/w_c = -1/z_e$$

TRANFORMACIÓ DE VECTORS (EXCEPTE NORMALS)

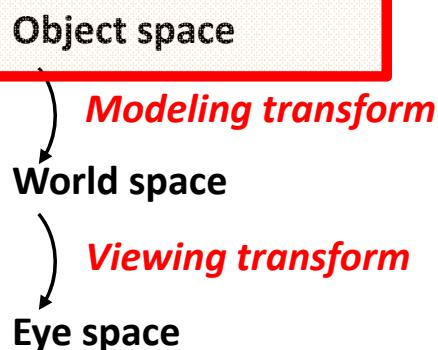
Sistemes de coordenades



Sistemes de coordenades

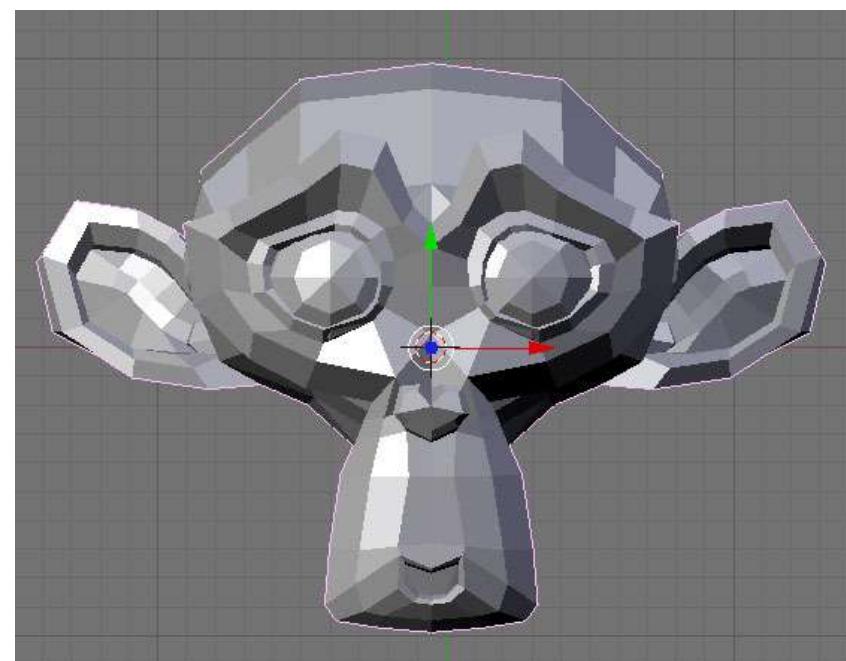


Sistemes de coordenades

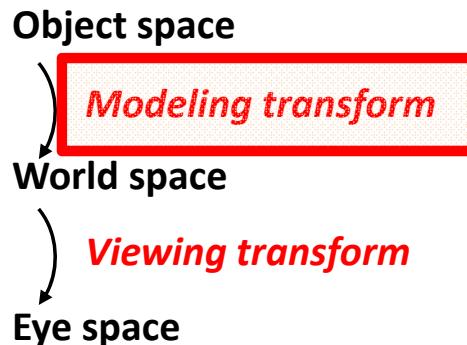


Object space (x_m, y_m, z_m, w_m)
Model space, SC del model, SC de l'objecte

- SC utilitzat per modelar l'objecte.
- w_m normalment serà 1.0 (punts) o 0.0 (vectors)



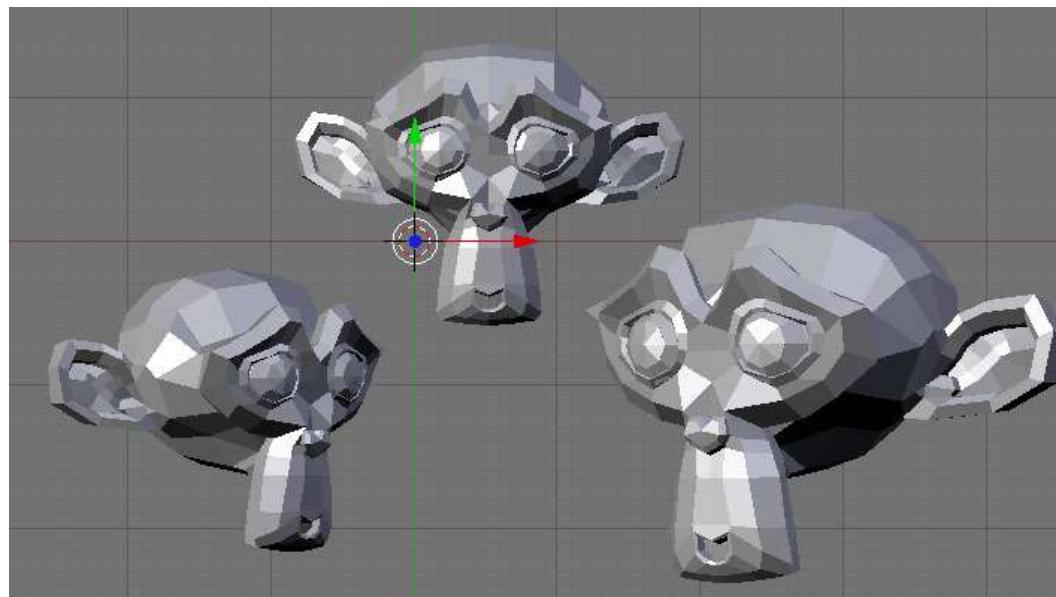
Sistemes de coordenades



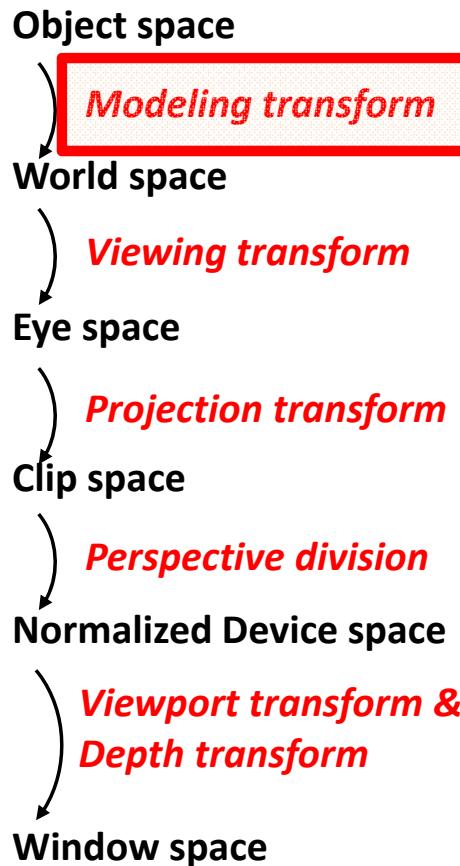
Modeling transform

Transformació de modelat

- Aquesta transformació situa cada instància d'un objecte en relació a l'escena.
- Sovint és la identitat, o una composició de translacions, rotacions i escalats



Sistemes de coordenades



Modeling transform
Transformació de modelat

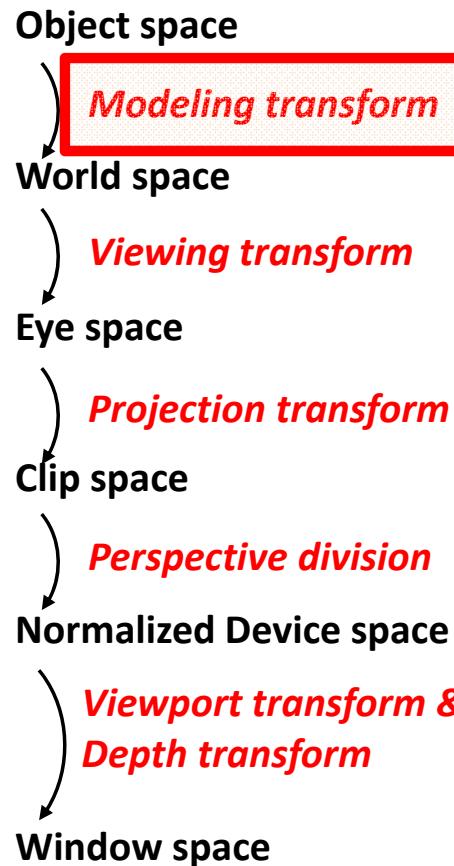
$$\text{translate}(t_x, t_y, t_z) \quad T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

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

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

Sistemes de coordenades



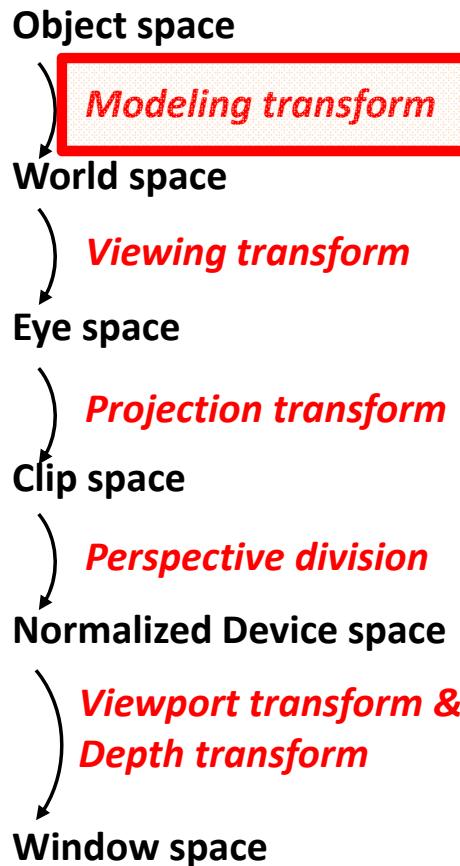
Modeling transform
Transformació de modelat

$$\text{glRotate}^*(\alpha, 1, 0, 0): \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate}^*(\alpha, 0, 1, 0): \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{glRotate}^*(\alpha, 0, 0, 1): \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Sistemes de coordenades



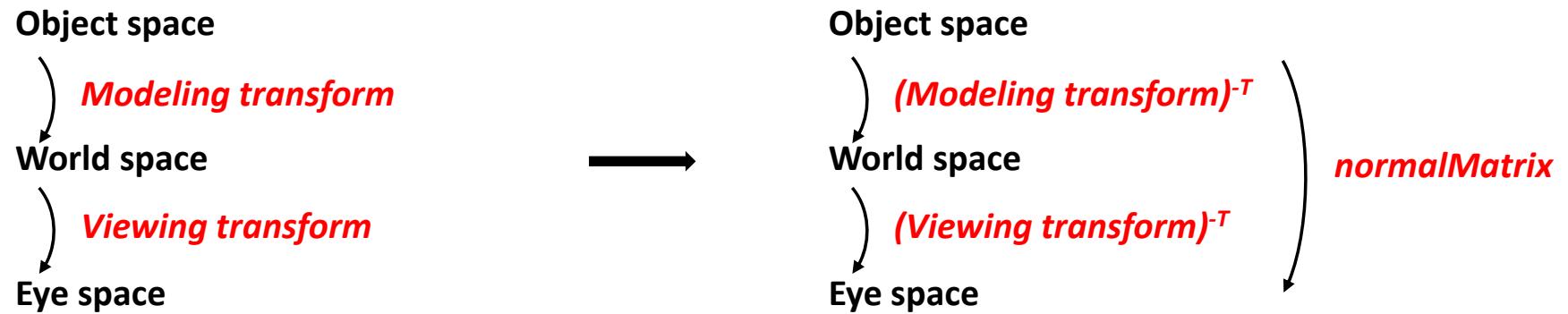
- Punts, vectors: matriu 4×4 multiplicada pel punt/vector

$$\begin{bmatrix} x_a \\ y_a \\ z_a \\ w_a \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix}$$

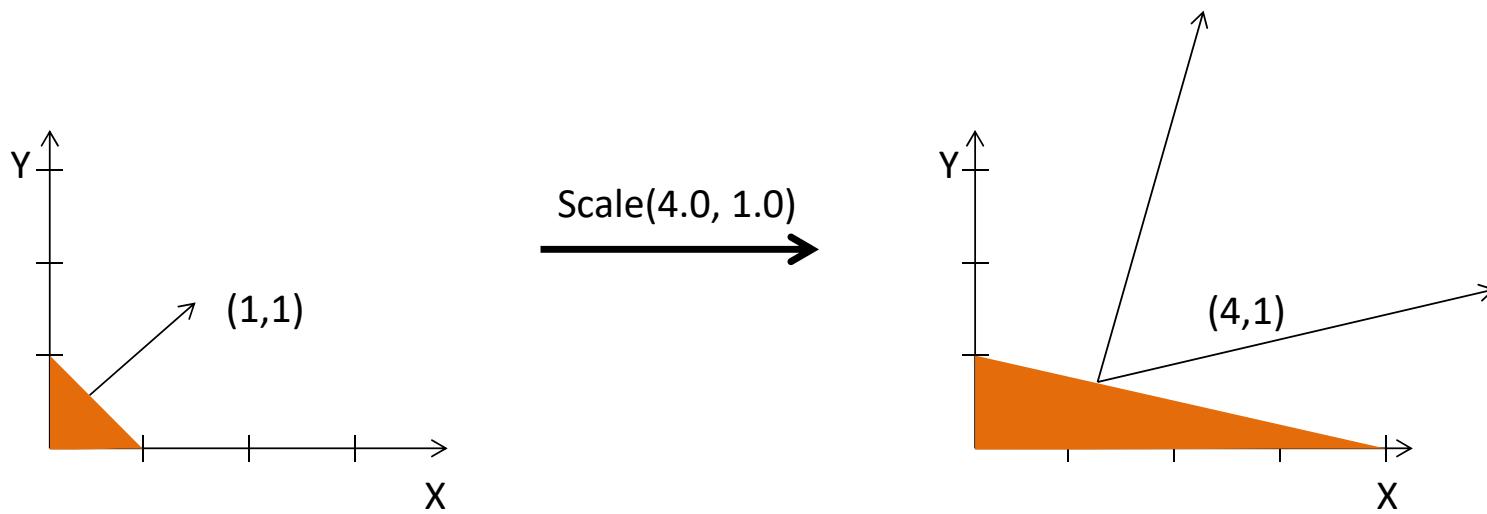
$$\begin{bmatrix} x_a \\ y_a \\ z_a \\ w_a \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ z_m \\ 0 \end{bmatrix}$$

TRANSFORMACIÓ NORMALS

Sistemes de coordenades



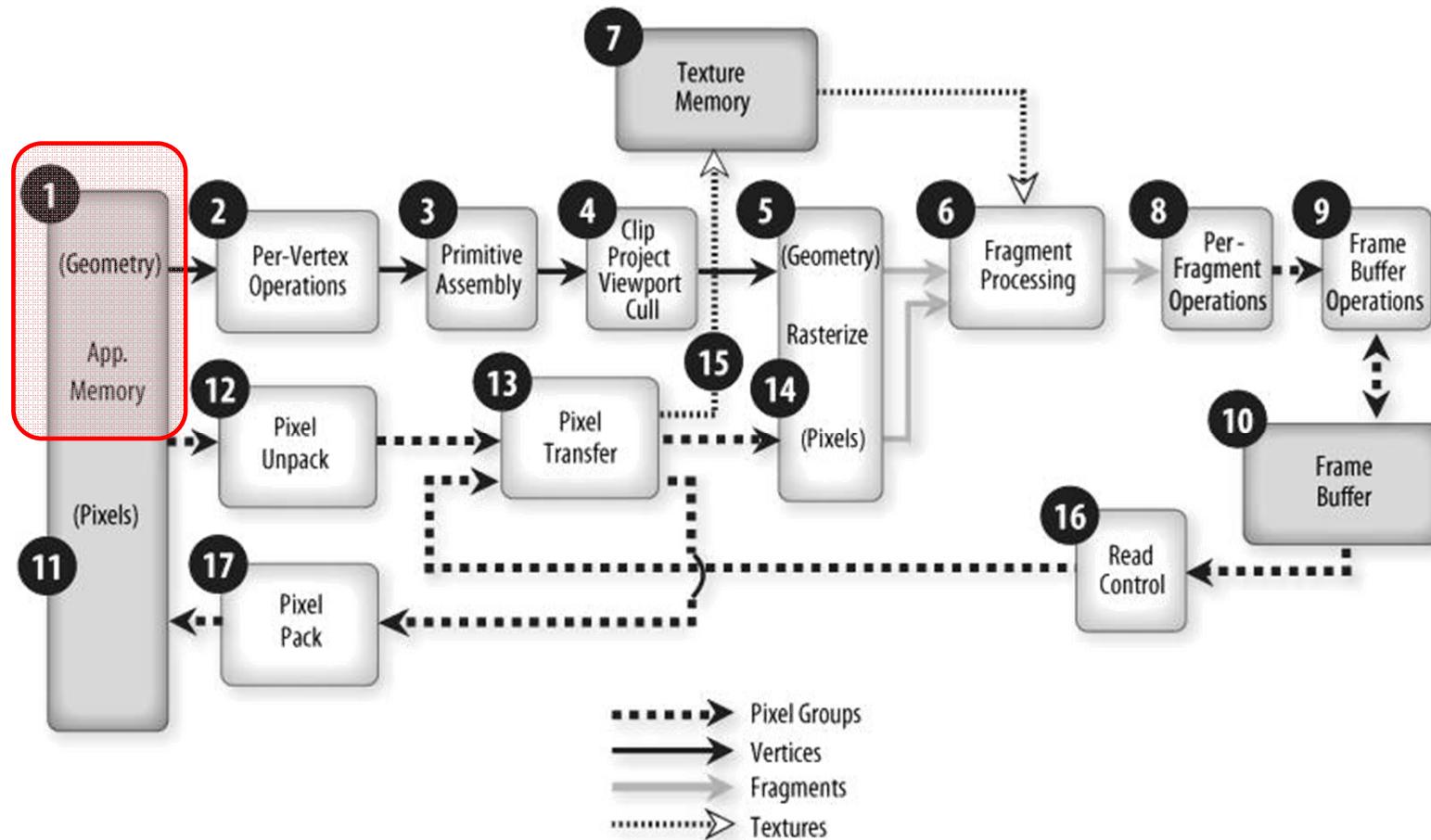
Sistemes de coordenades



Transformació de normals

PIPELINE OPENGL

Pipeline OpenGL



Pipeline OpenGL

1. Dibuix de primitives

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

- Vèrtex a vèrtex: *glBegin*, *glVertex*, *glEnd* (compatibility profile)
- Vertex array object: *glDrawArrays*, *glDrawElements*...

Exemple (*compatibility*)

```
glBegin(GL_POLYGON);
    glNormal3f(nx0, ny0, nz0);
    glVertex3f(x0, y0, z0);           Coords i normals en
    glNormal3f(nx1, ny1, nz1);       object space
    glVertex3f(x1, y1, z1);
...
glEnd();
```

Exemple (*compatibility/core*)

```
// createBuffers
glm::vec3 Vertices[...]; // Tres vèrtexs amb X, Y i Z
Vertices[0] = glm::vec3(-1.0, -1.0, 0.0);
...
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);

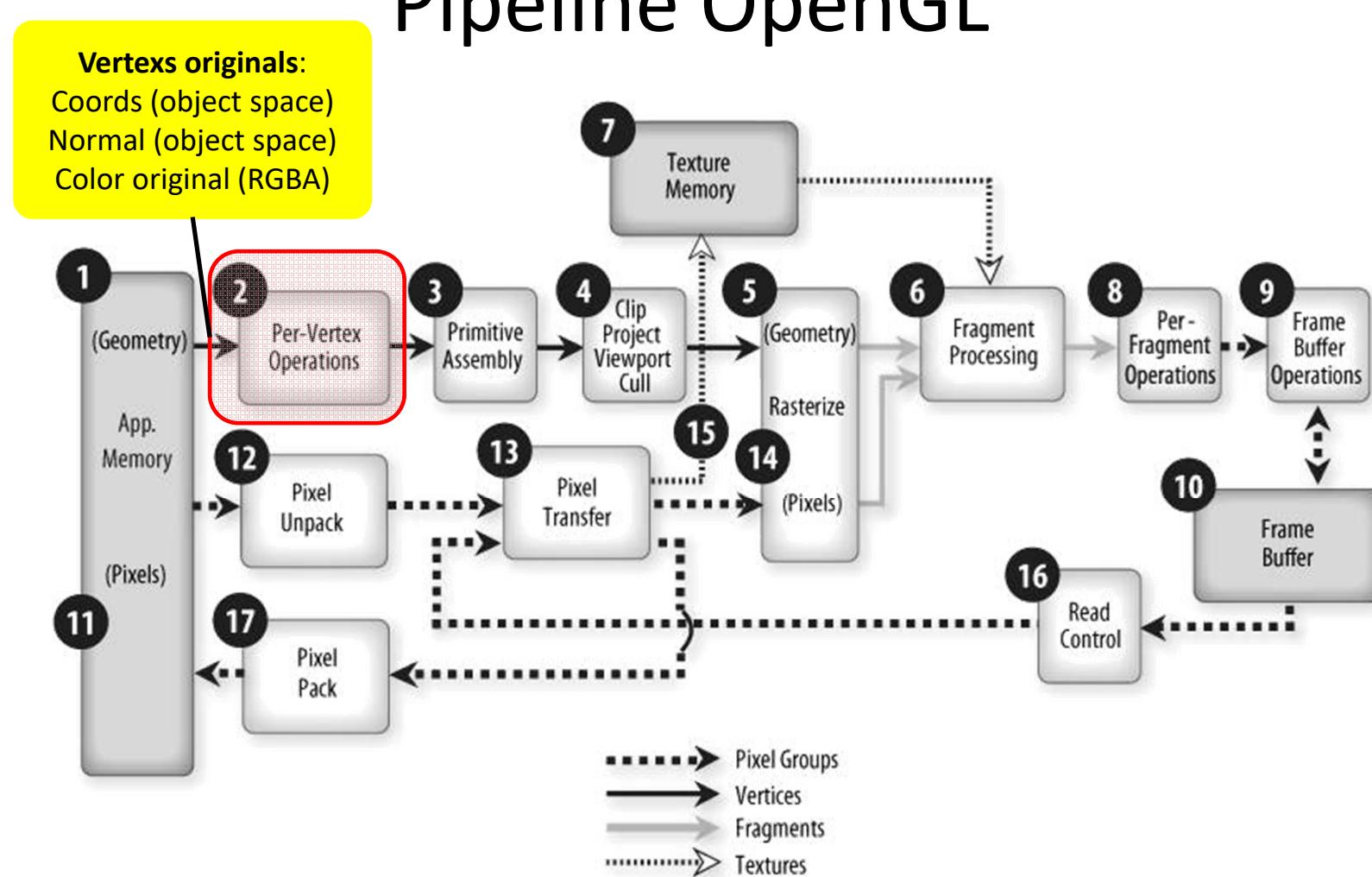
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
 glEnableVertexAttribArray(0);

glBindVertexArray(0);

// paintGL
glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, 0, numVerts);
glBindVertexArray(0);
```

Coords i normals en
object space

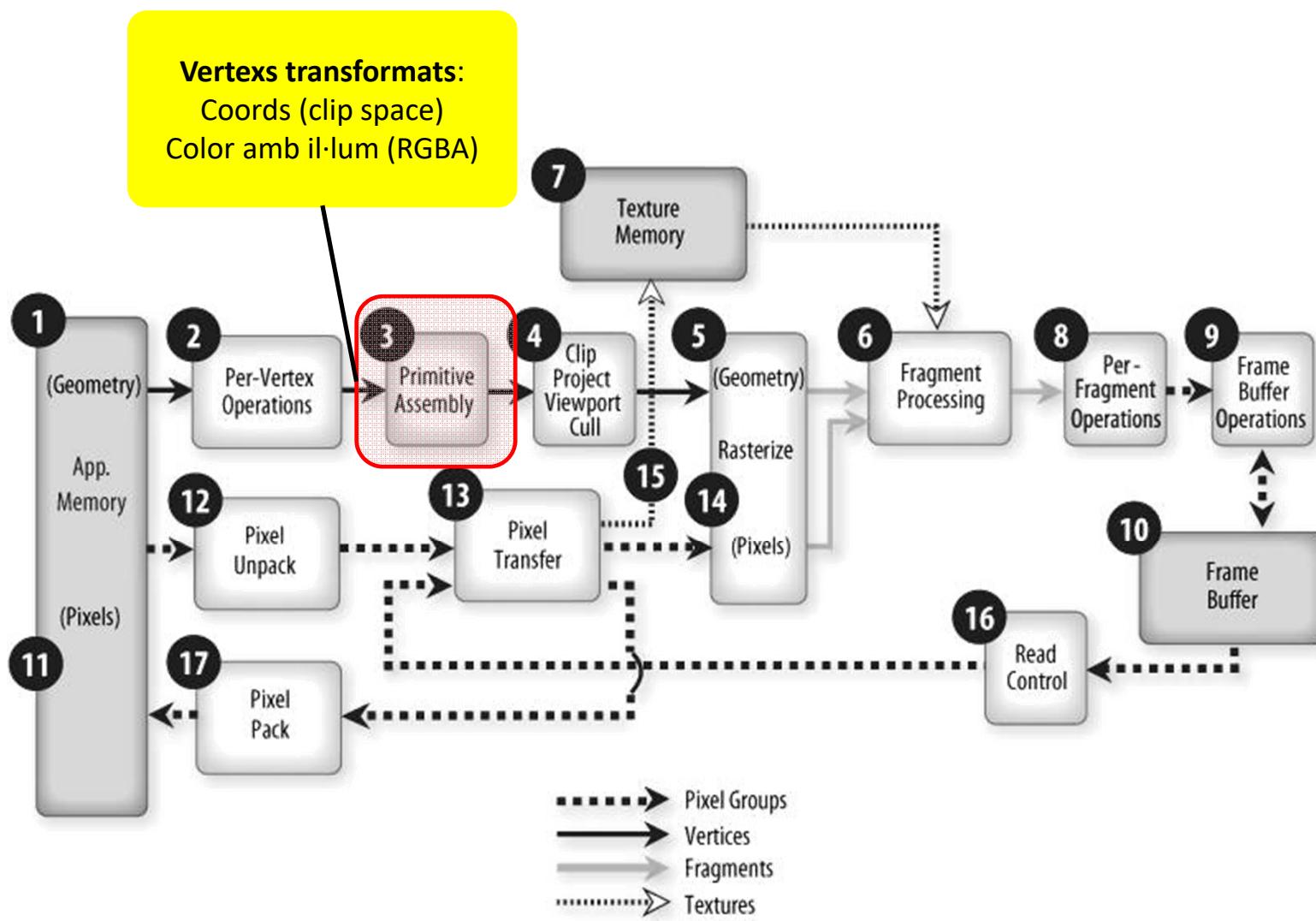
Pipeline OpenGL



Pipeline OpenGL

2. Per-vertex operations

- Es transformen els vèrtexs (modelview i projection).
- Es transformen les normals (trasposta de l'inversa de la submatriu 3x3 de la modelview) i es pot calcular la il·luminació del vèrtex.



Pipeline OpenGL

3. Primitive assembly

- Els vèrtexs s'agrupen formant primitives.
- Cada primitiva requereix un clipping diferent.

Pipeline OpenGL - glDrawArrays

```
glDrawArrays(mode, first, count);
```

vertex

	x	y	z
v0	1.0	0.0	0.0
v1			
v2			
v3			
v4			
v5			
v6			
v7			
...			

GL_POINTS

GL_LINES

GL_TRIANGLES

GL_TRIANGLE_STRIP

Pipeline OpenGL - glDrawArrays

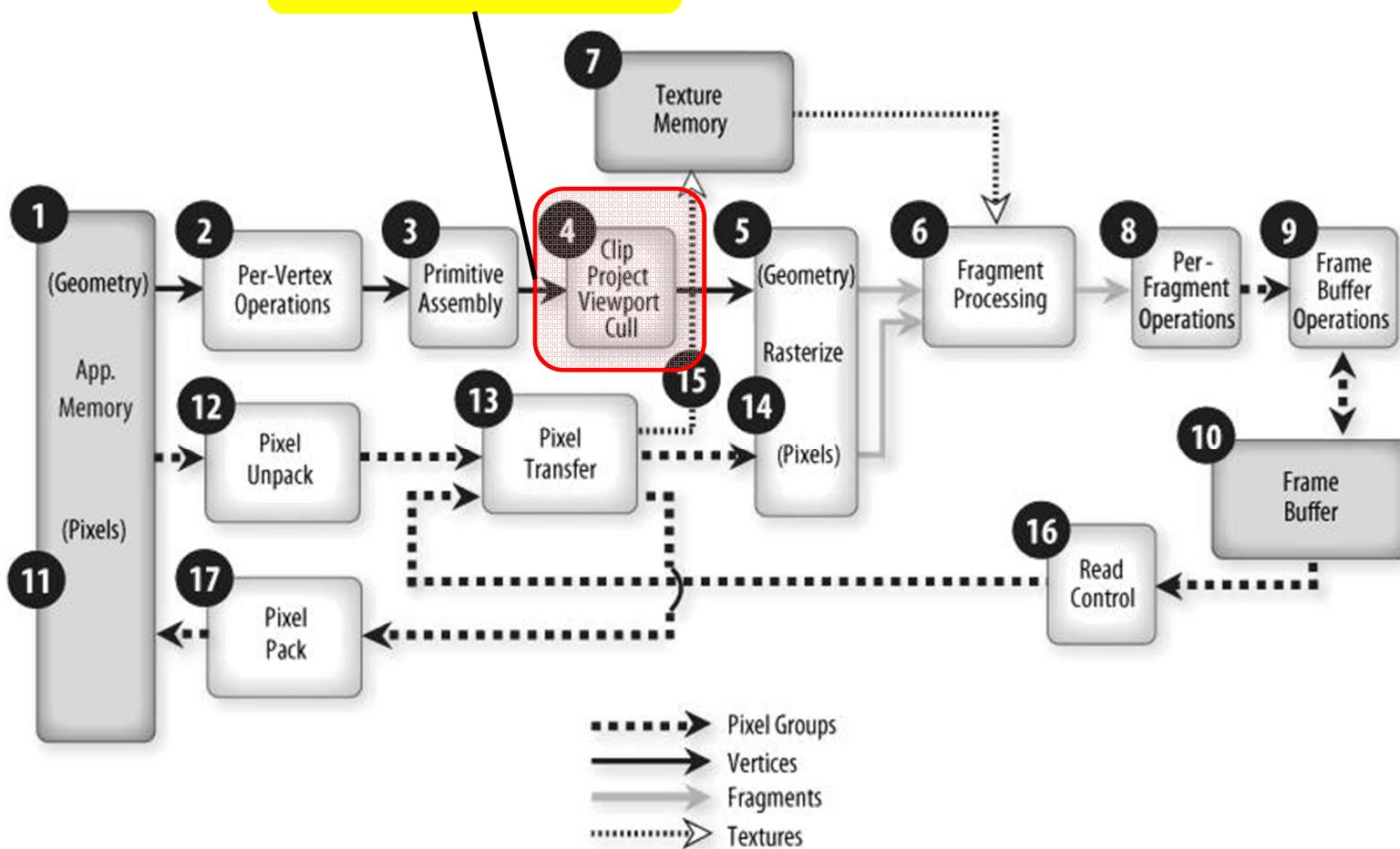
```
glDrawElements(mode, count, type, indices);
```

	<i>vertex</i>		<i>indices</i>
	x	y	z
v0	1.0	0.0	0.0
v1			
v2			
v3			
v4			
v5			
v6			
v7			

GL_TRIANGLES

GL_TRIANGLE_STRIP

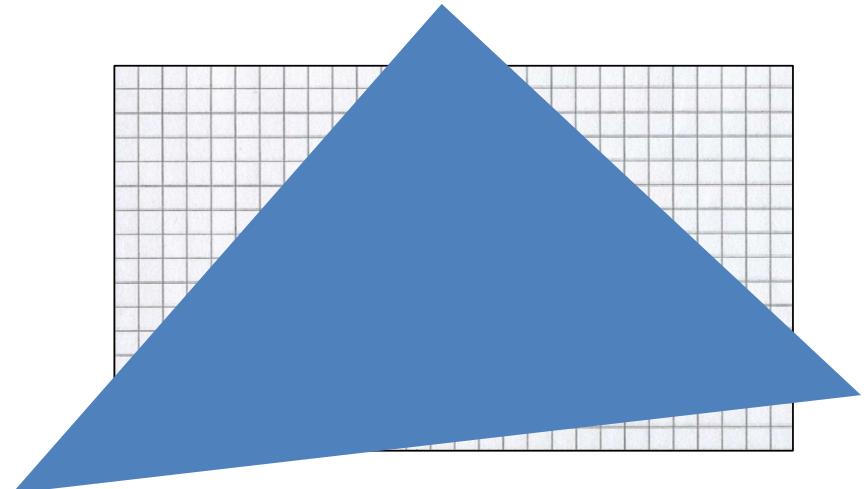
Primitives transformades: (clip space) i il·luminades

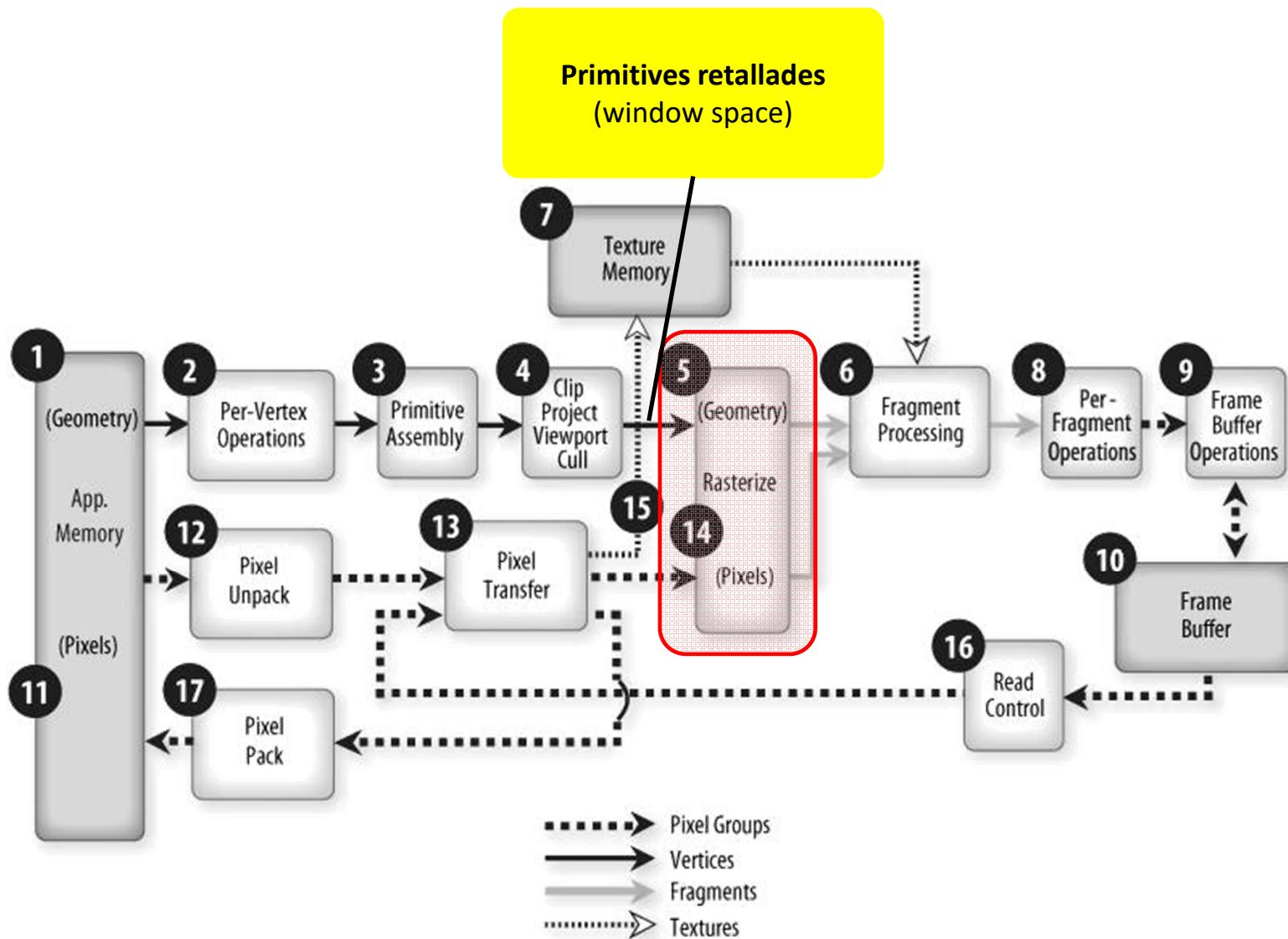


Pipeline OpenGL

4. Primitive processing

- Clipping (retallat) a la piràmide de visió.
- Divisió de perspectiva: es divideix (x,y,z) per w
- Viewport & depth transform → window space
- Backface culling – `glEnable(GL_CULL_FACE)`





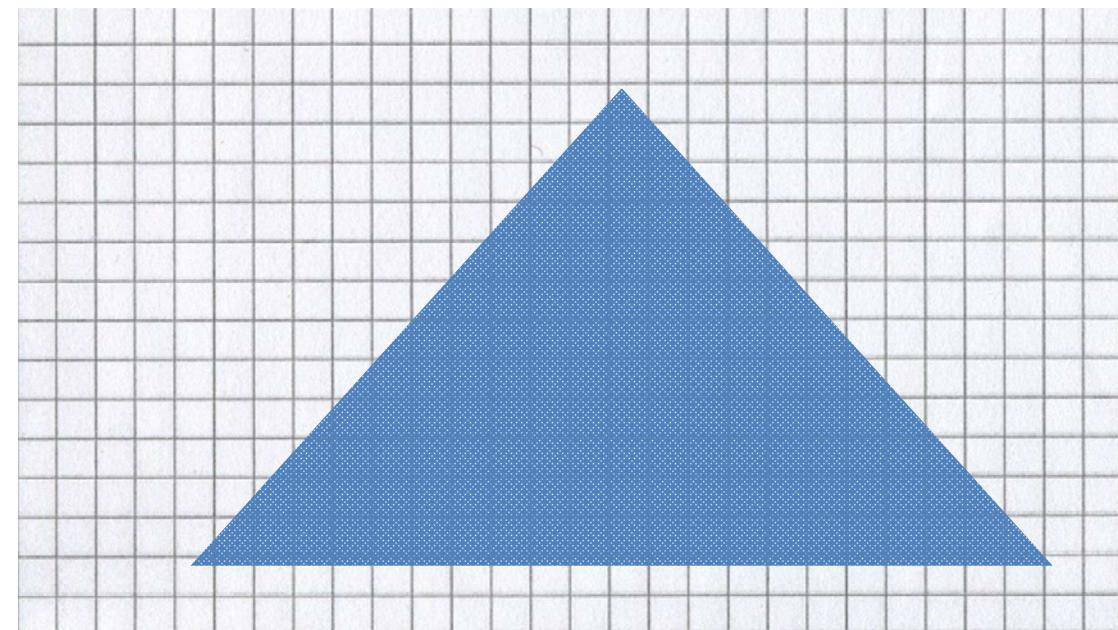
Pipeline OpenGL

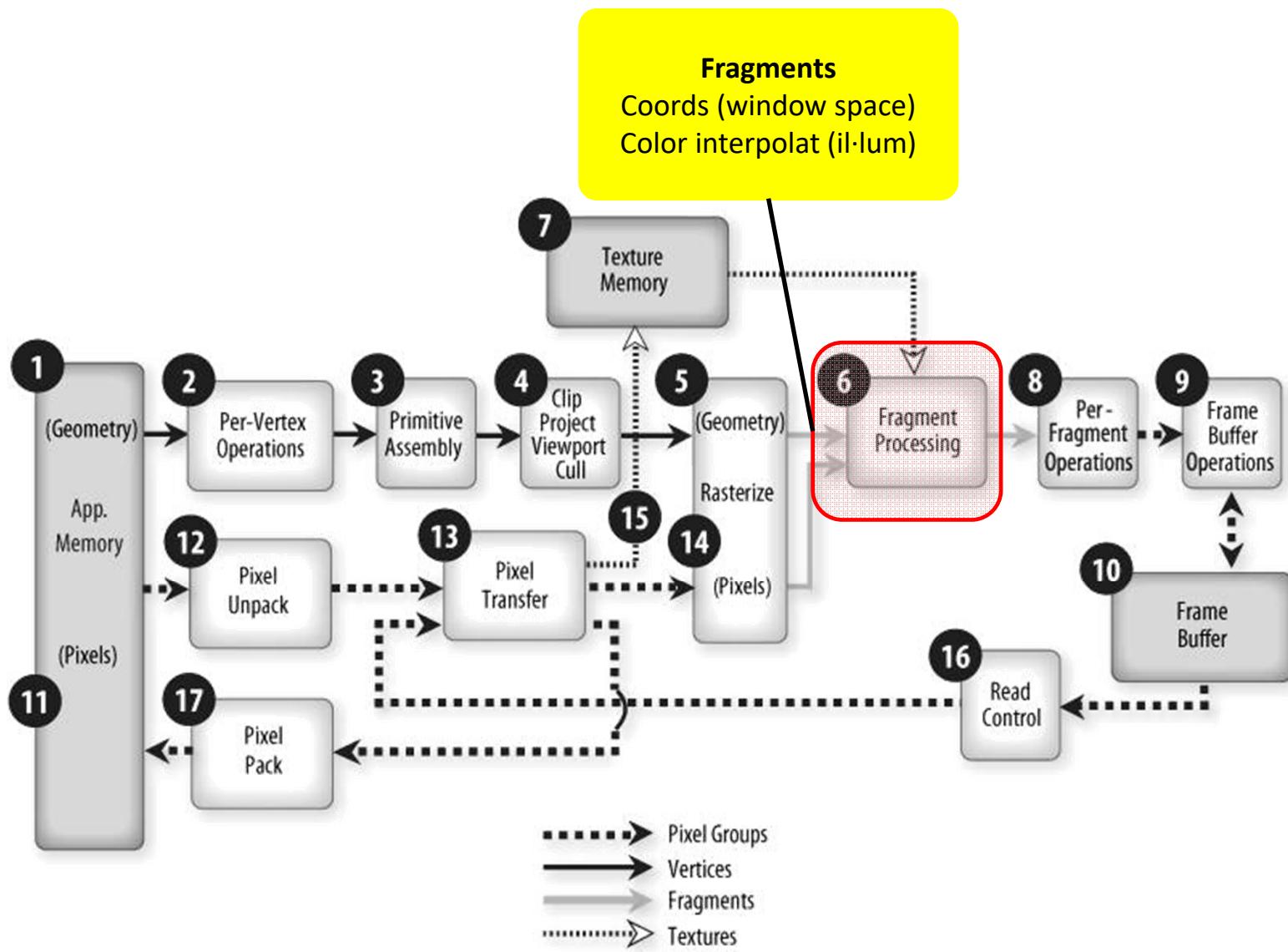
5. Rasterització

Generació dels fragments corresponents a la primitiva retallada.

Cada fragment té usualment diversos atributs:

- Coordenades (window space)
- Color (interpolat)
- Coordenades de textura (interpolades)

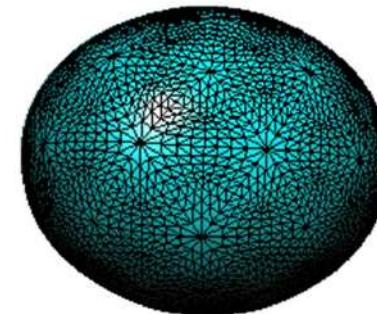
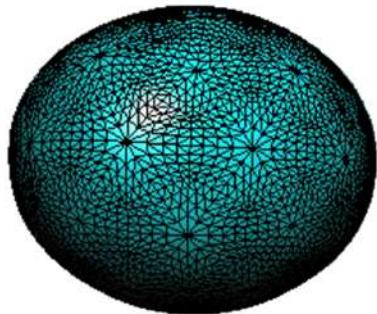


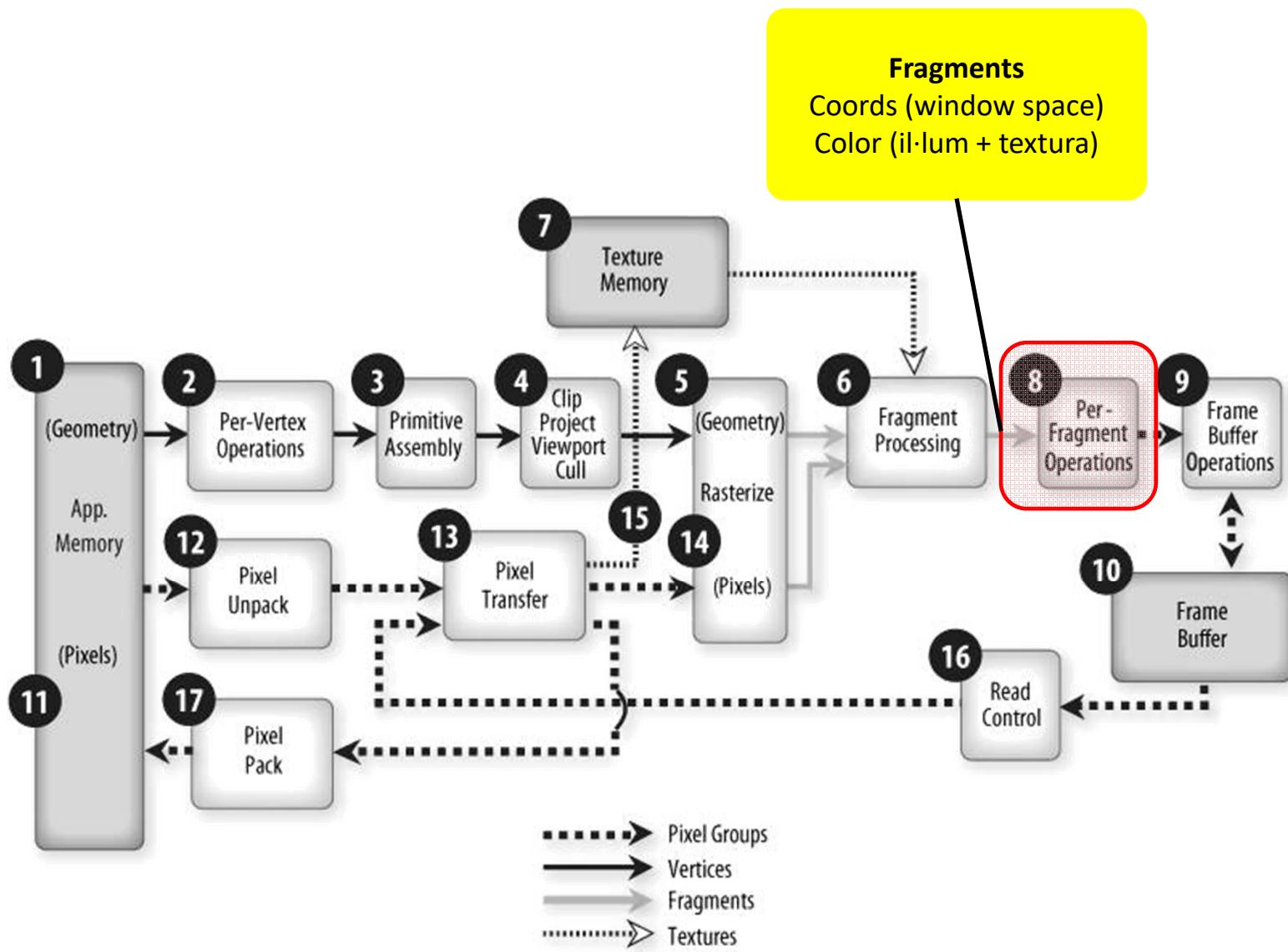


Pipeline OpenGL

6. Fragment processing (“shading”)

- Càlcul del color del fragment (texture mapping, il·lum per fragment, etc).





Pipeline OpenGL

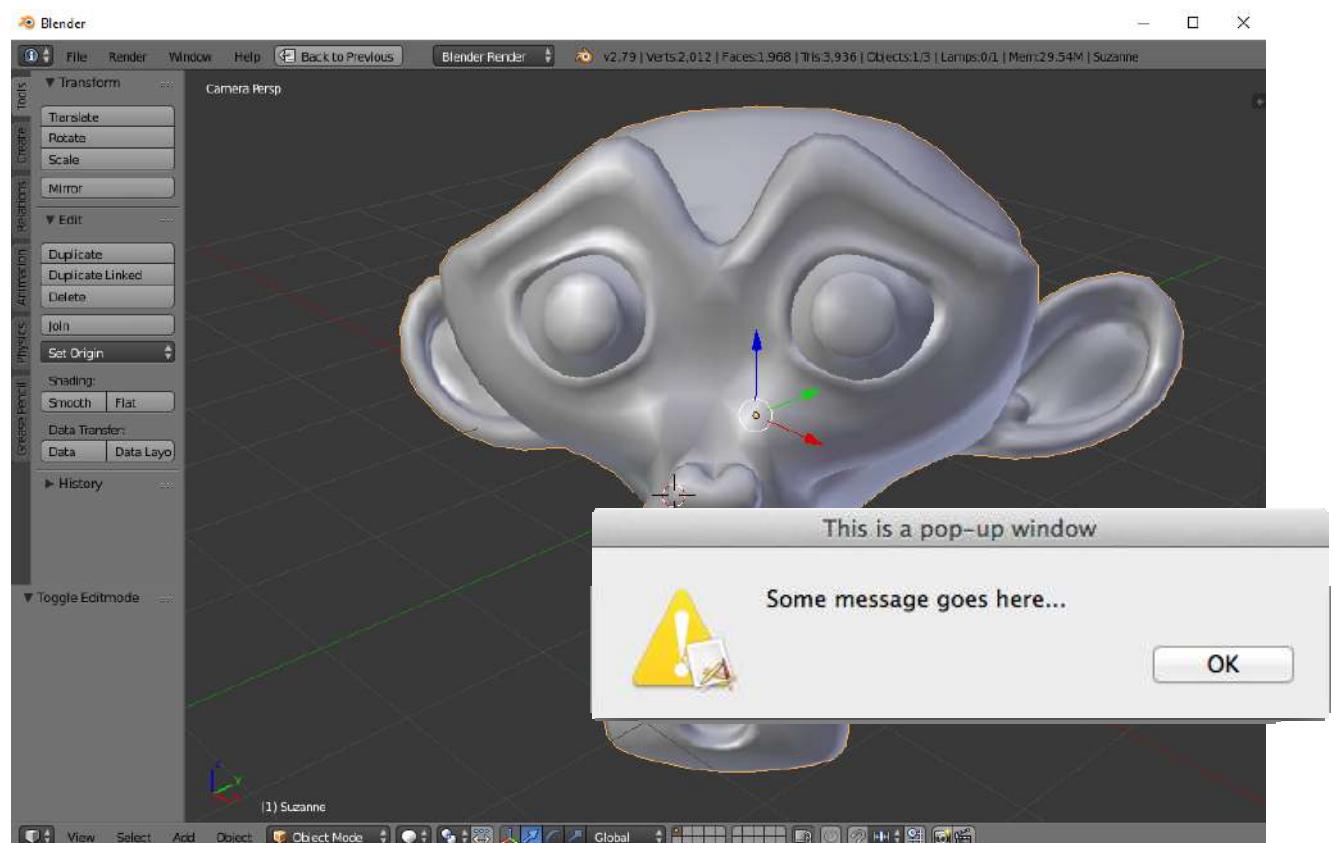
8. Per-fragment operations (“raster operations”)

- Pixel ownership test
- *Scissor test*
- *Alpha test*
- Stencil test
- Depth test (test Z-buffer)
- Blending
- *Dithering*
- *Logical Ops (glLogicOp)*

Pipeline OpenGL

8. Per-fragment operations (“raster operations”)

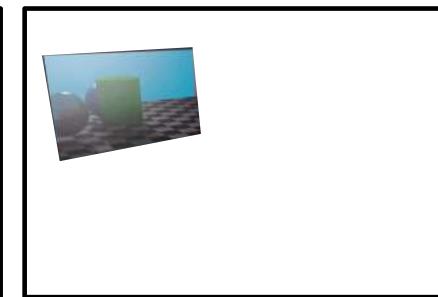
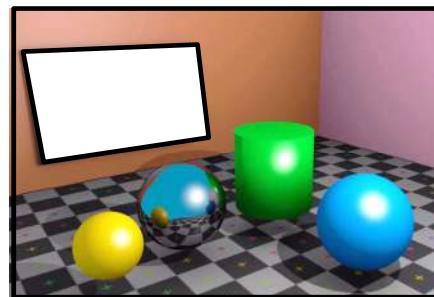
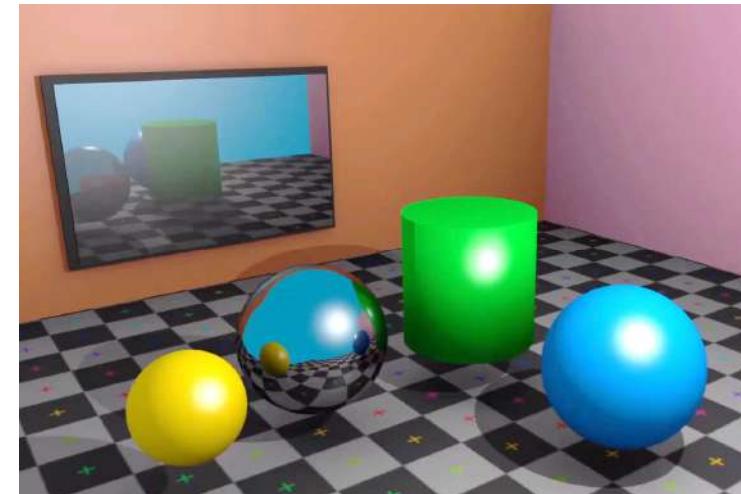
- **Pixel ownership test**
- *Scissor test*
- *Alpha test*
- Stencil test
- Depth test (test Z-buffer)
- Blending
- *Dithering*
- *Logical Ops (glLogicOp)*



Pipeline OpenGL

8. Per-fragment operations (“raster operations”)

- Pixel ownership test
- *Scissor test*
- *Alpha test*
- **Stencil test**
- Depth test (test Z-buffer)
- Blending
- *Dithering*
- *Logical Ops (glLogicOp)*



`glEnable(GL_STENCIL_TEST);`

`glStencilFunc(comparació, valorRef, mask)`
`GL_ALWAYS, GL_EQUAL, ...`

`glStencilOp(fail, zfail, zpass)`

`GL_KEEP, GL_ZERO, GL_INCR, GL_REPLACE`

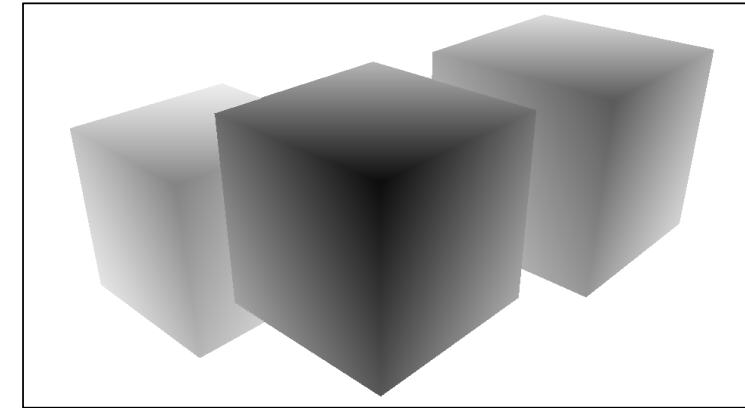
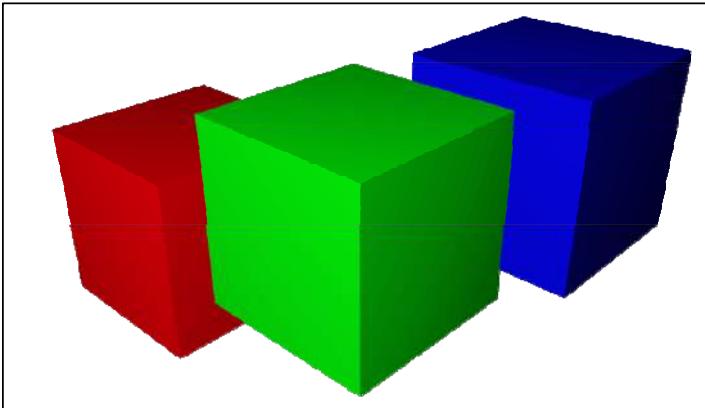
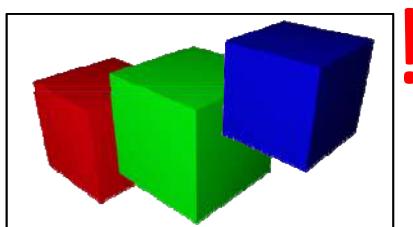
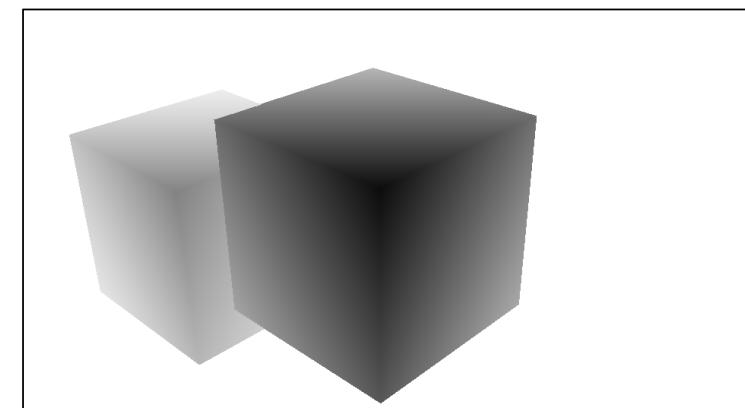
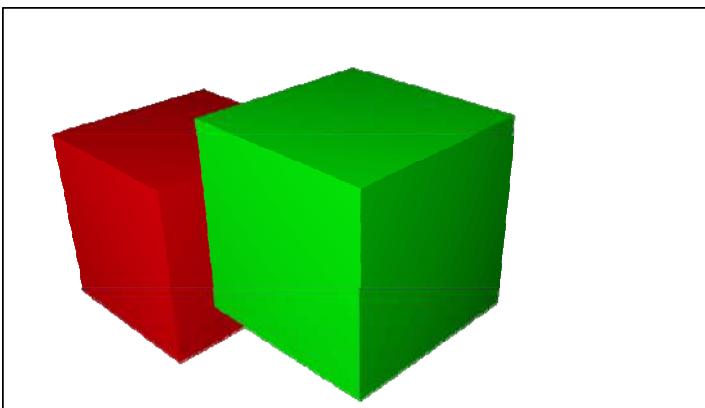
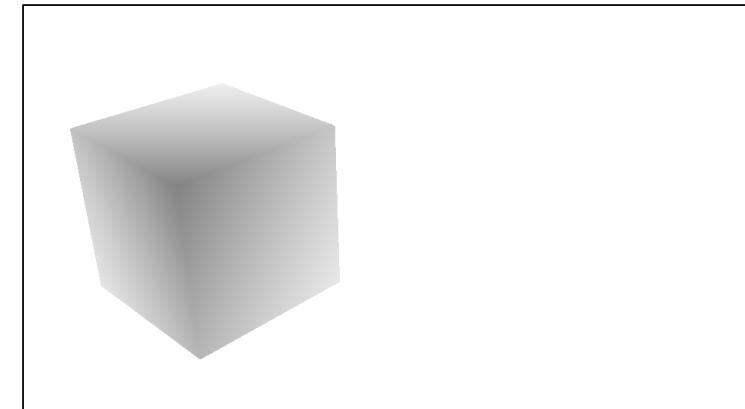
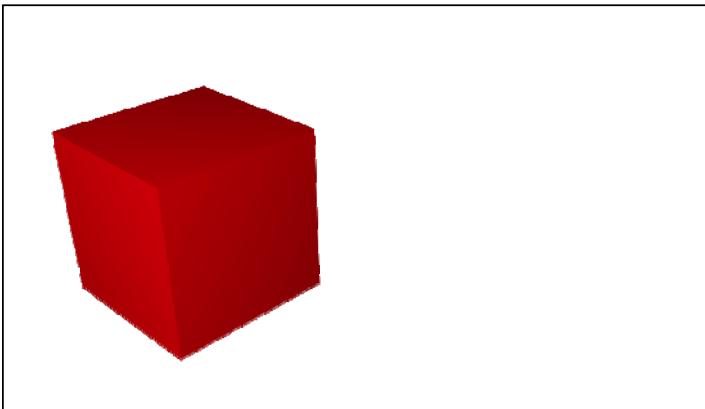
Pipeline OpenGL

8. Per-fragment operations (“raster operations”)

- Pixel ownership test
- *Scissor test*
- *Alpha test*
- Stencil test
- **Depth test (test Z-buffer)**
- Blending
- *Dithering*
- *Logical Ops (glLogicOp)*

```
glEnable(GL_DEPTH_TEST);  
glDepthFunc(GL_LESS);  
    GL_LESS, GL_EQUAL, GL_GREATER, ...
```



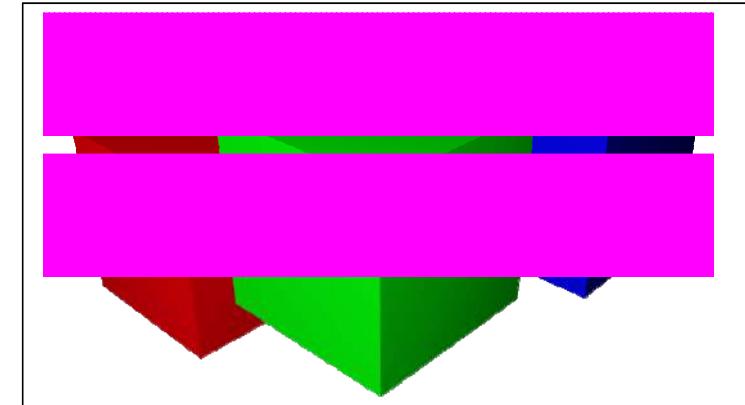
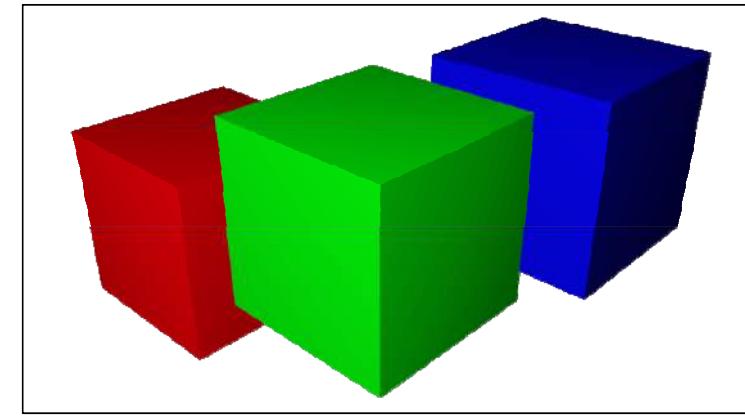


Pipeline OpenGL

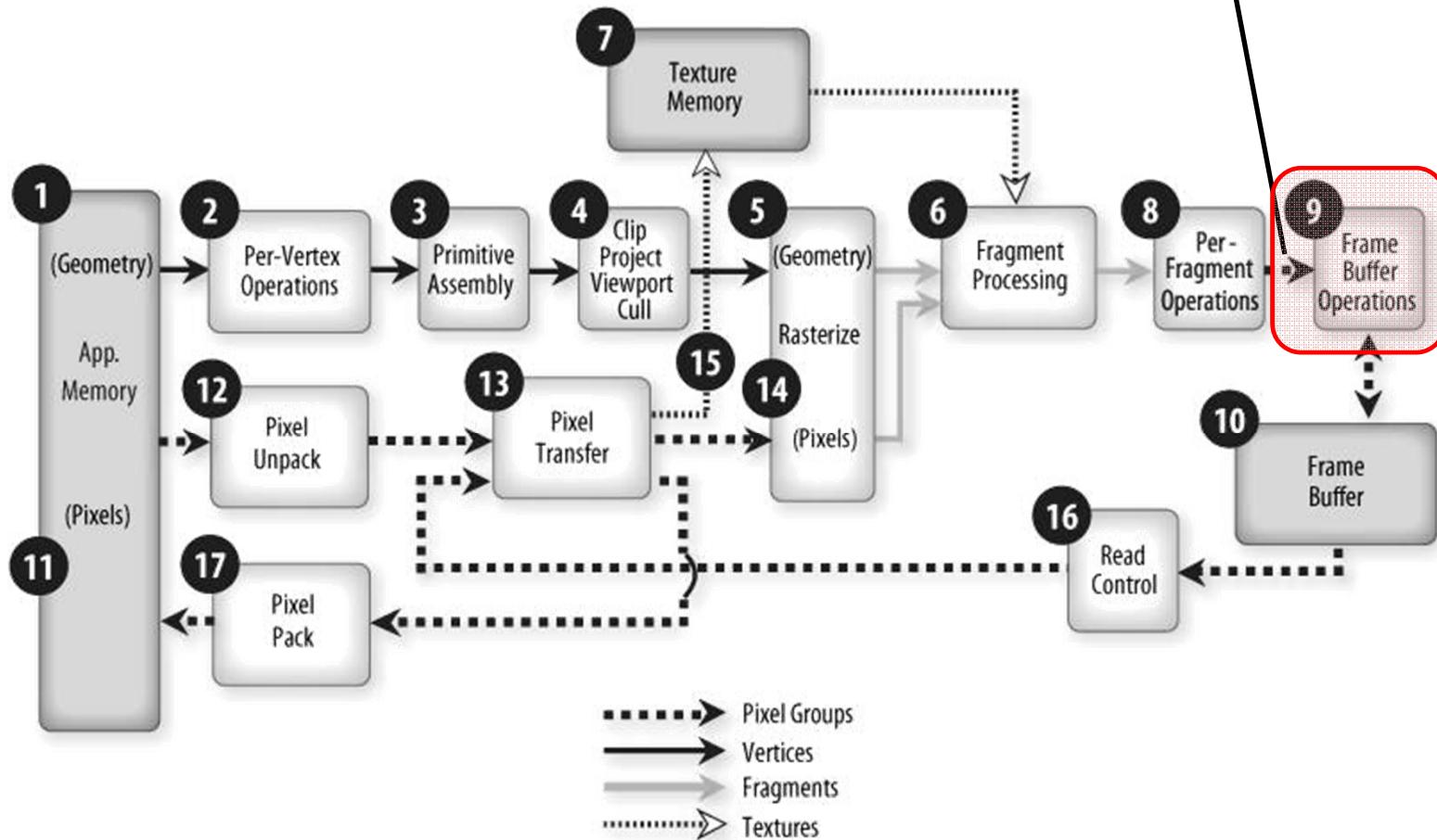
8. Per-fragment operations (“raster operations”)

- Pixel ownership test
- *Scissor test*
- *Alpha test*
- Stencil test
- Depth test (test Z-buffer)
- **Blending**
- *Dithering*
- *Logical Ops (glLogicOp)*

```
glEnable(GL_BLEND);  
glBlendFunc(...);
```



Fragments visibles
 Coords (window space)
 Color (il·lum + texture +
 alpha blending)



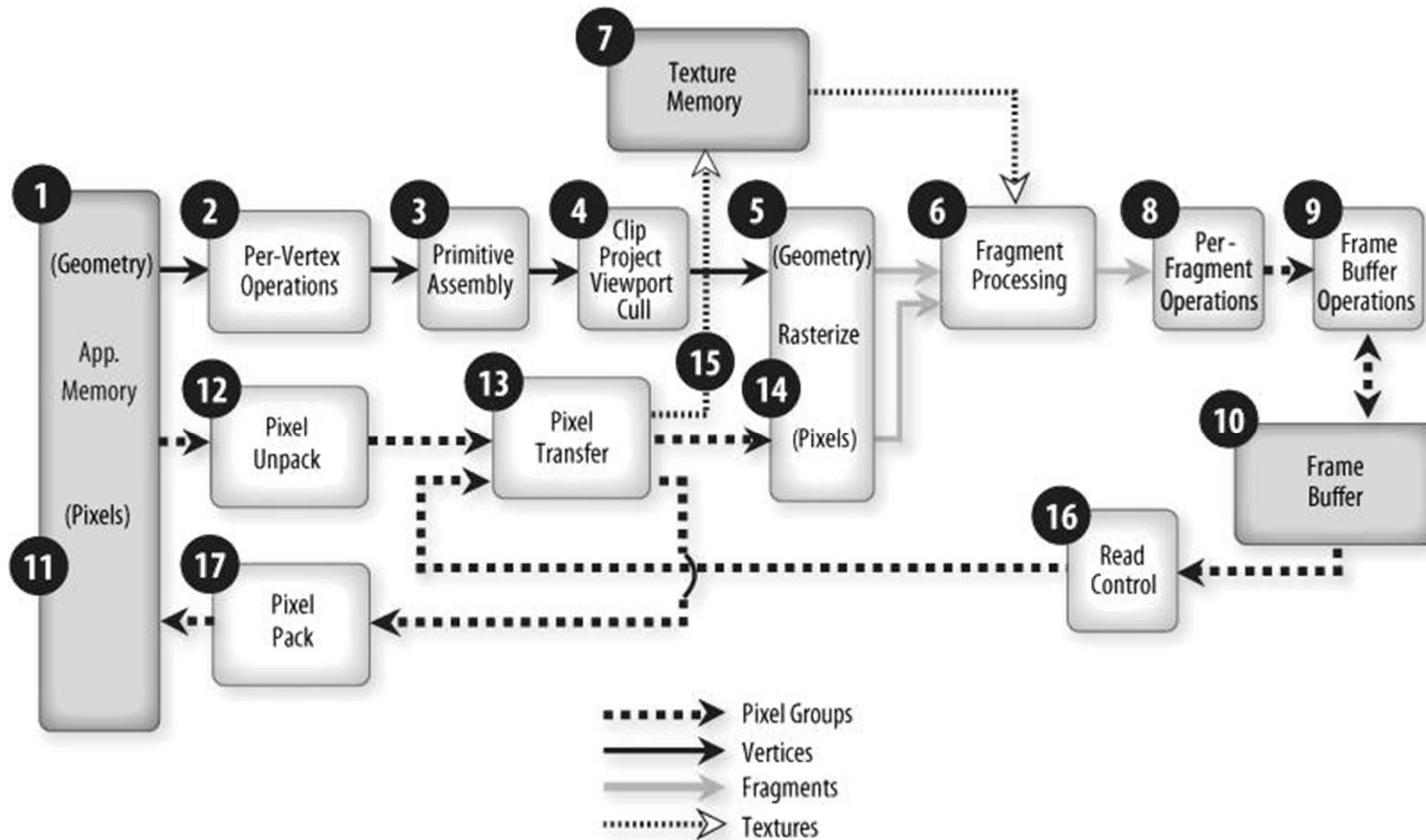
Pipeline OpenGL

9. Frame buffer operations

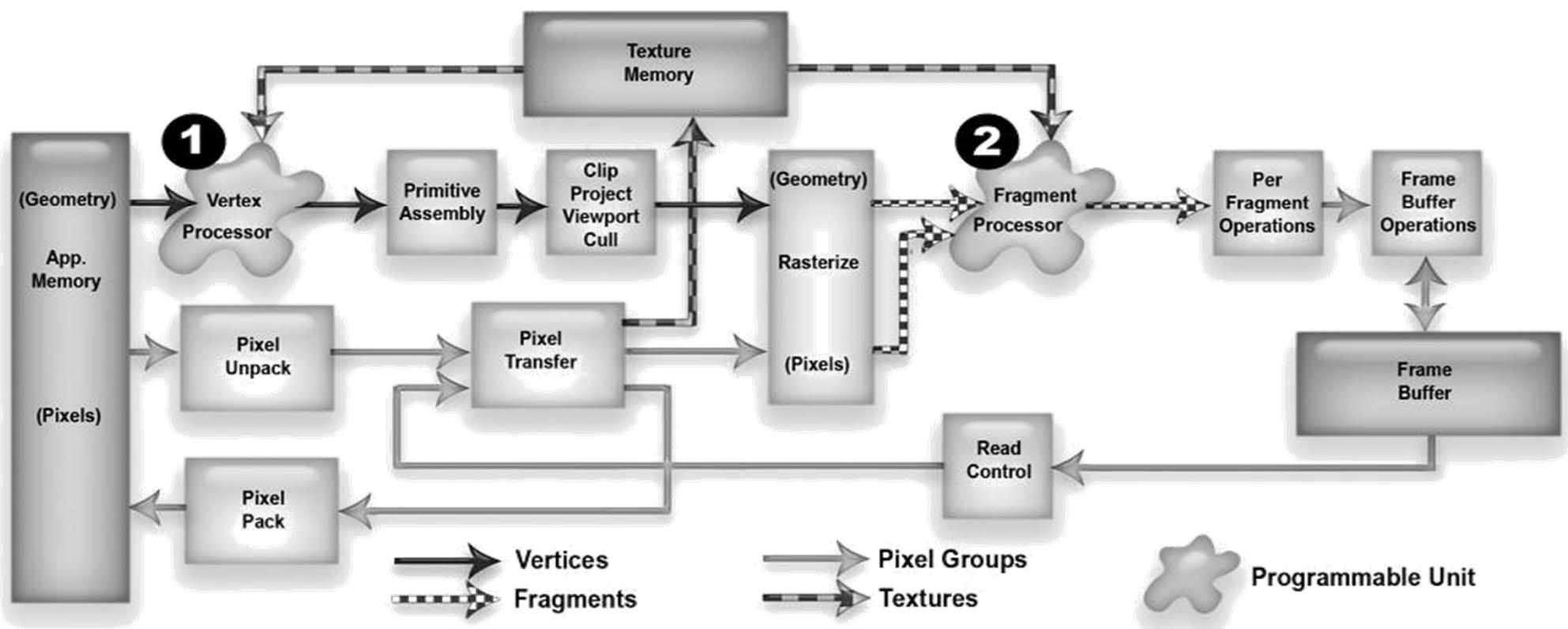
- Es modifiquen els buffers que s'hagin escollit amb glDrawBuffers
- Es veu afectada per glColorMask, glDepthMask...

PIPELINE PROGRAMABLE

Pipeline fix...



i pipeline programmable



Pipeline programable

- **Vertex processor:** part de la GPU capaç d'executar un programa per cada vèrtex.
- **Fragment processor:** part de la GPU capaç d'executar un programa per cada fragment.
- **Shader:** codi font d'un programa (o part) per la GPU
 - Vertex shader, fragment shader, geometry shader
- **Program:** executable d'un programa per la GPU
 - Vertex program, fragment program, geometry program

Pipeline programable

- Quan s'activa un **vertex program**, en lloc d'executar-se les operacions **per-vèrtex** prefixades d'OpenGL, s'executa el vertex program.
- Quan s'activa un **fragment program**, en lloc d'executar-se les operacions **per-fragment** prefixades d'OpenGL, s'executa el fragment program.
- Normalment el vertex/fragment program més senzill haurà de reproduir part de la funcionalitat fixa d'OpenGL.

Selecció d'elements 3D

C. Andujar i professors de gràfics

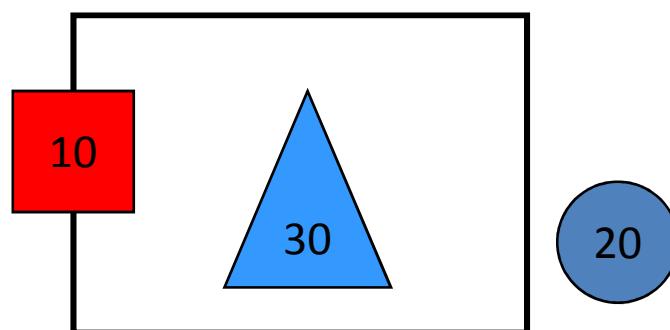
Abril 2012

MODE SELECCIÓ D'OPENGL

Exemple 1

```
glRenderMode(GL_SELECT);
...
for (int i=0; i<n; i++)
{
    glPushName(id);
    glBegin(GL_POLYGON);
    glVertex3f(...)
    ...
    glEnd();
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```

i	pila de noms (ID)	buffer selecció	hits
			0



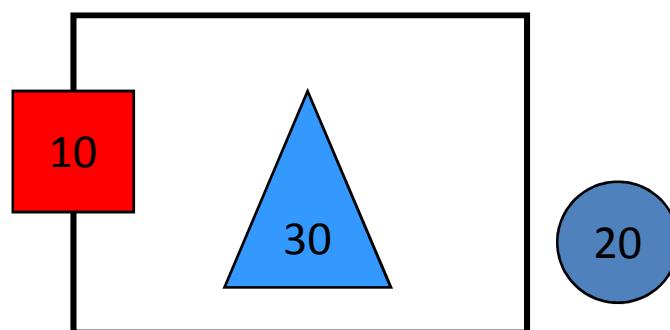
Exemple 1

```
glRenderMode(GL_SELECT);
...
for (int i=0; i<n; i++)
{
    glPushName(id);
    glBegin(GL_POLYGON);
    glVertex3f(...)

    ...
    glEnd();
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```

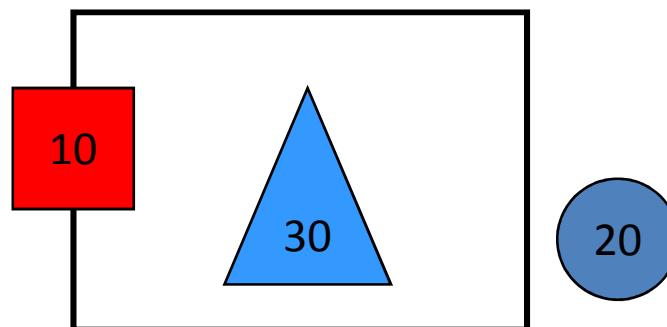


i	pila de noms (ID)	buffer selecció	hits
0	10		0



Exemple 1

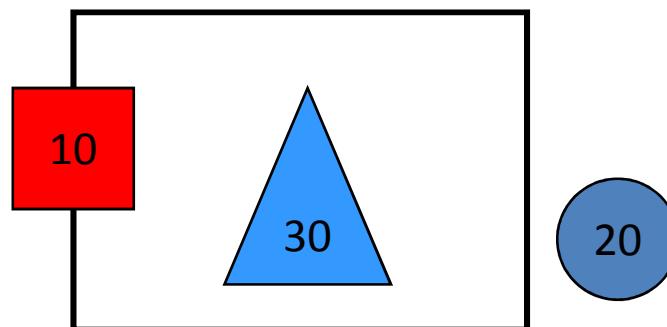
```
glRenderMode(GL_SELECT);
...
for (int i=0; i<n; i++)
{
    glPushName(id);
    glBegin(GL_POLYGON);
    glVertex3f(...)
    ...
    glEnd();
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```



i	pila de noms (ID)	buffer selecció	hits
0	10	1 minZ maxZ 10	1

Exemple 1

```
glRenderMode(GL_SELECT);
...
for (int i=0; i<n; i++)
{
    glPushName(id);
    glBegin(GL_POLYGON);
    glVertex3f(...)
    ...
    glEnd();
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```



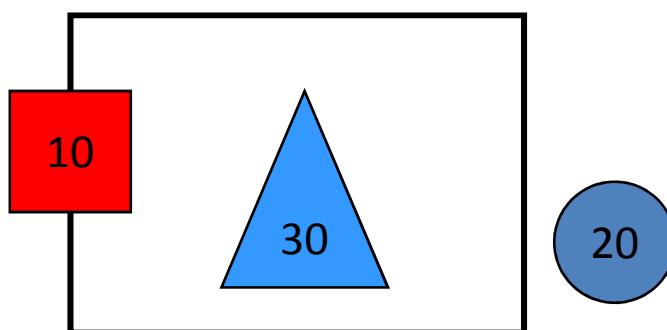
i	pila de noms (ID)	buffer selecció	hits
0		1 minZ maxZ 10	1

Exemple 1

```
glRenderMode(GL_SELECT);
...
for (int i=0; i<n; i++)
{
    glPushName(id);
    glBegin(GL_POLYGON);
    glVertex3f(...)

    ...
    glEnd();
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```

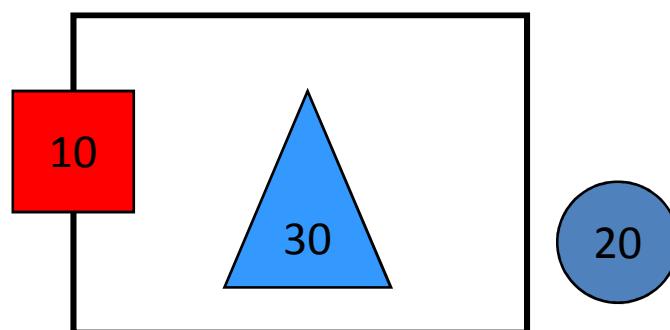
i	pila de noms (ID)	buffer selecció	hits
1	20	1 minZ maxZ 10	1



Exemple 1

```
glRenderMode(GL_SELECT);
...
for (int i=0; i<n; i++)
{
    glPushName(id);
    glBegin(GL_POLYGON);
    glVertex3f(...)
    ...
    glEnd();
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```

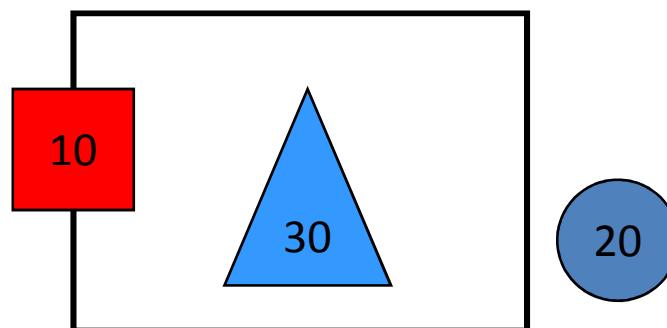
i	pila de noms (ID)	buffer selecció	hits
1		1 minZ maxZ 10	1



Exemple 1

```
glRenderMode(GL_SELECT);
...
for (int i=0; i<n; i++)
{
    glPushName(id);
    glBegin(GL_POLYGON);
    glVertex3f(...)

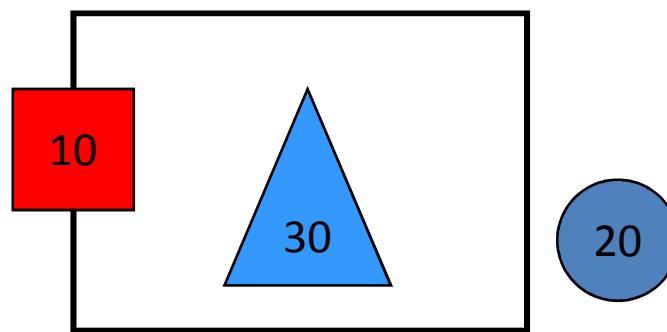
    ...
    glEnd();
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```



i	pila de noms (ID)	buffer selecció	hits
2	30	1 minZ maxZ 10	1

Exemple 1

```
glRenderMode(GL_SELECT);  
...  
for (int i=0; i<n; i++)  
{  
    glPushName(id);  
    glBegin(GL_POLYGON);  
    glVertex3f(...)  
    ...  
    glEnd();  
    glPopName();  
}  
hits = glRenderMode(GL_RENDER);
```

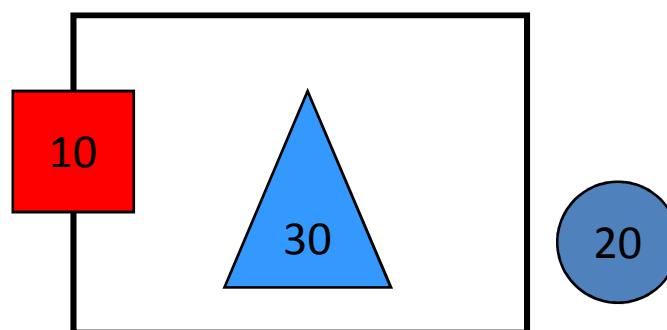


i	pila de noms (ID)	buffer selecció	hits
2	30	1 minZ maxZ 10 1 minZ' maxZ' 30	2

1
0

Exemple 1

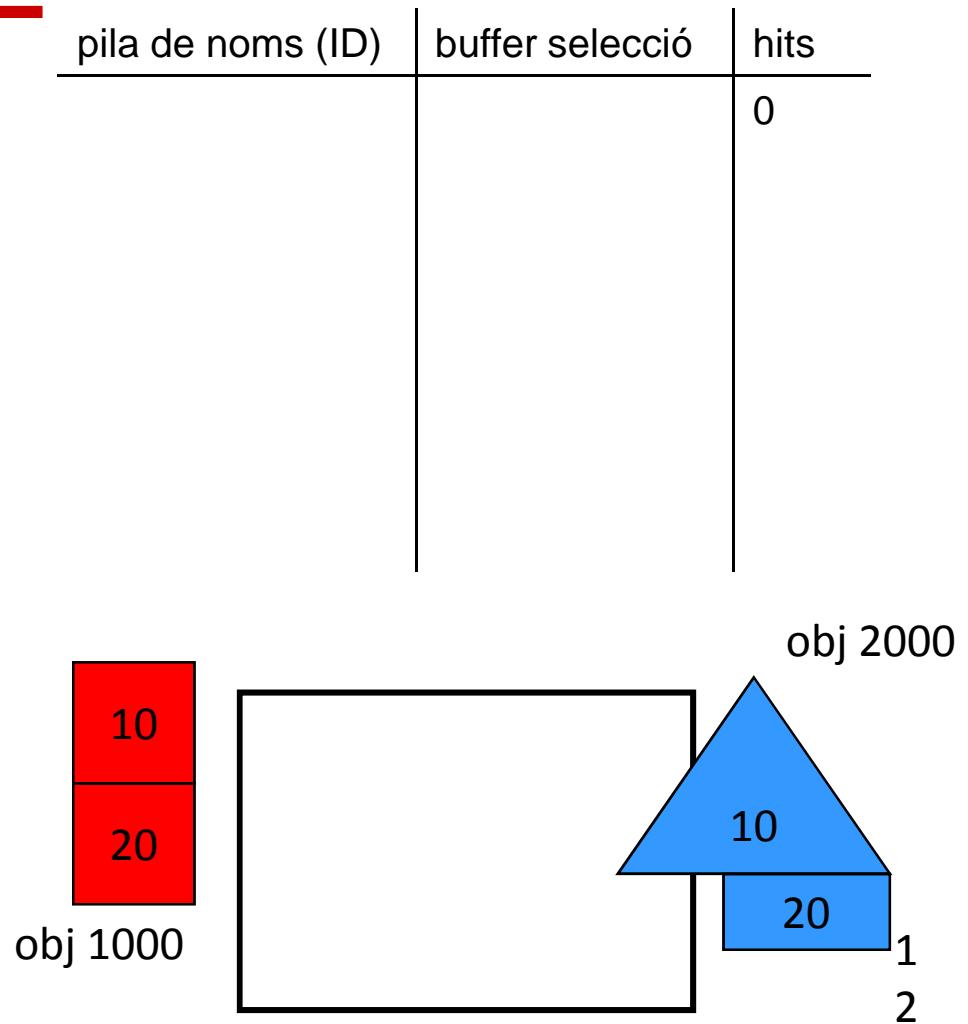
```
glRenderMode(GL_SELECT);  
...  
for (int i=0; i<n; i++)  
{  
    glPushName(id);  
    glBegin(GL_POLYGON);  
    glVertex3f(...)  
    ...  
    glEnd();  
    glPopName();  
}  
hits = glRenderMode(GL_RENDER);
```



i	pila de noms (ID)	buffer selecció	hits
2		1 minZ maxZ 10 1 minZ' maxZ' 30	2

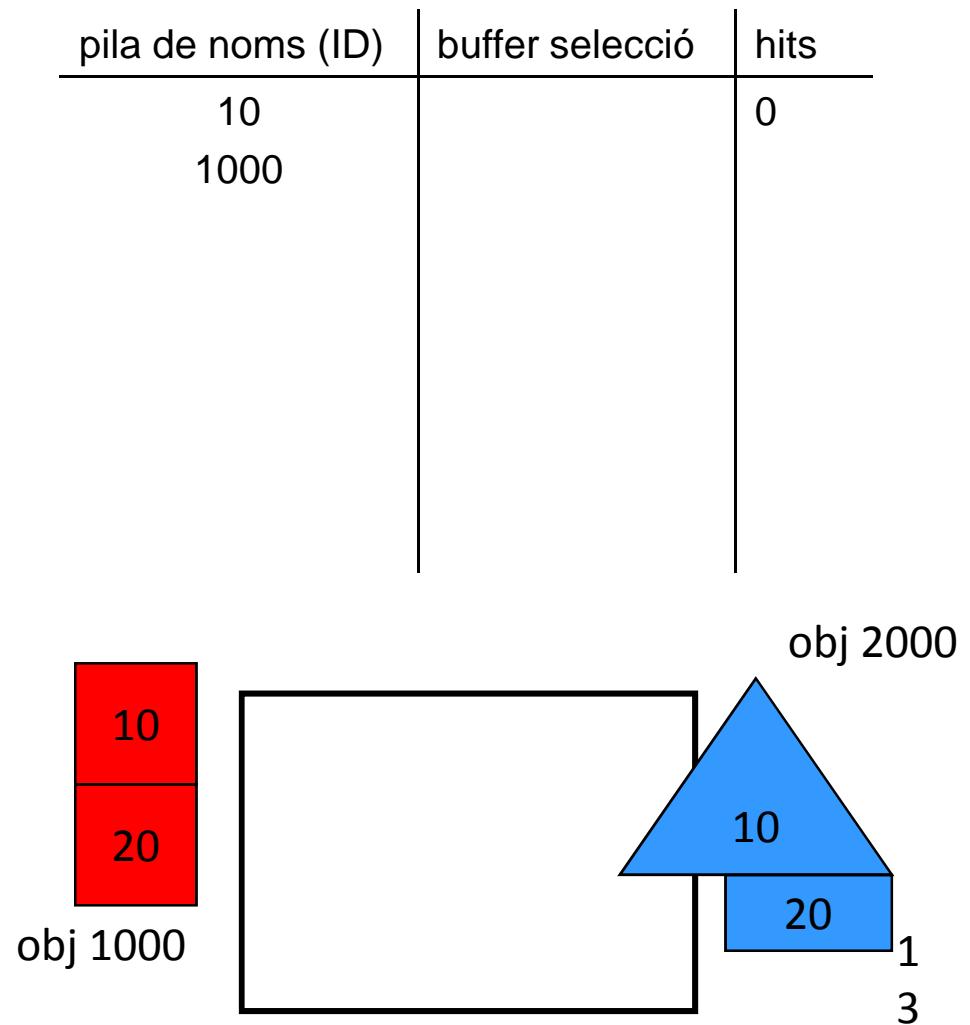
Exemple 2

```
glRenderMode(GL_SELECT);
per cada objecte ←
{
    glPushName(id_objecte);
    per cada cara
    {
        glPushName(id_cara);
        glBegin(GL_POLYGON);
        ...
        glEnd();
        glPopName();
    }
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```



Exemple 2

```
glRenderMode(GL_SELECT);
per cada objecte
{
    glPushName(id_objecte);
    per cada cara
    {
        glPushName(id_cara);
        glBegin(GL_POLYGON);
        ...
        glEnd();
        glPopName();
    }
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```



Exemple 2

```
glRenderMode(GL_SELECT);
per cada objecte
{
    glPushName(id_objecte);
    per cada cara
    {
        glPushName(id_cara);
        glBegin(GL_POLYGON);
        ...
        glEnd();
        glPopName();
    }
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```

pila de noms (ID)	buffer selecció	hits
20		0
1000		

obj 2000

obj 1000

10

20

10

20

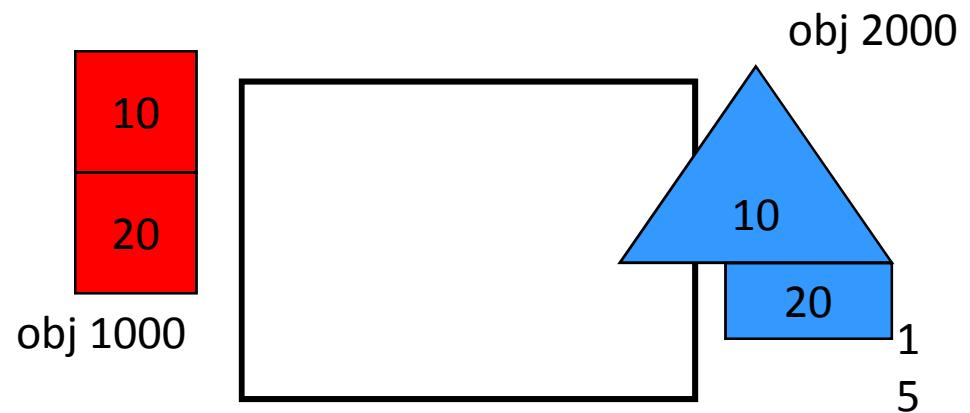
1

4

Exemple 2

```
glRenderMode(GL_SELECT);
per cada objecte
{
    glPushName(id_objecte);
    per cada cara
    {
        glPushName(id_cara);
        glBegin(GL_POLYGON);
        ...
        glEnd();
        glPopName();
    }
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```

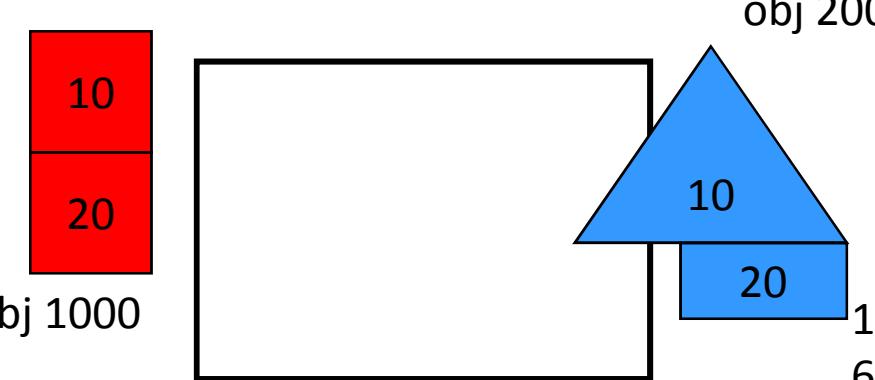
pila de noms (ID)	buffer selecció	hits
10	2	1
2000	minZ	
	maxZ	
	2000	
	10	



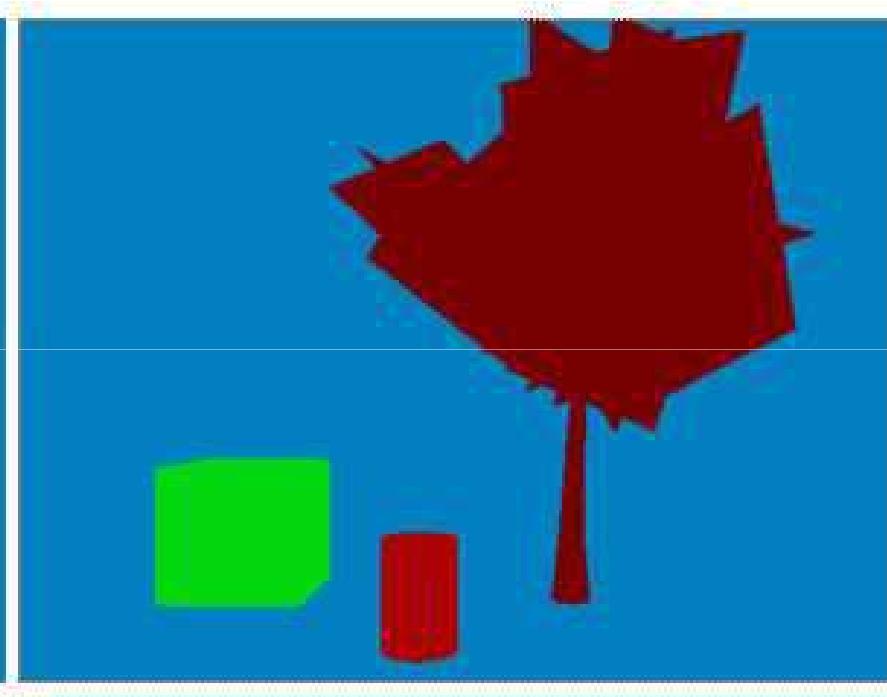
Exemple 2

```
glRenderMode(GL_SELECT);
per cada objecte
{
    glPushName(id_objecte);
    per cada cara
    {
        glPushName(id_cara);
        glBegin(GL_POLYGON);
        ...
        glEnd();
        glPopName();
    }
    glPopName();
}
hits = glRenderMode(GL_RENDER);
```

pila de noms (ID)	buffer selecció	hits
	2	1
	minZ	
	maxZ	
	2000	
	10	
obj 1000		
10		
20		
obj 2000		
10		
20		
1		
6		



LECTURA DEL BUFFER DE COLOR



SELECCIÓ AMB TESTS D'INTERSECCIÓ

Desprojecció d'un punt

```
GLdouble modelview[16];
glGetDoublev(GL_MODELVIEW_MATRIX, modelview);

GLdouble projection[16];
glGetDoublev(GL_PROJECTION_MATRIX, projection);

GLint viewport[4];
glGetIntegerv(GL_VIEWPORT, viewport);

GLfloat winX = float(mouseX);
GLfloat winY = float(viewport[3] - mouseY);
GLfloat winZ;
glReadPixels(mouseX, viewport[3]-mouseY, 1, 1, GL_DEPTH_COMPONENT, GL_FLOAT, &winZ);

GLdouble posX, posY, posZ;
gluUnProject(winX, winY, winZ, modelview, projection, viewport, &posX, &posY, &posZ);
```

Textures

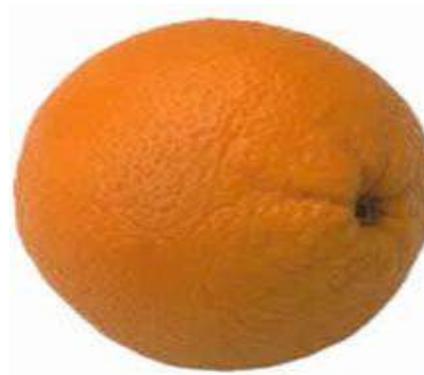
Carlos Andujar

Feb 2021

INTRODUCCIÓ

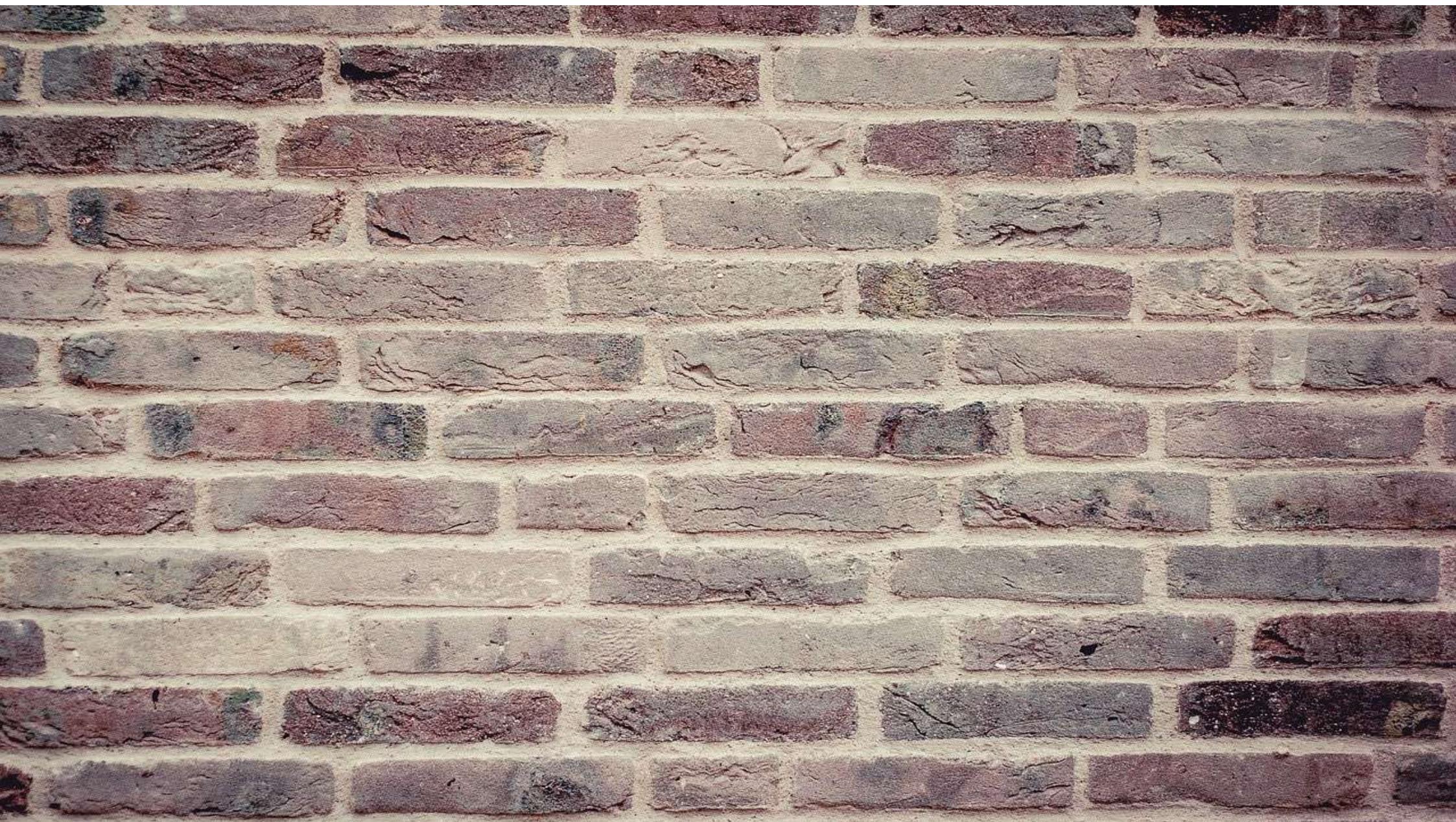
Representació de detalls superficials

Variacions de la *geometria*:

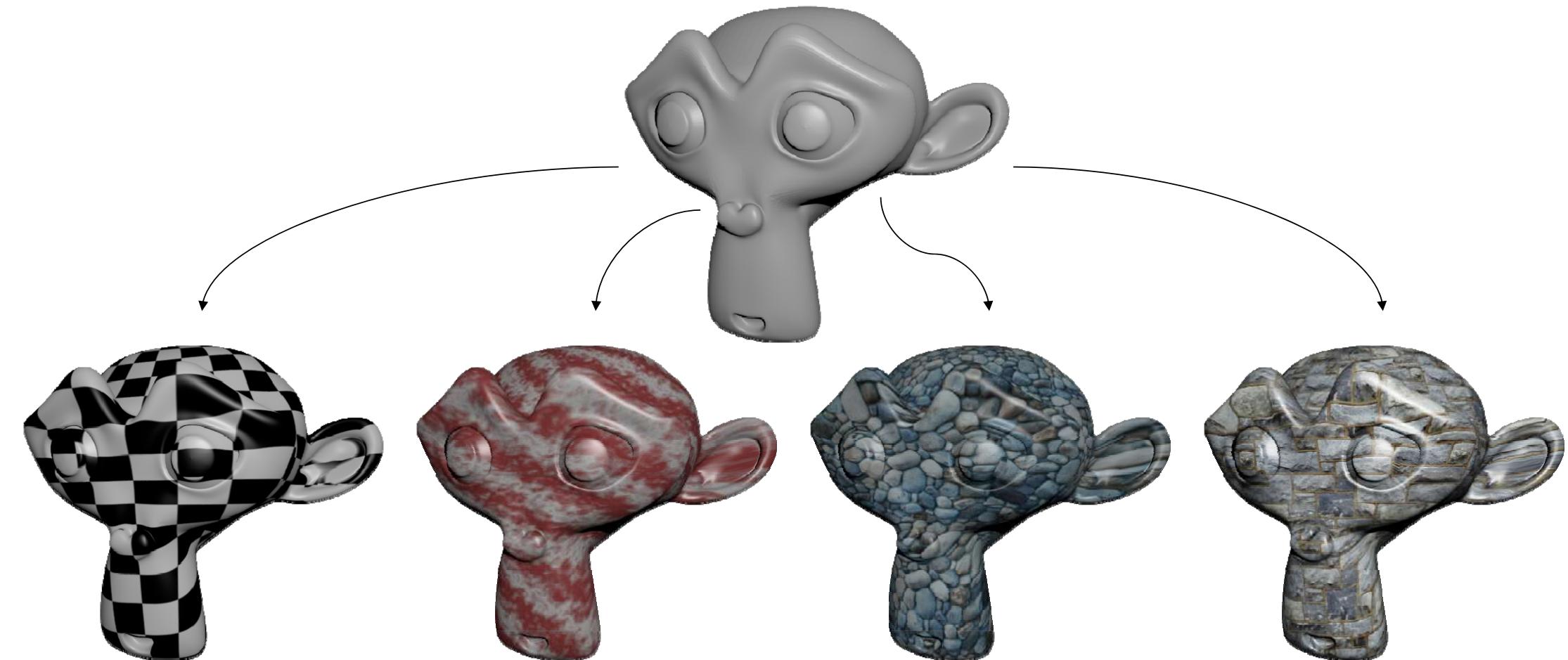


Variacions de les *propietats òptiques*:



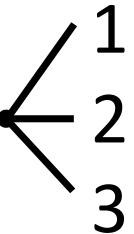


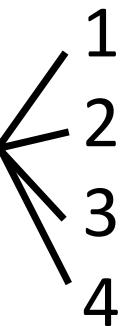
Representació de detalls superficials



DEFINICIONS

Textures

Una textura és una taula de  dimensions, on cada cel·la

enmagatzema una certa propietat amb  canals.

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

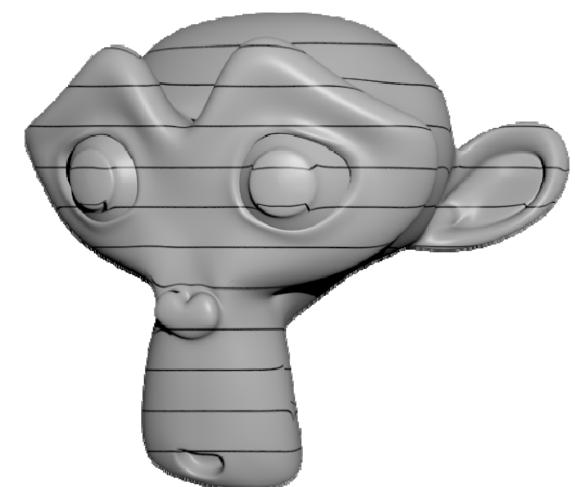
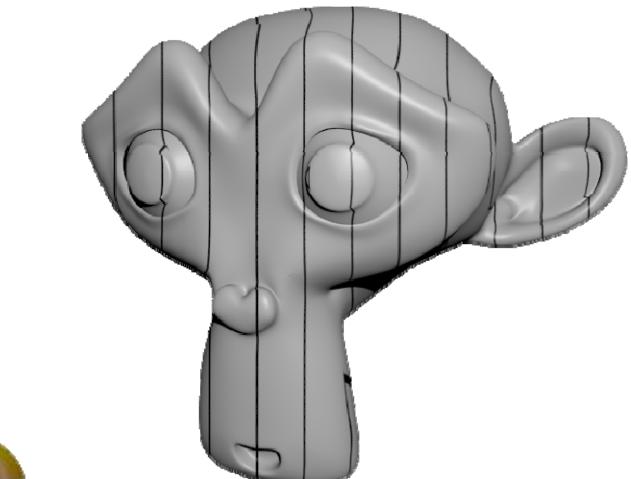
Textures

Una textura és una taula de  dimensions, on cada cel·la enmagatzema una certa propietat amb  canals.

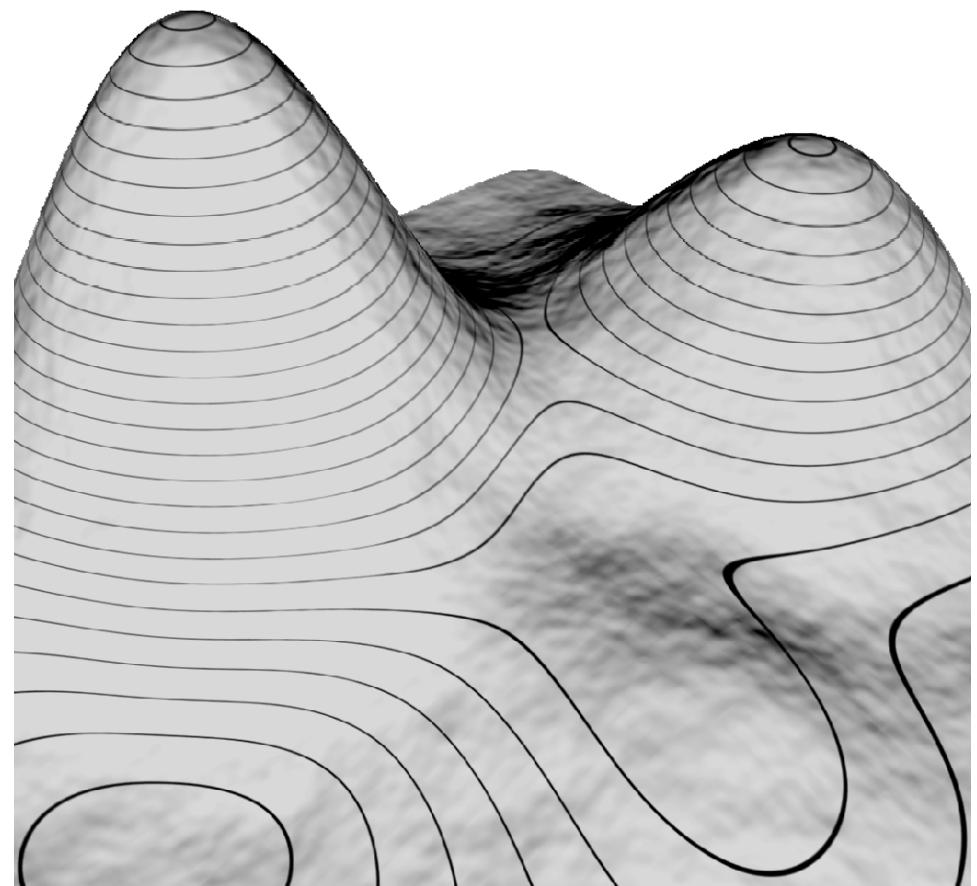
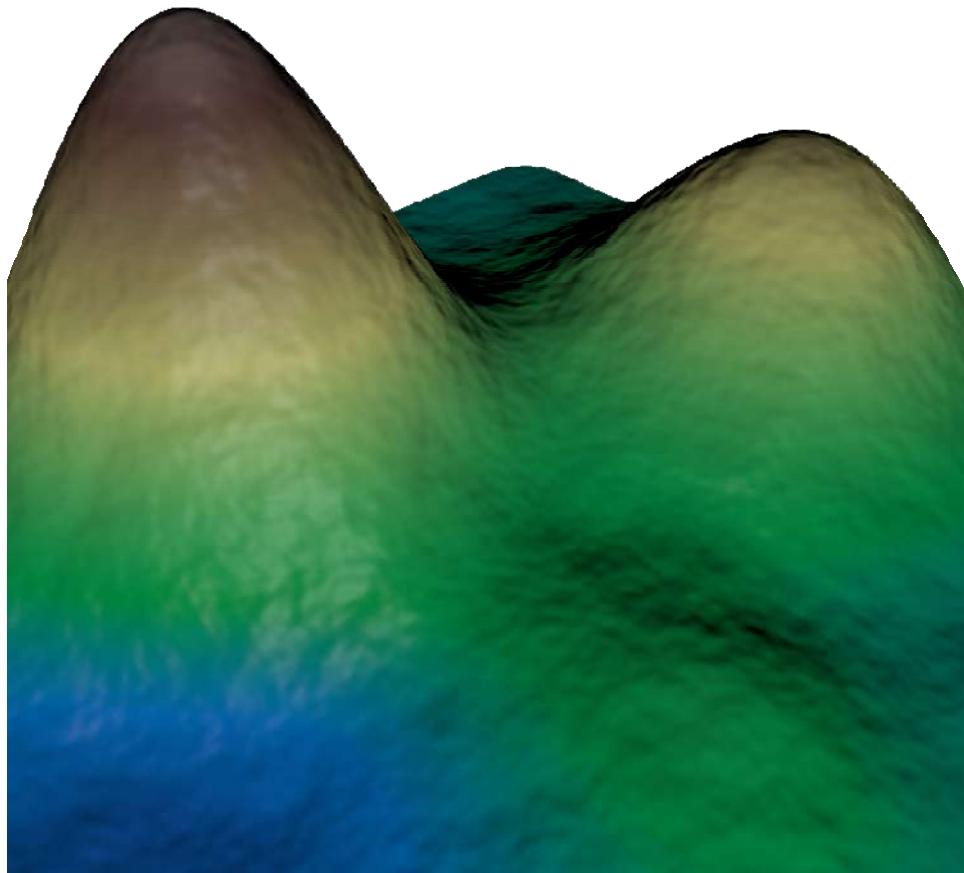
Textures 1D



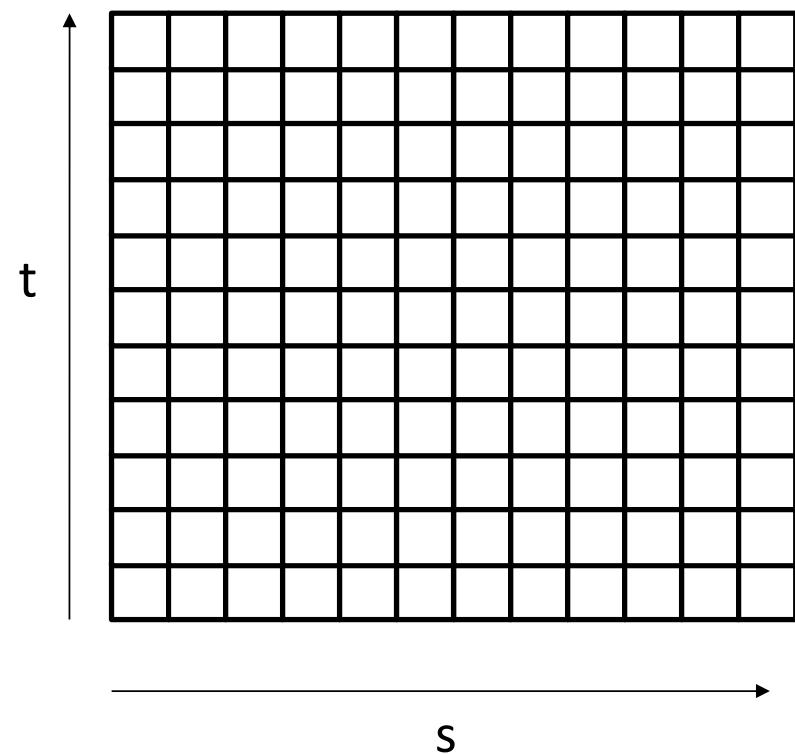
s



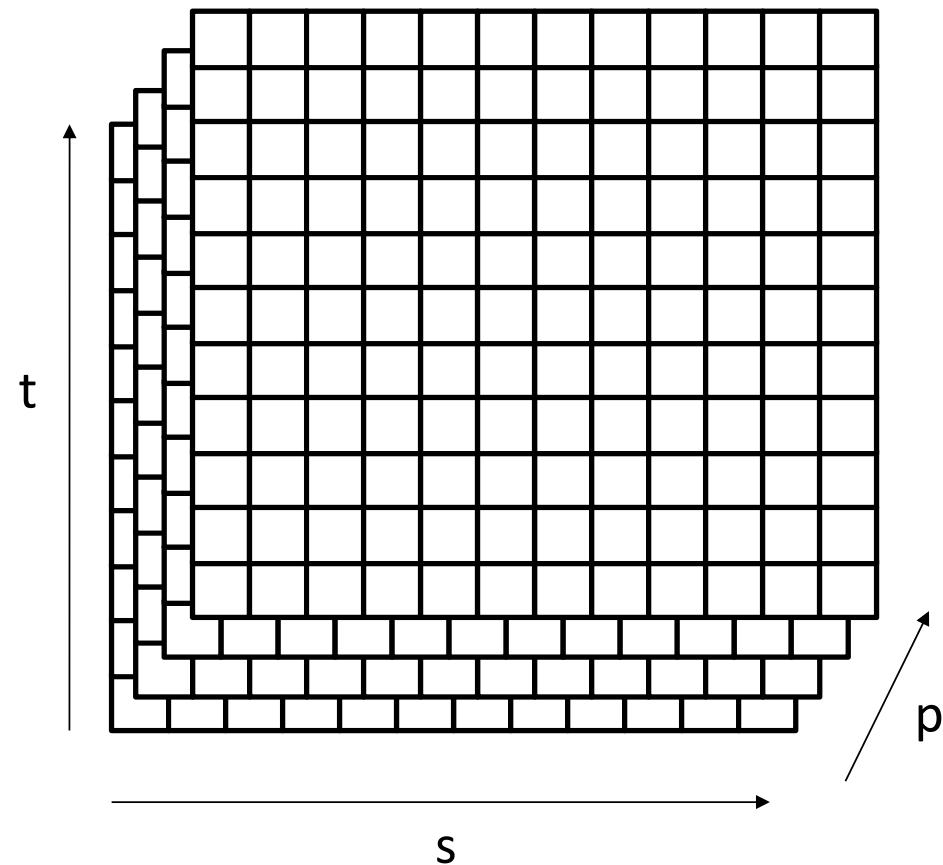
Textures 1D



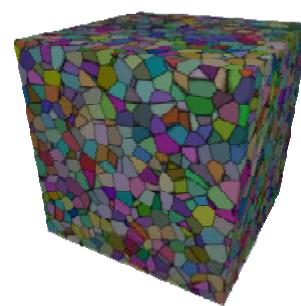
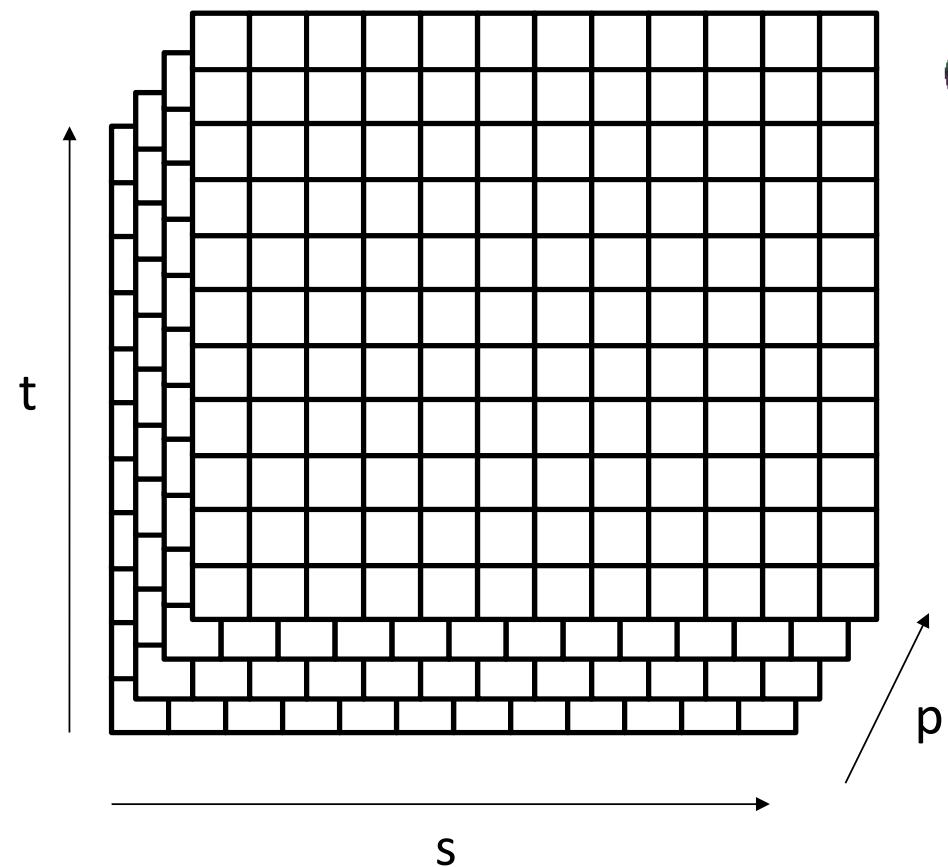
Textures 2D



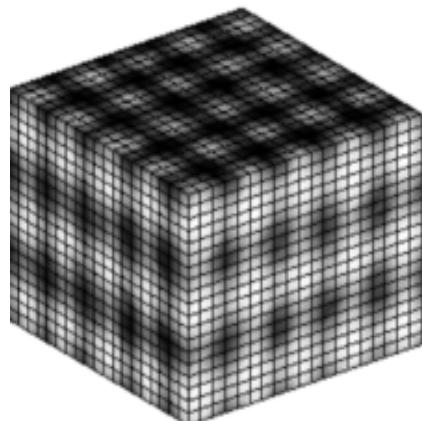
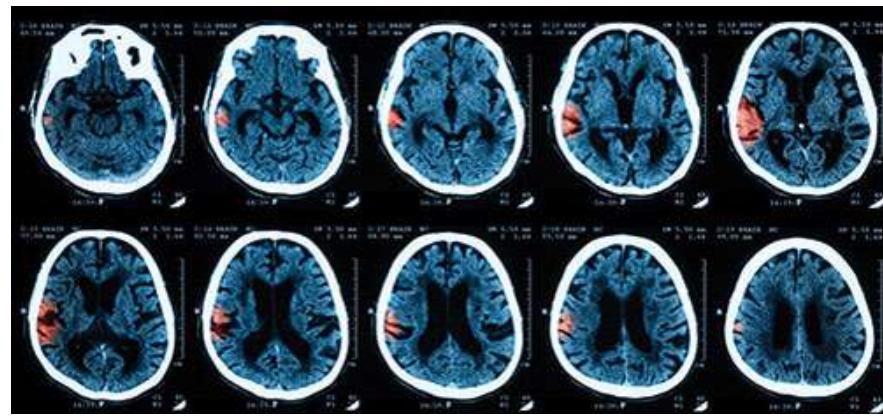
Textures 3D



Textures 3D



Textures 3D

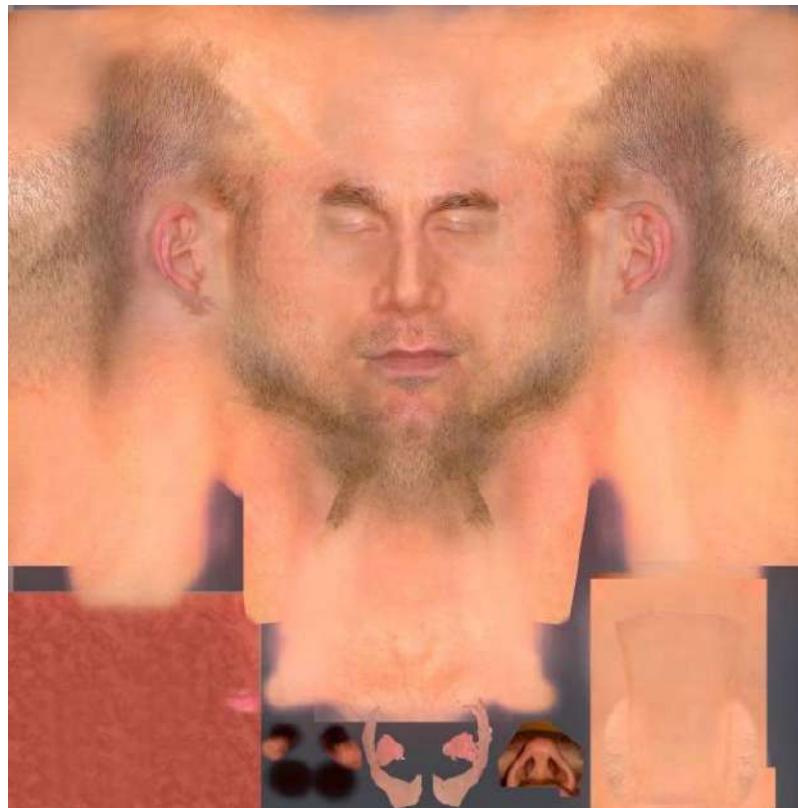


Textures

Una textura és una taula de dimensions, on cada cel·la enmagatzema una certa **propietat amb canals.**

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

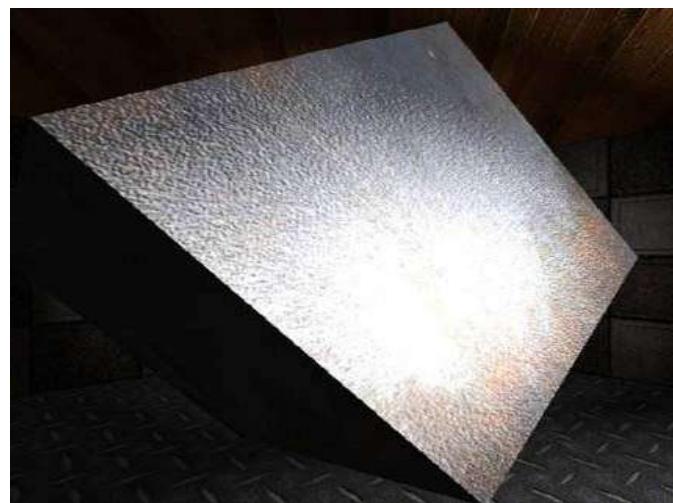
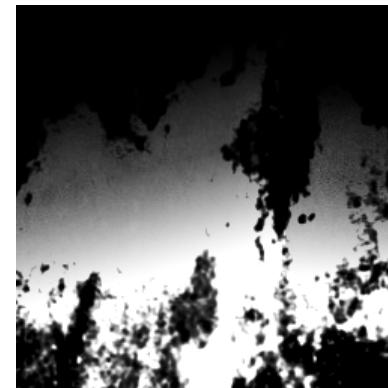
K_d (color map)



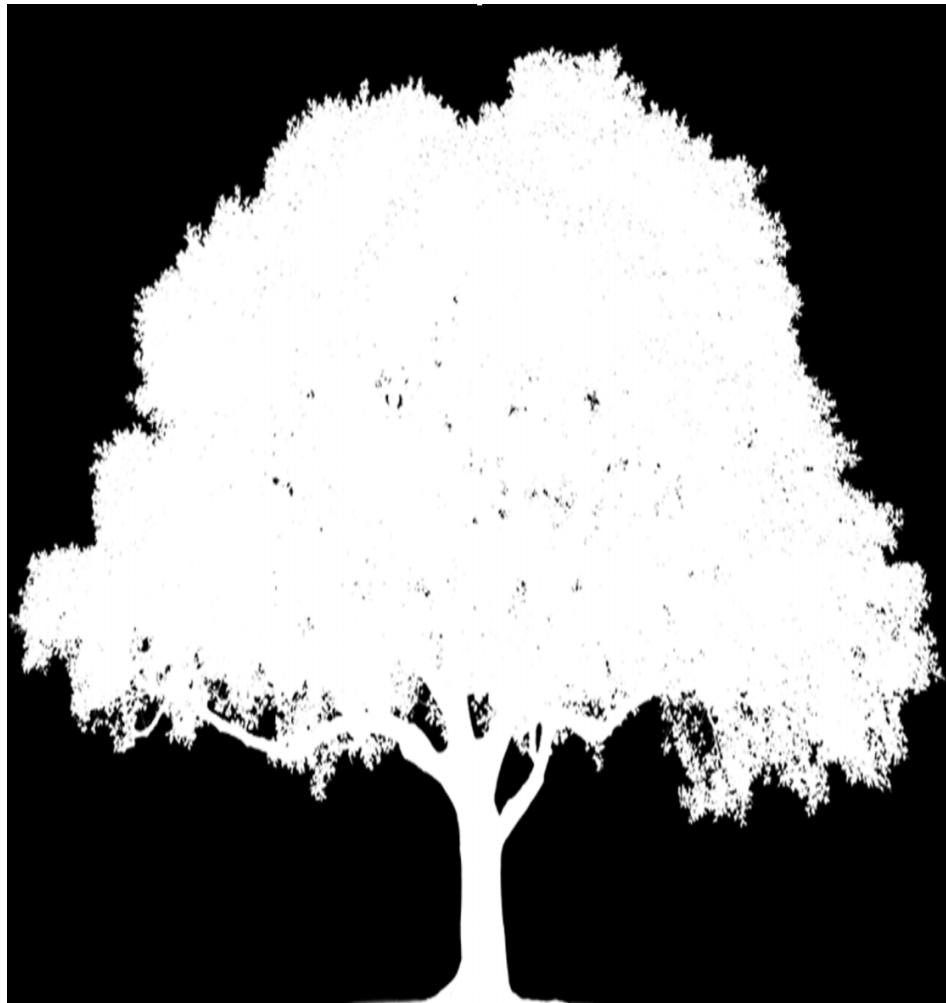
K_s (gloss map)



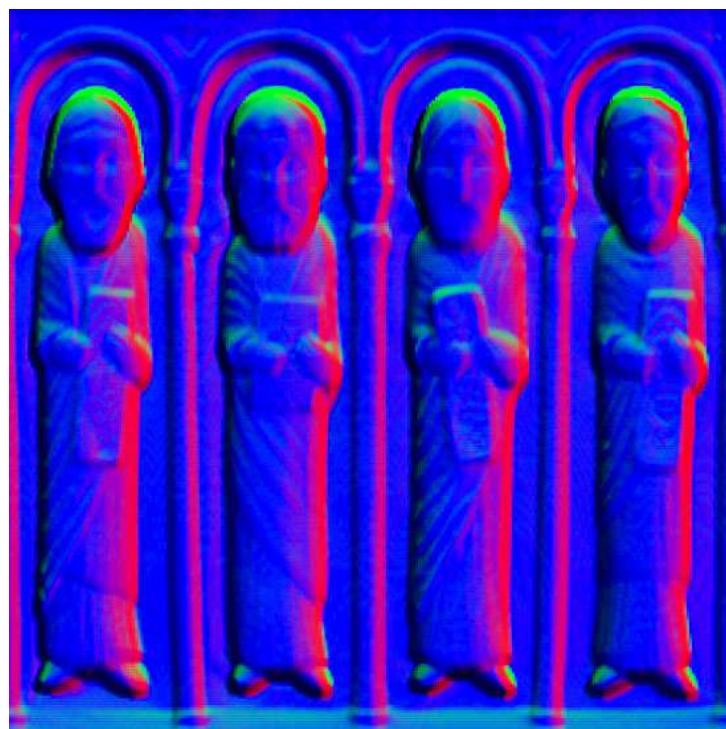
K_s (gloss map)



Opacitat (opacity map, alpha mask)



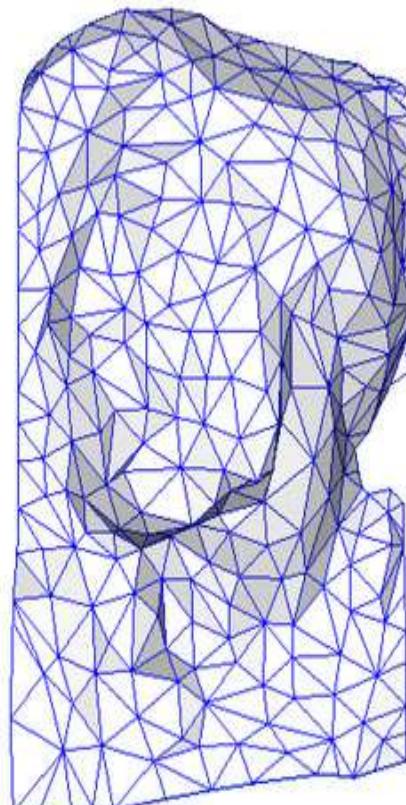
Normal (normal map)



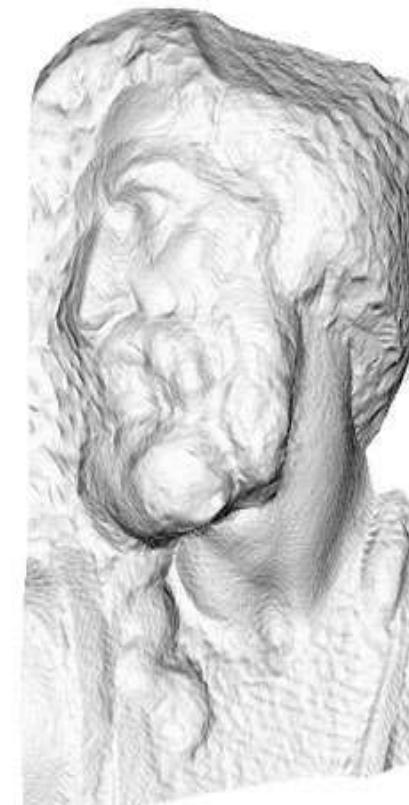
Normal (normal mapping)



original mesh
4M triangles

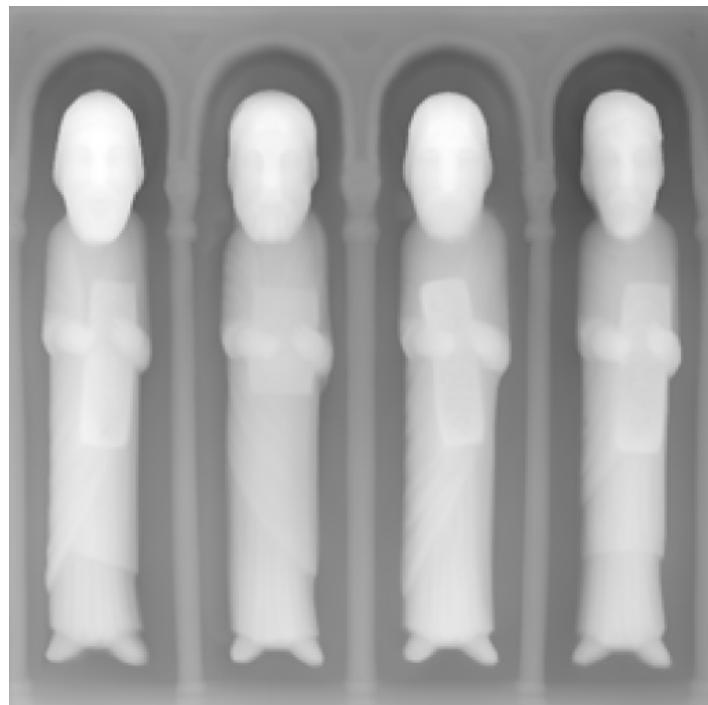


simplified mesh
500 triangles

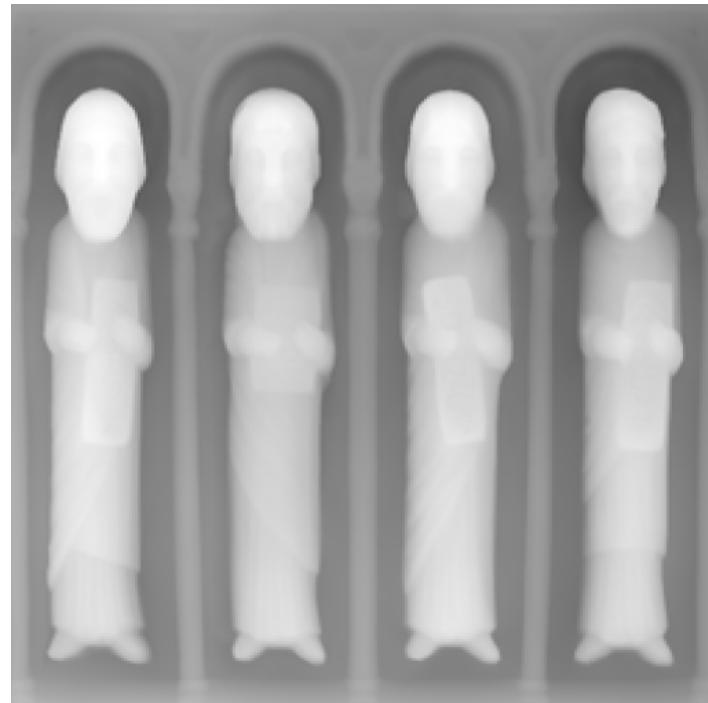


simplified mesh
and normal mapping
500 triangles

Desplaçament (bump mapping)

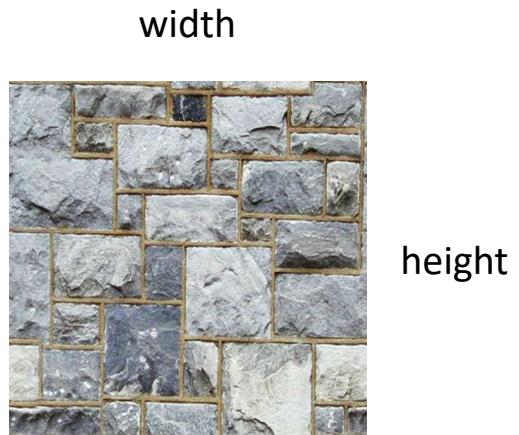


Desplaçament (displacement mapping)

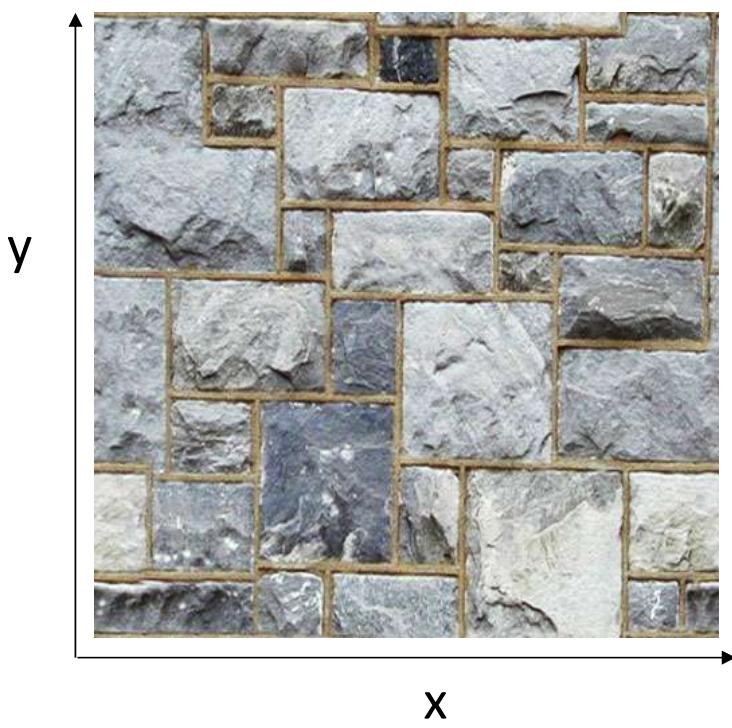


Mida d'una textura

- # texels en cada dimensió
- Habitualment w, h són potència de 2.

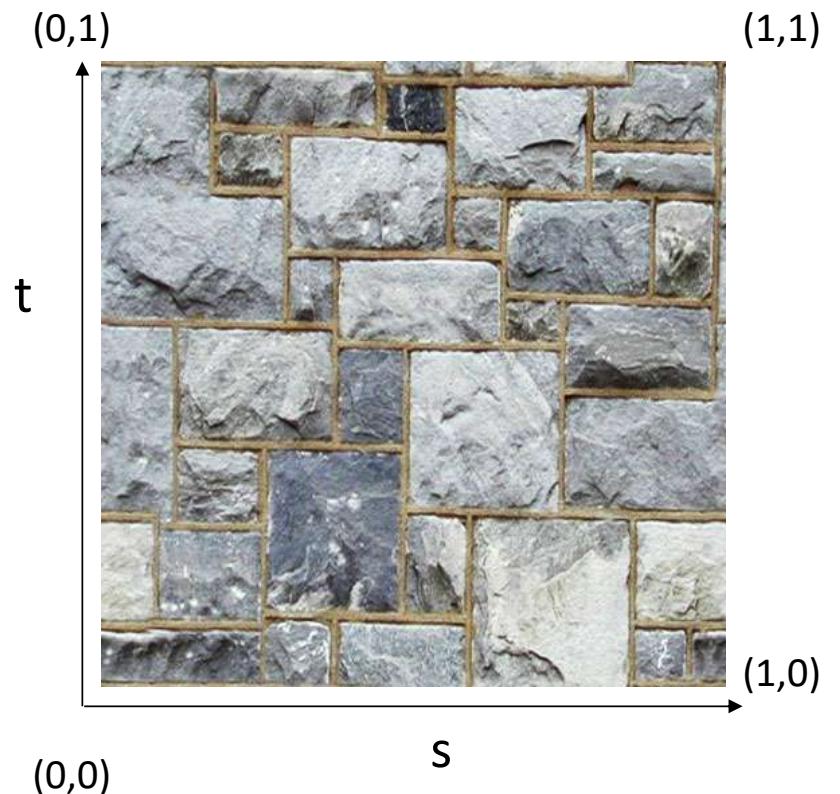


Espai normalitzat de textura



$$x \in [0, \text{width}]$$

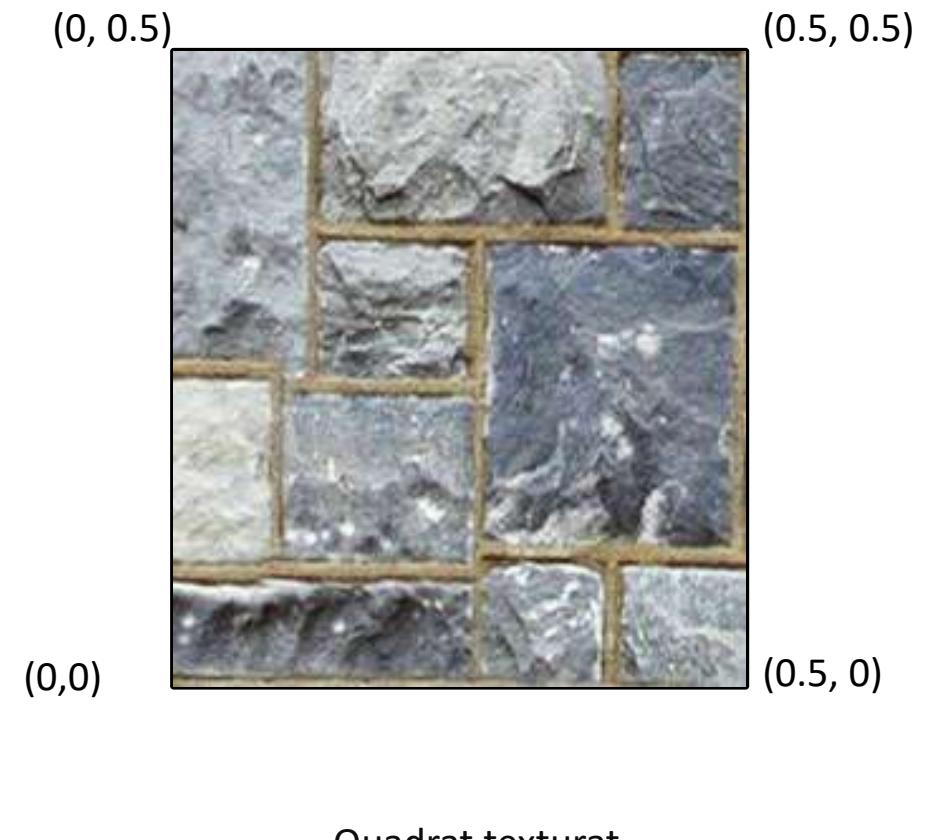
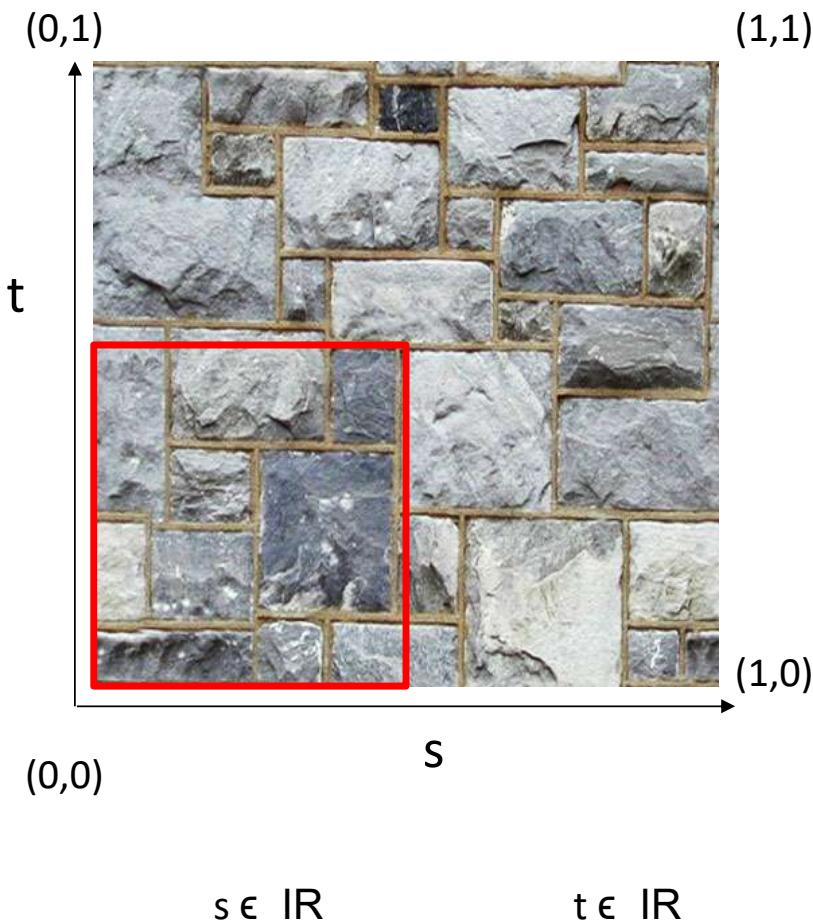
$$y \in [0, \text{height}]$$



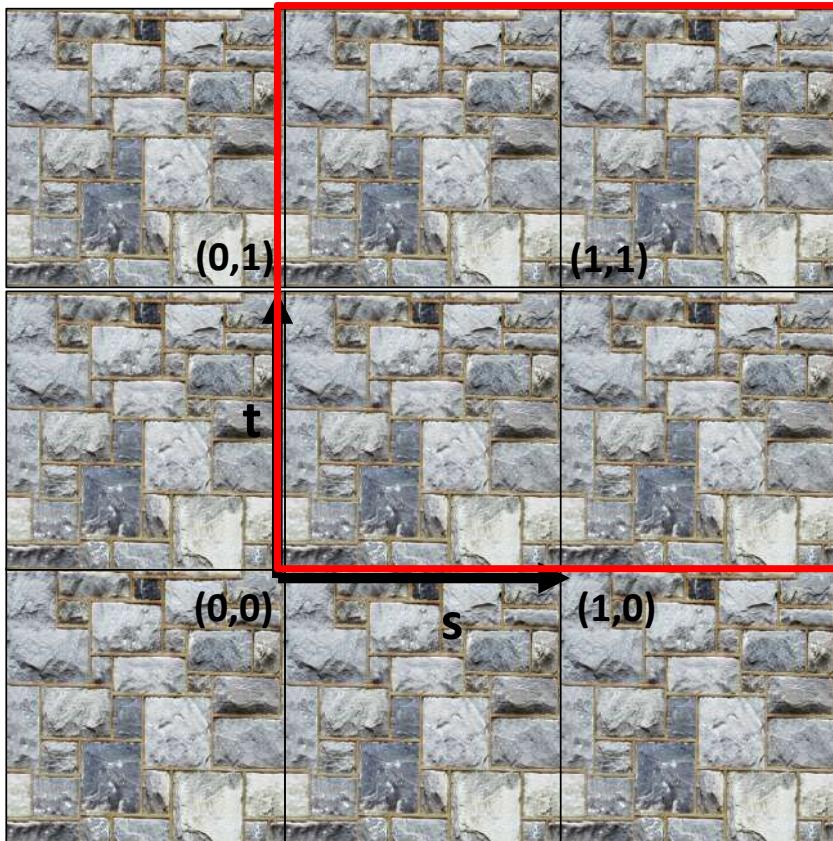
$$s \in \mathbb{R}$$

$$t \in \mathbb{R}$$

Espai normalitzat de textura



Espai normalitzat de textura



$(0, 2)$



$(2, 2)$



$(0, 0)$

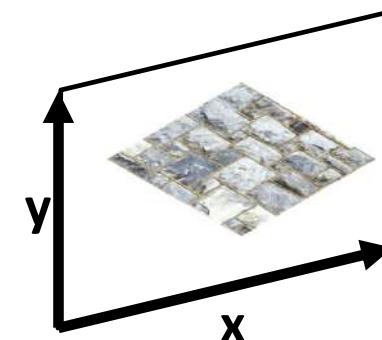
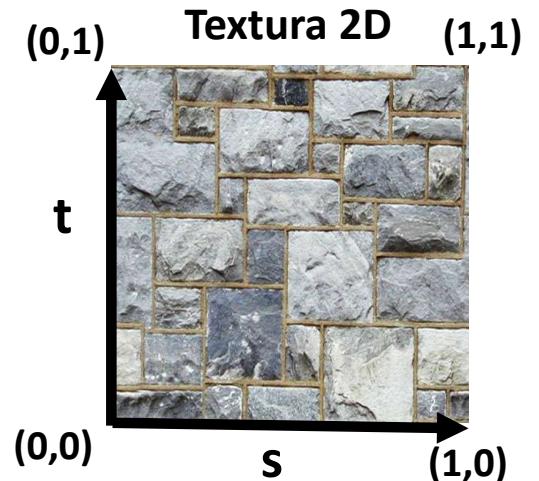
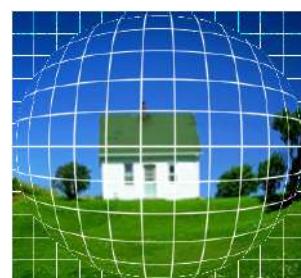
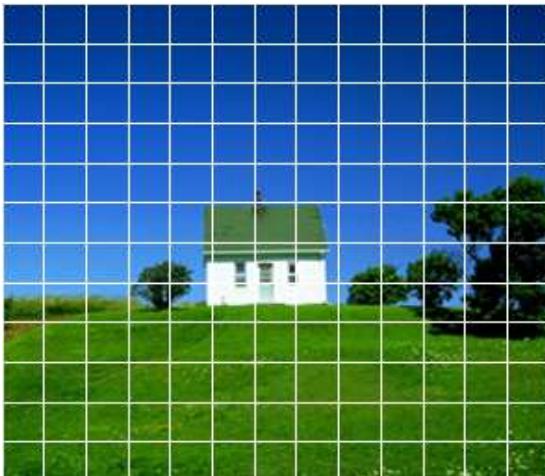
$(2, 0)$

Quadrat texturat

MAPPING

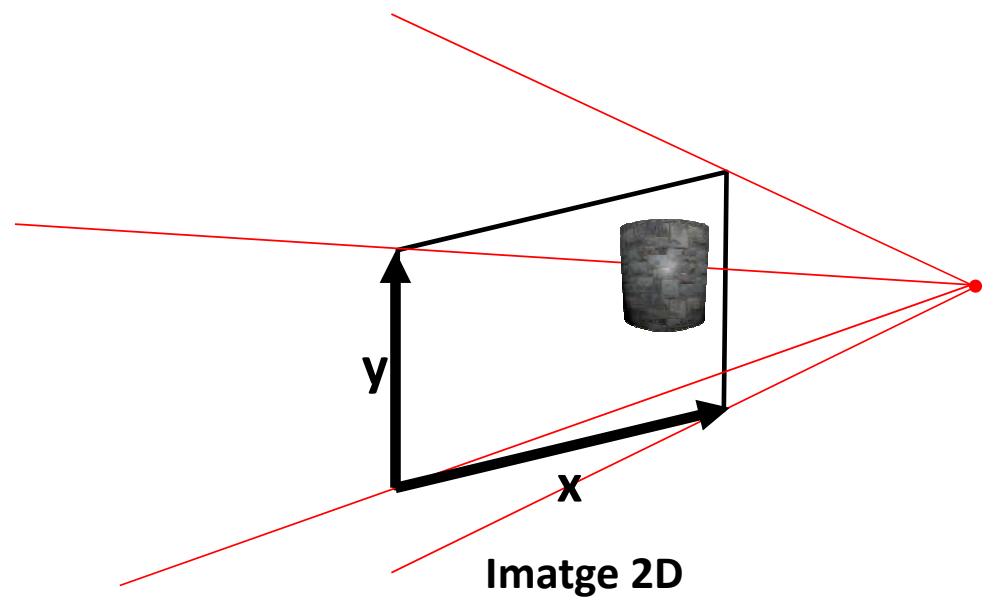
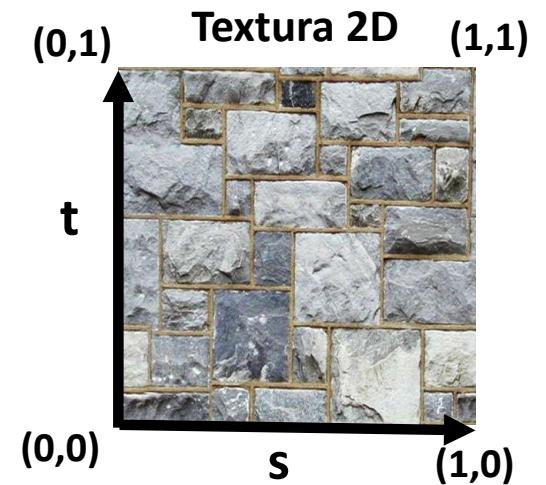


Forward / Inverse mapping



Imatge 2D

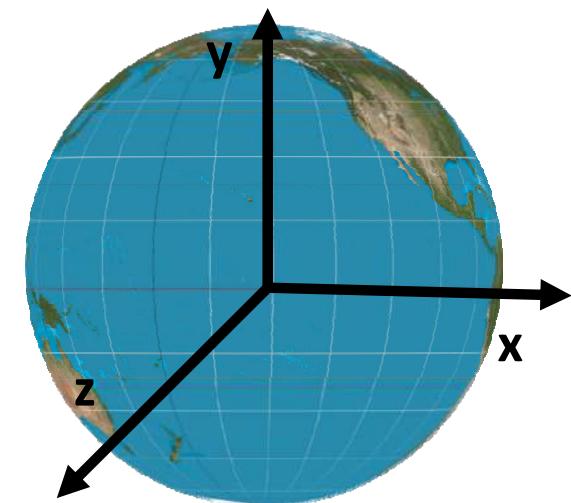
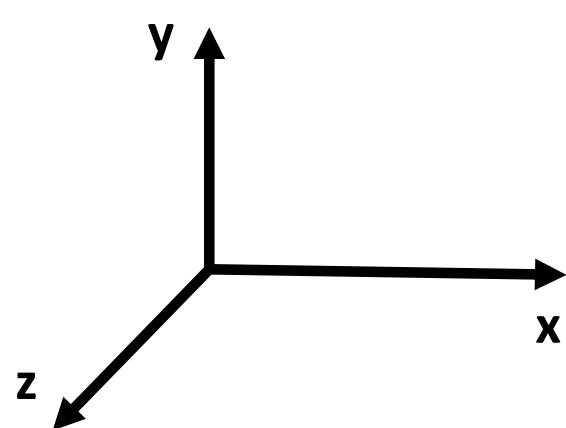
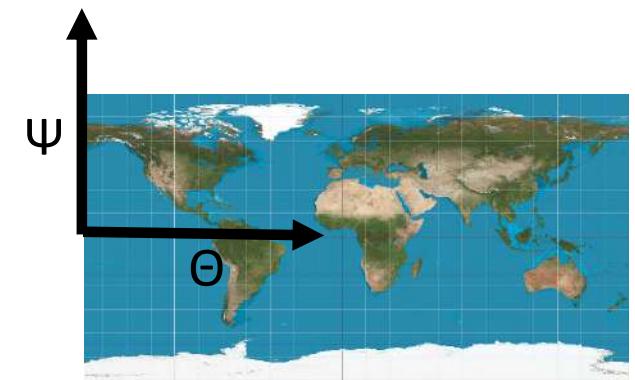
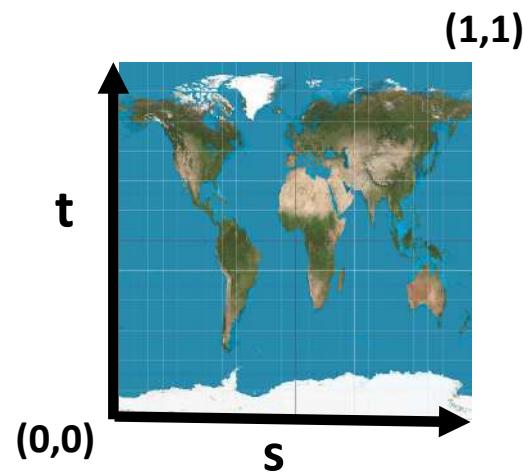
Forward / Inverse mapping



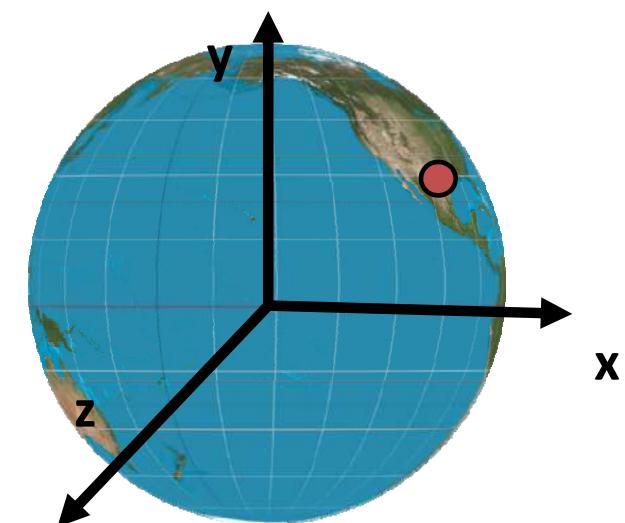
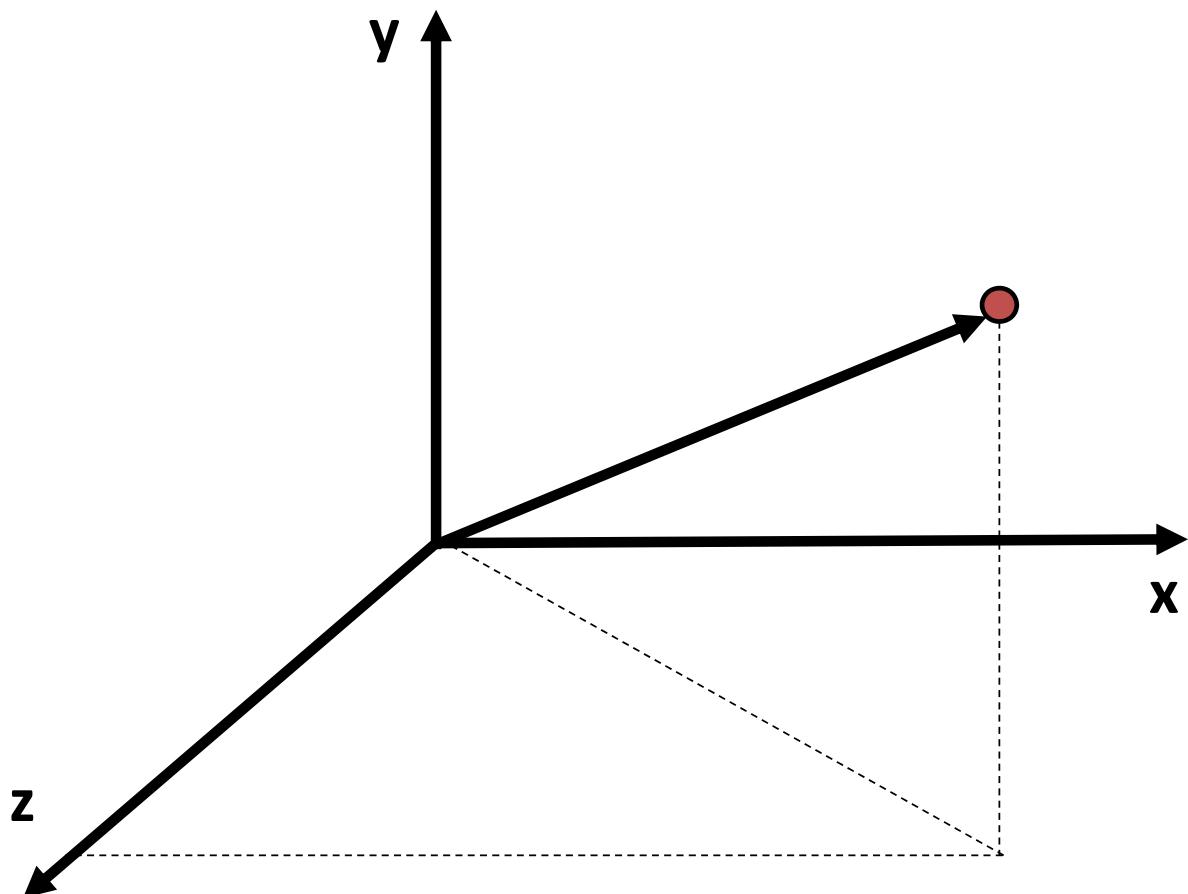
Exemple 1: Mapping esfèric

Input: $s \in [0,1]$, $t \in [0,1]$

Output: $x,y,z \in$ esfera unitat



Exemple 1: Mapping esfèric



Exemple 1: Mapping esfèric

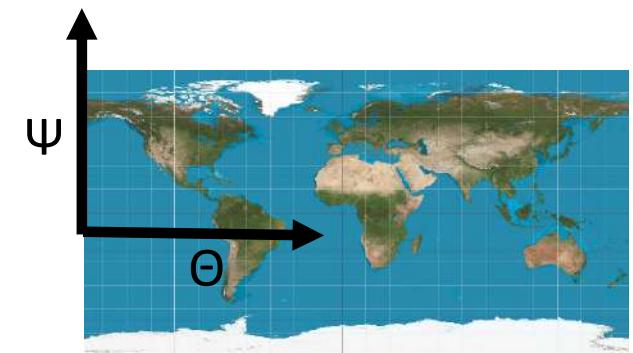
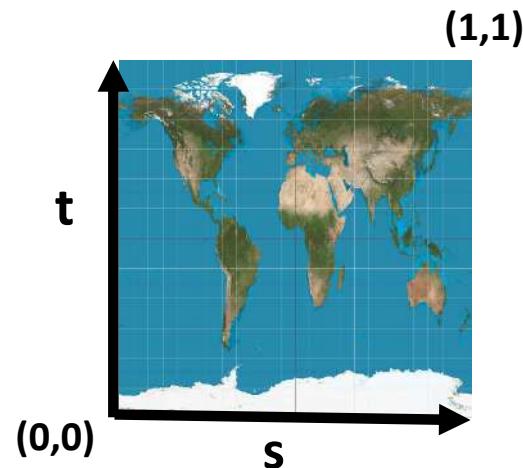
Input: $s \in [0,1]$, $t \in [0,1]$

Output: $x,y,z \in$ esfera unitat

// pas $(s, t) \rightarrow (\Theta, \Psi)$

$\Theta = 2\pi s;$

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

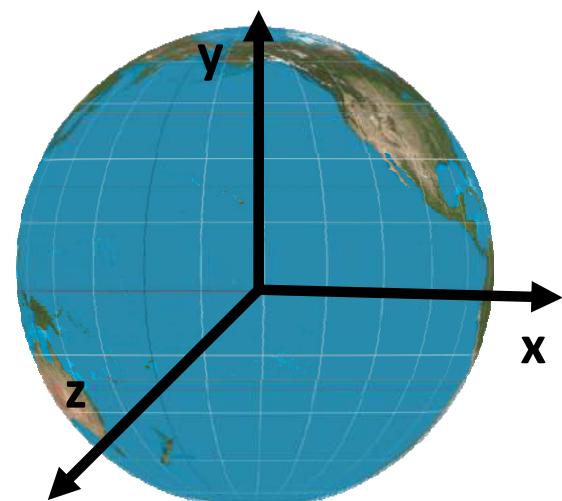


// pas esfèriques $\rightarrow (x,y,z)$

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

$y = \sin(\Psi);$

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



Exemple 2: Mapping cilíndric

Input: $s \in [0,1]$, $t \in [0,1]$

Output: $x,y,z \in$ cilindre $r=1$ sobre pla XZ

// pas $(s, t) \rightarrow (\Theta, h)$

$\Theta = 2\pi s;$

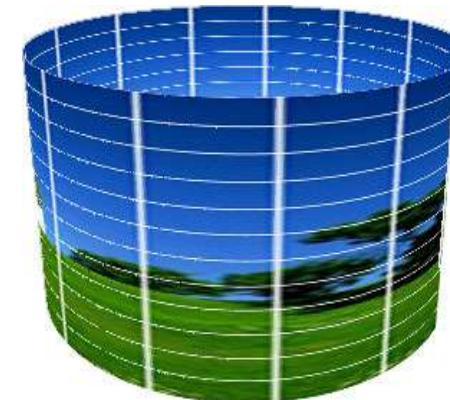
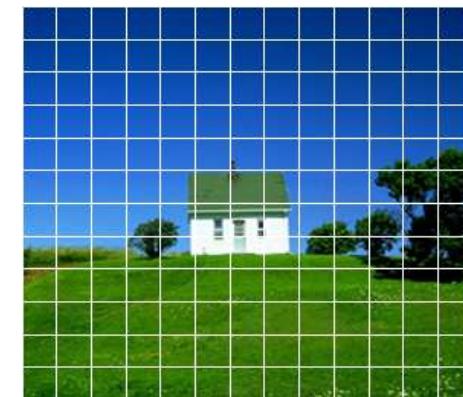
$h = t;$

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

$x = \sin(\Theta);$

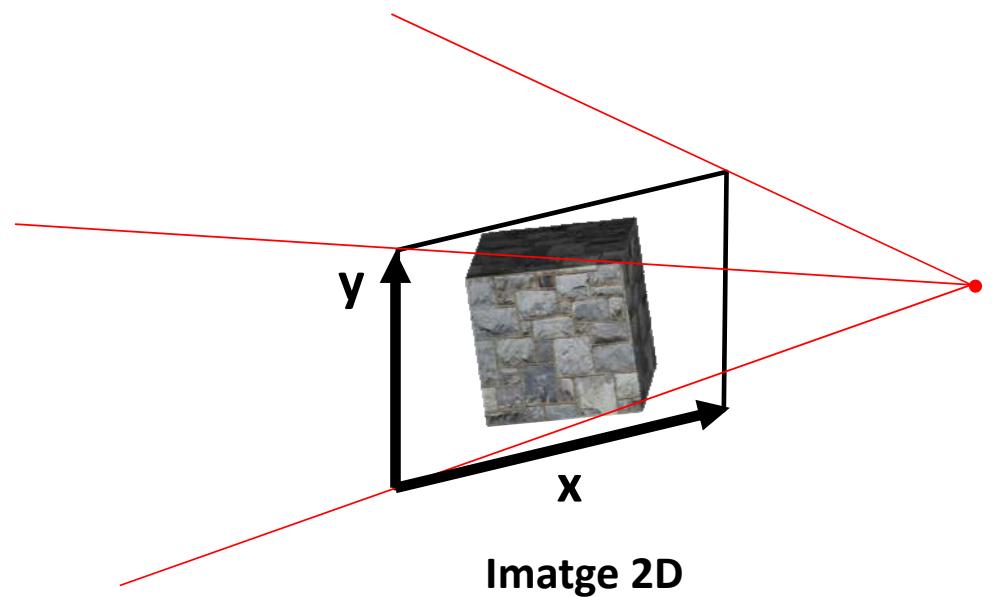
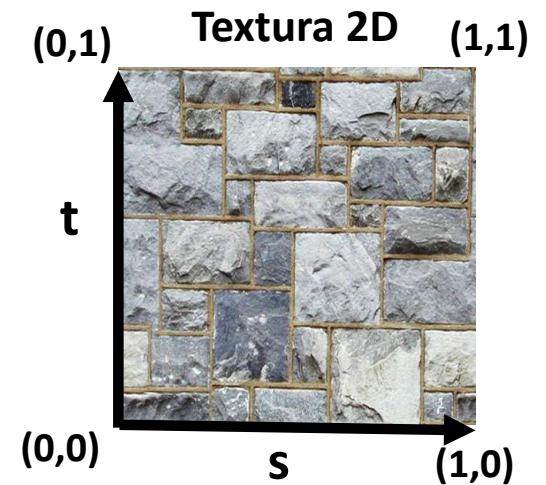
$y = h;$

$z = \cos(\Theta);$

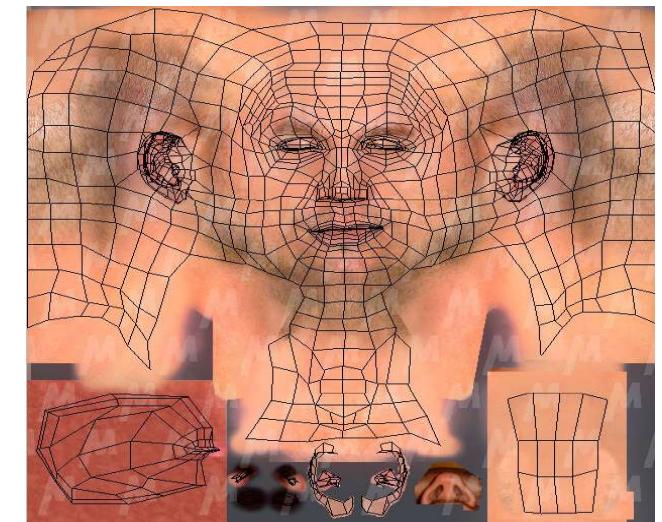
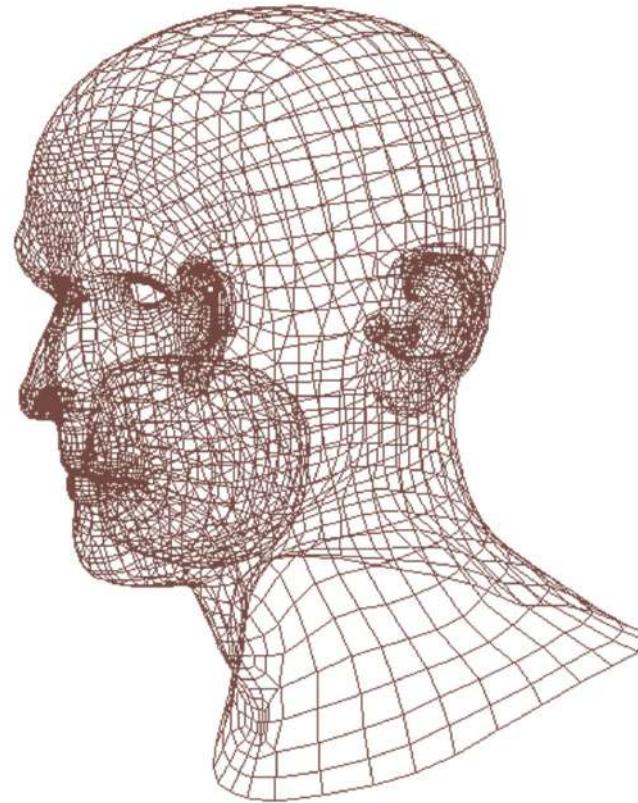
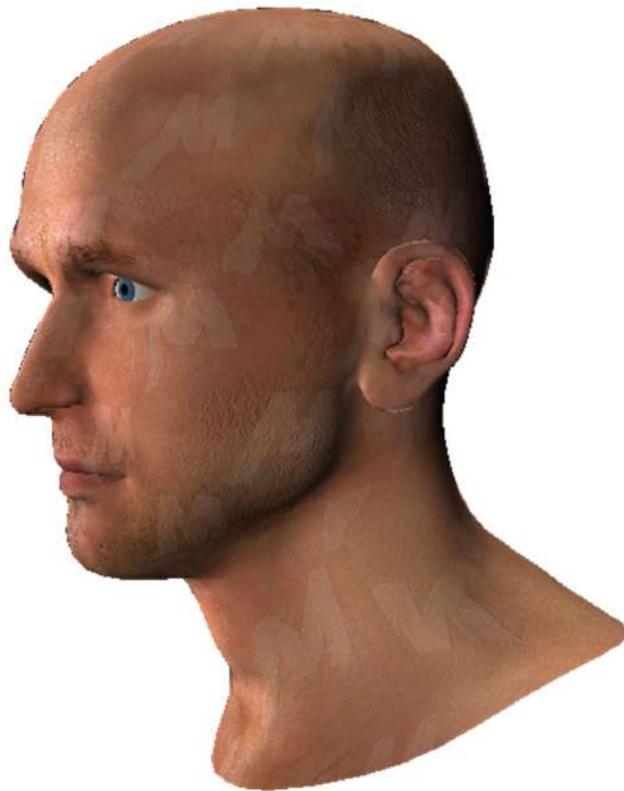


MAPPING EN OPENGL

En gràfics, habitualment



En gràfics, habitualment

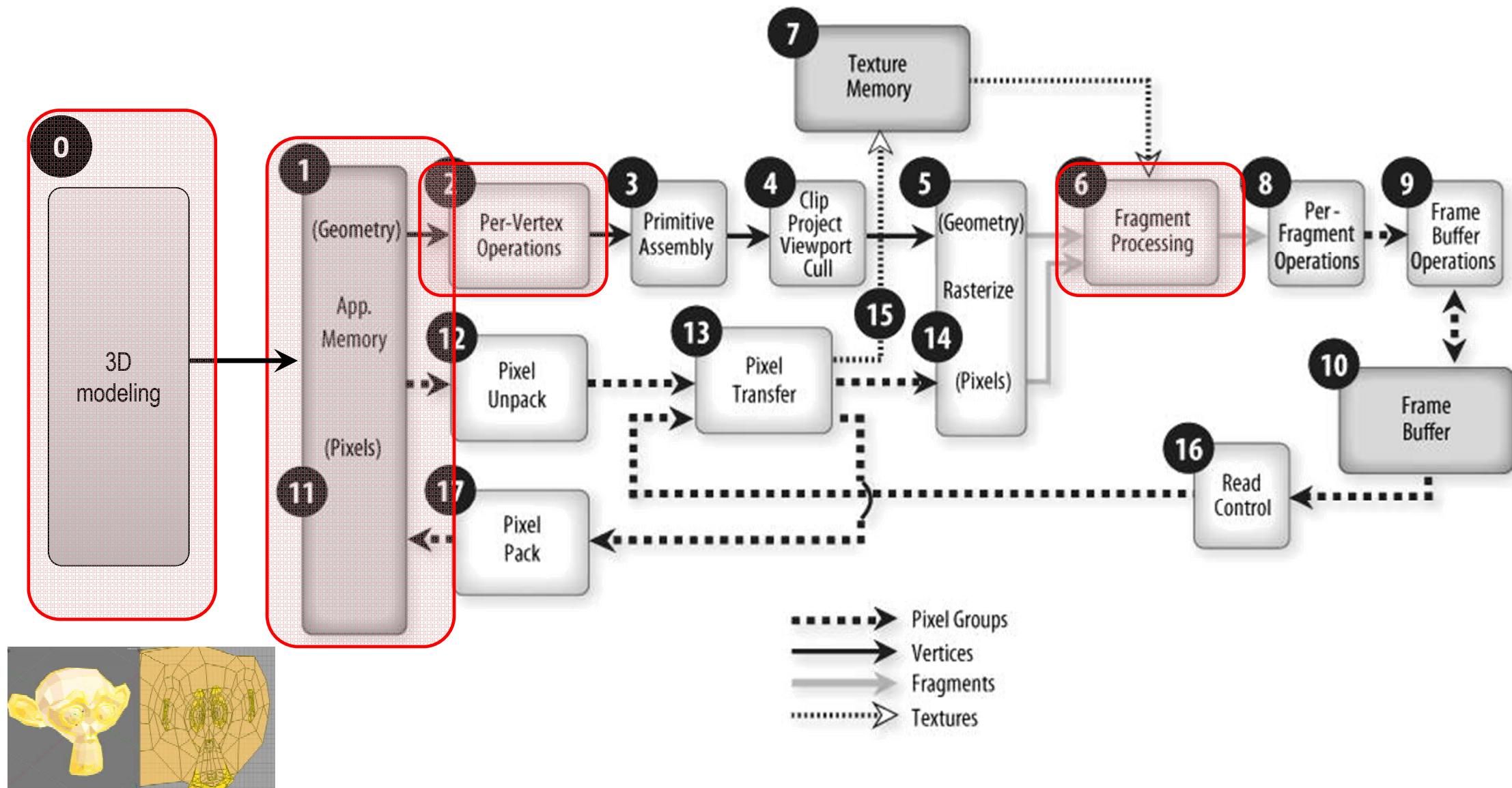


DEMO (TEXTURE MAPPING AMB BLENDER)

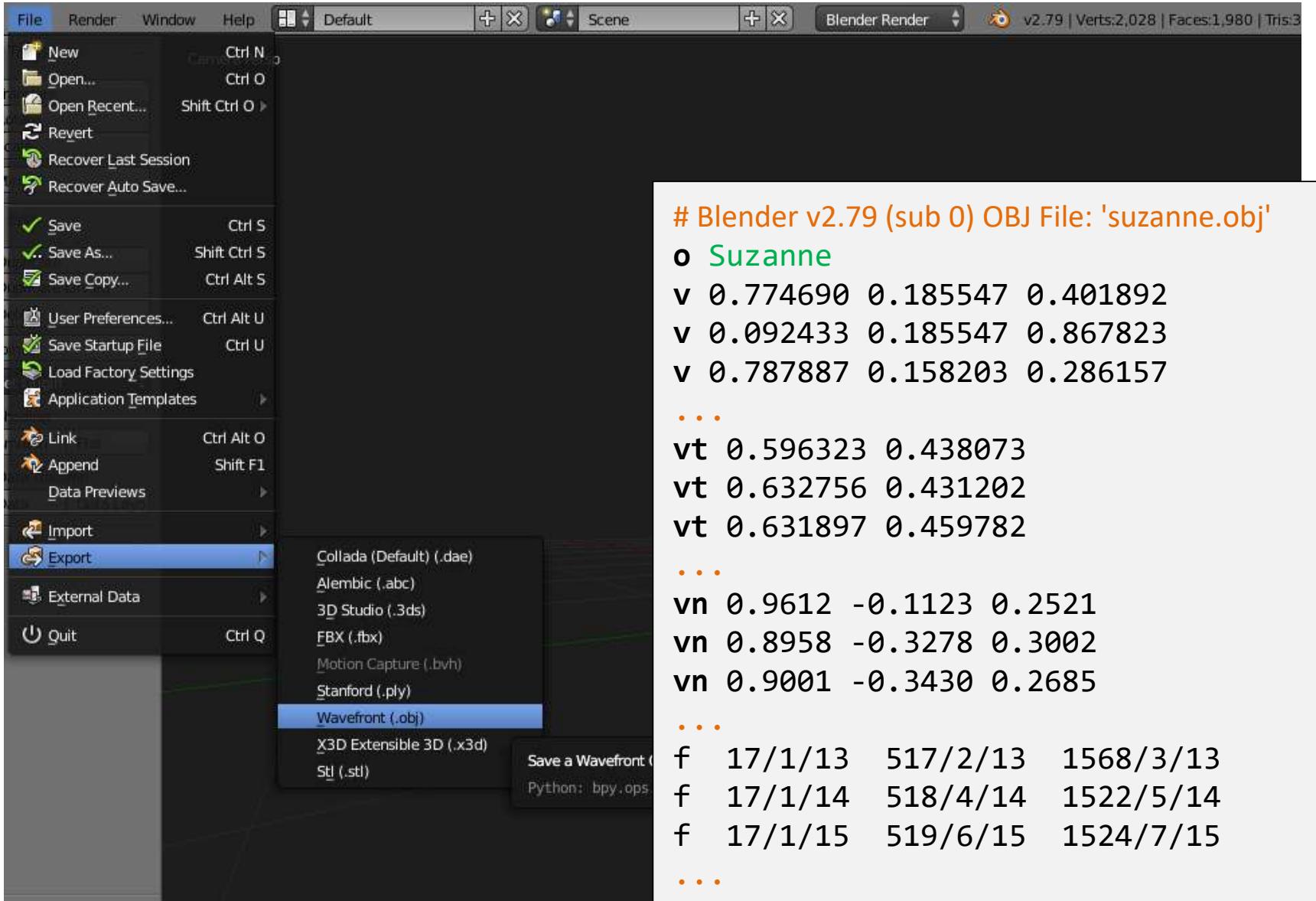
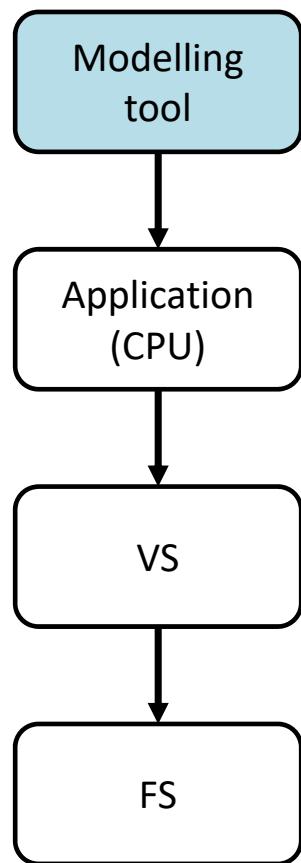
DEMO (TEXTURE MAPPING AMB OPENGL)

En quina etapa es generen les coordenades de textura?

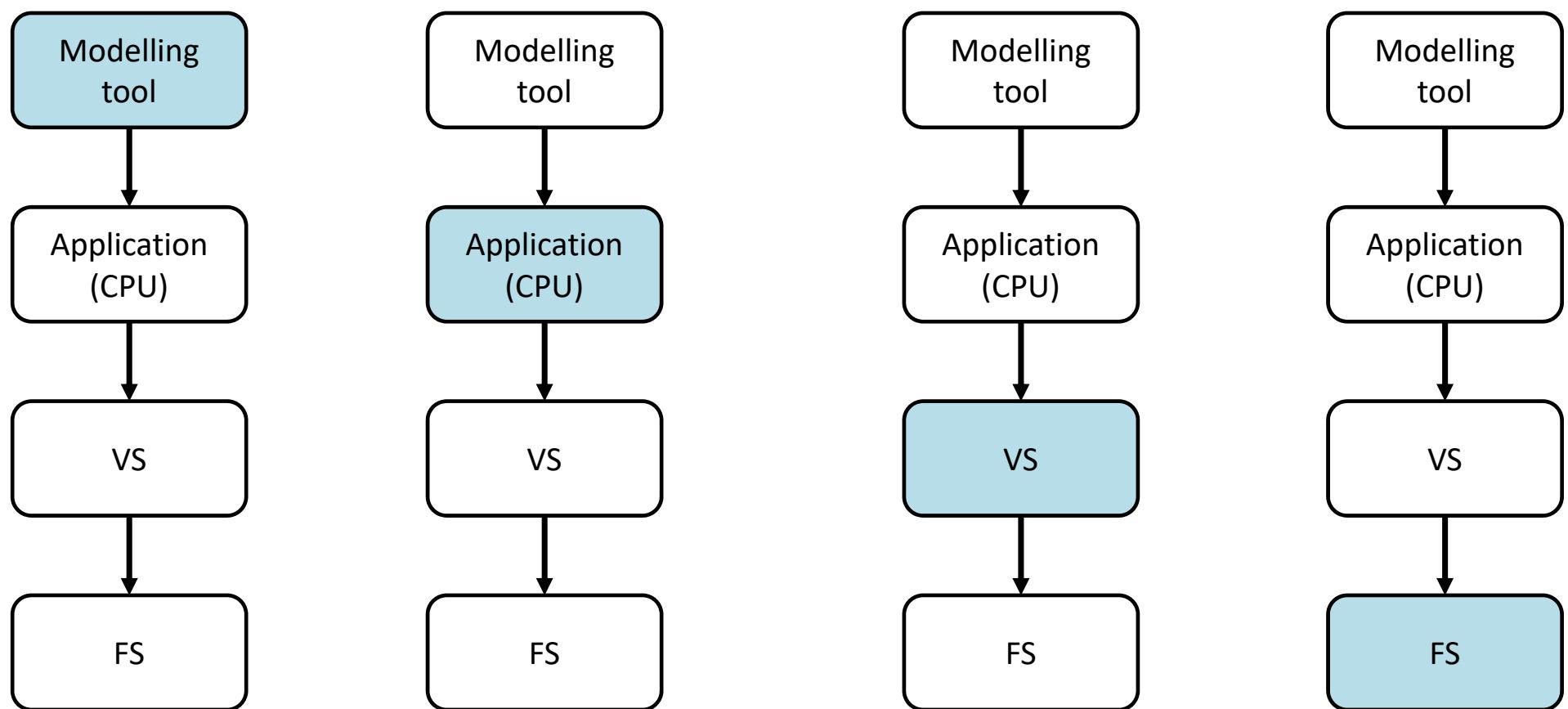
COORDENADES DE TEXTURA AL PIPELINE



Options



Opcions per a generar coords de textura



MÈTODES PER GENERAR COORDS DE TEXTURA

Generació de coordenades de textura

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

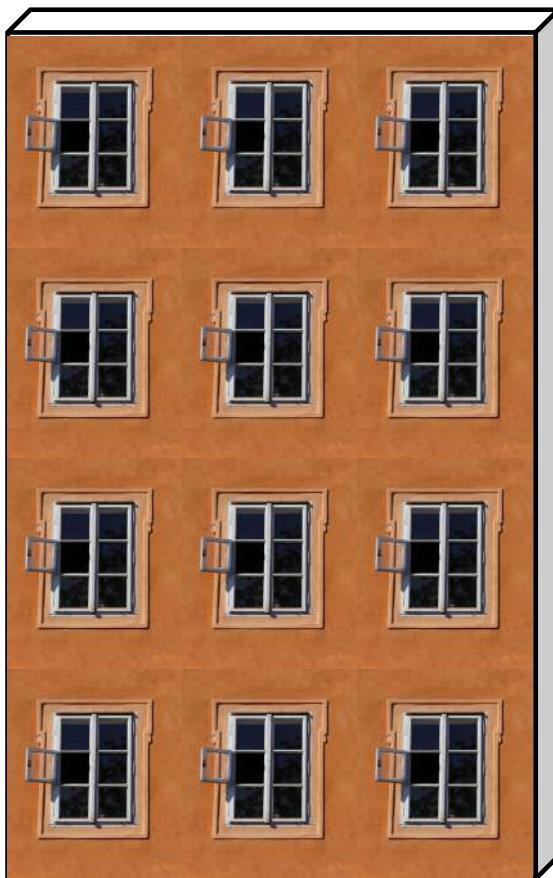
Plans S,T

Paràmetres:

- plans S,T

Càlcul s, t:

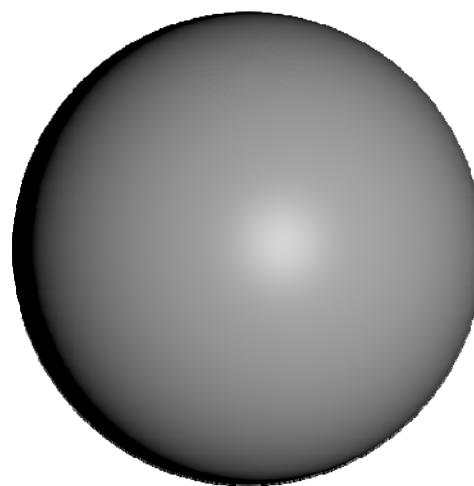
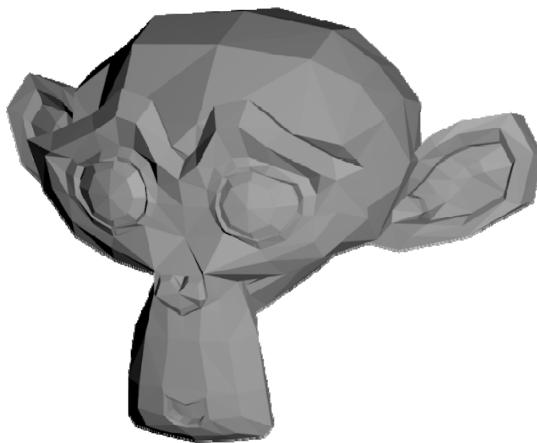
Plans S,T (exemple)



Generació de coordenades de textura

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

Parametrització amb superfície auxiliar



H1	H2	H3	H4	H5	H6	H7	H8
G1	G2	G3	G4	G5	G6	G7	G8
F1	F2	F3	F4	F5	F6	F7	F8
E1	E2	E3	E4	E5	E6	E7	E8
D1	D2	D3	D4	D5	D6	D7	D8
C1	C2	C3	C4	C5	C6	C7	C8
B1	B2	B3	B4	B5	B6	B7	B8
A1	A2	A3	A4	A5	A6	A7	A8

Exemples: S^{-1} mappings

Input: $s \in [0,1]$, $t \in [0,1]$

Output: $x,y,z \in$ esfera unitat

```
// pas (s, t) → (θ, ψ)
```

```
θ = 2πs;
```

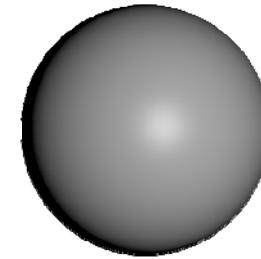
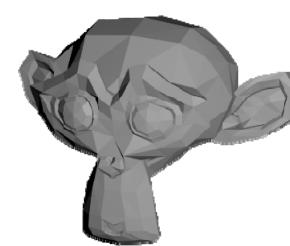
```
ψ = π(t-0.5);
```

```
// pas esfèriques → (x,y,z)
```

```
x = sin(θ)cos(ψ);
```

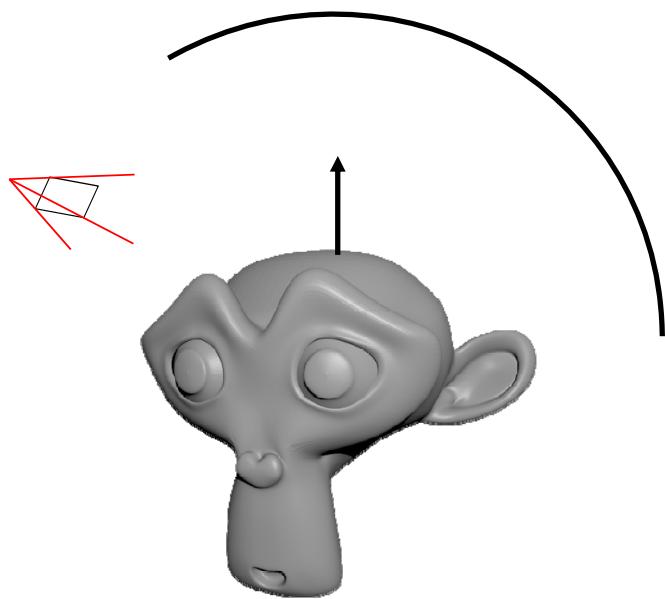
```
y = sin(ψ);
```

```
z = cos(θ)cos(ψ);
```

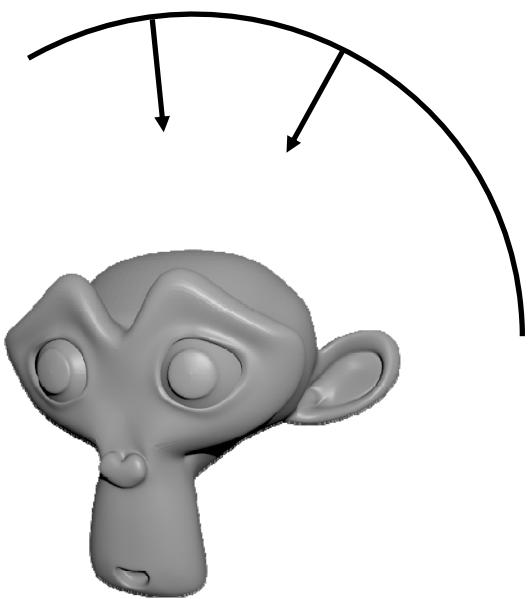


H1	H2	H3	H4	H5	H6	H7	H8
G1	G2	G3	G4	G5	G6	G7	G8
F1	F2	F3	F4	F5	F6	F7	F8
E1	E2	E3	E4	E5	E6	E7	E8
D1	D2	D3	D4	D5	D6	D7	D8
C1	C2	C3	C4	C5	C6	C7	C8
B1	B2	B3	B4	B5	B6	B7	B8
A1	A2	A3	A4	A5	A6	A7	A8

Exemples: O mappings

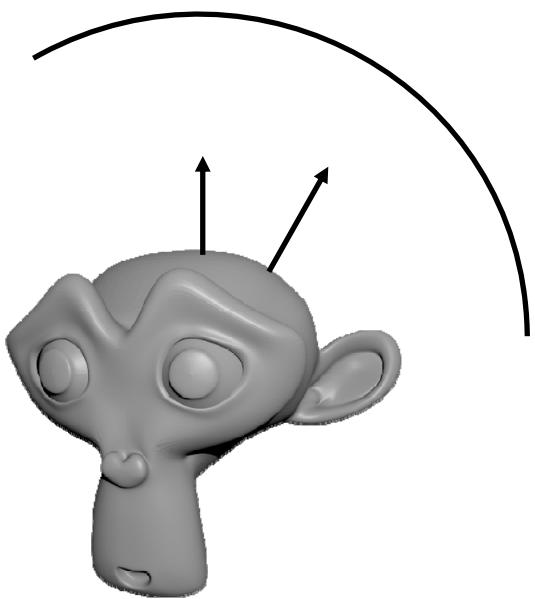


Reflected view ray

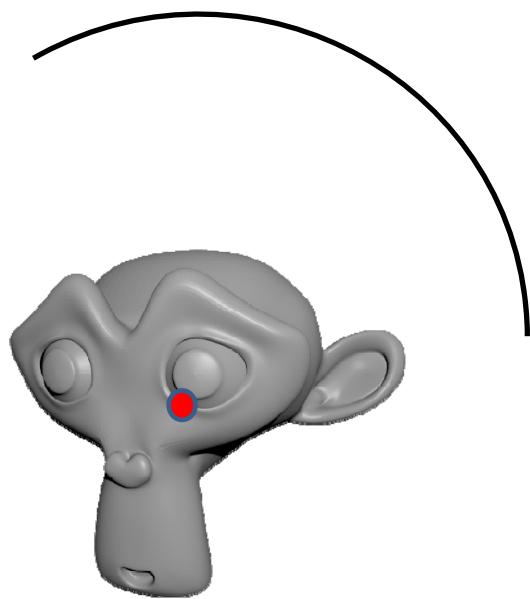


Intermediate surface normal

Exemples: O mappings

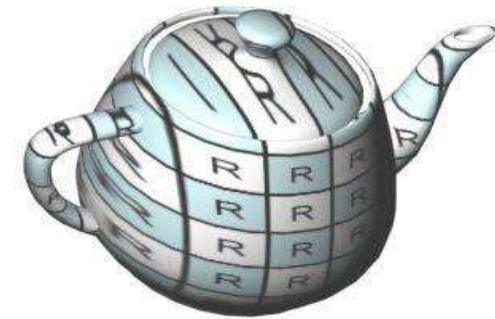
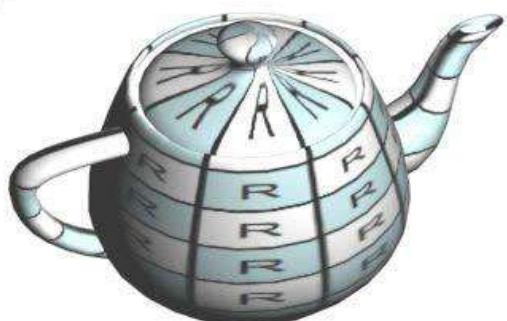
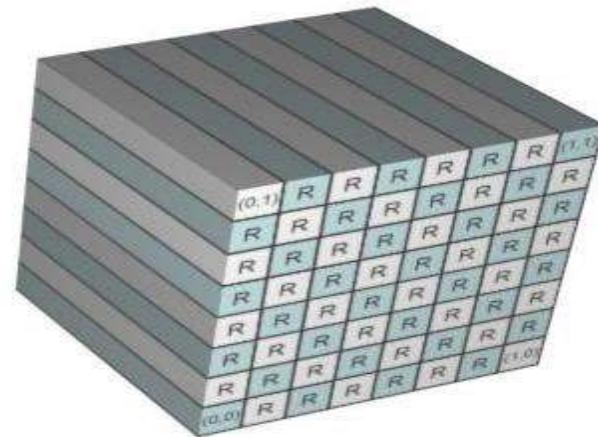
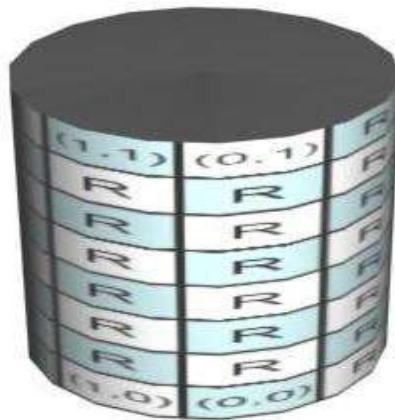


Object Normal



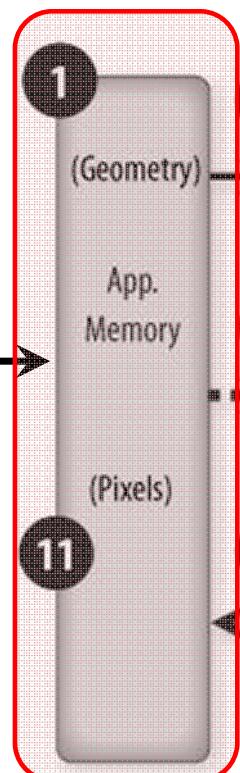
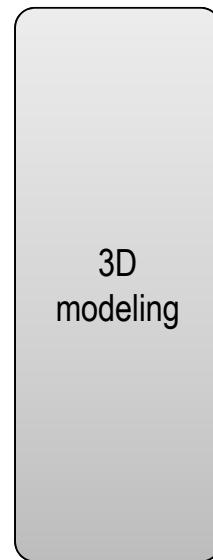
Object centroid

Projeccions esfèrica, cilíndrica i plana



CREACIÓ DE LA TEXTURA

0



11

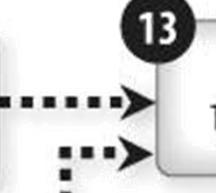


7

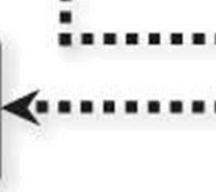
Texture Memory



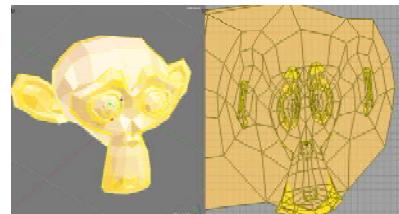
Frame Buffer



15

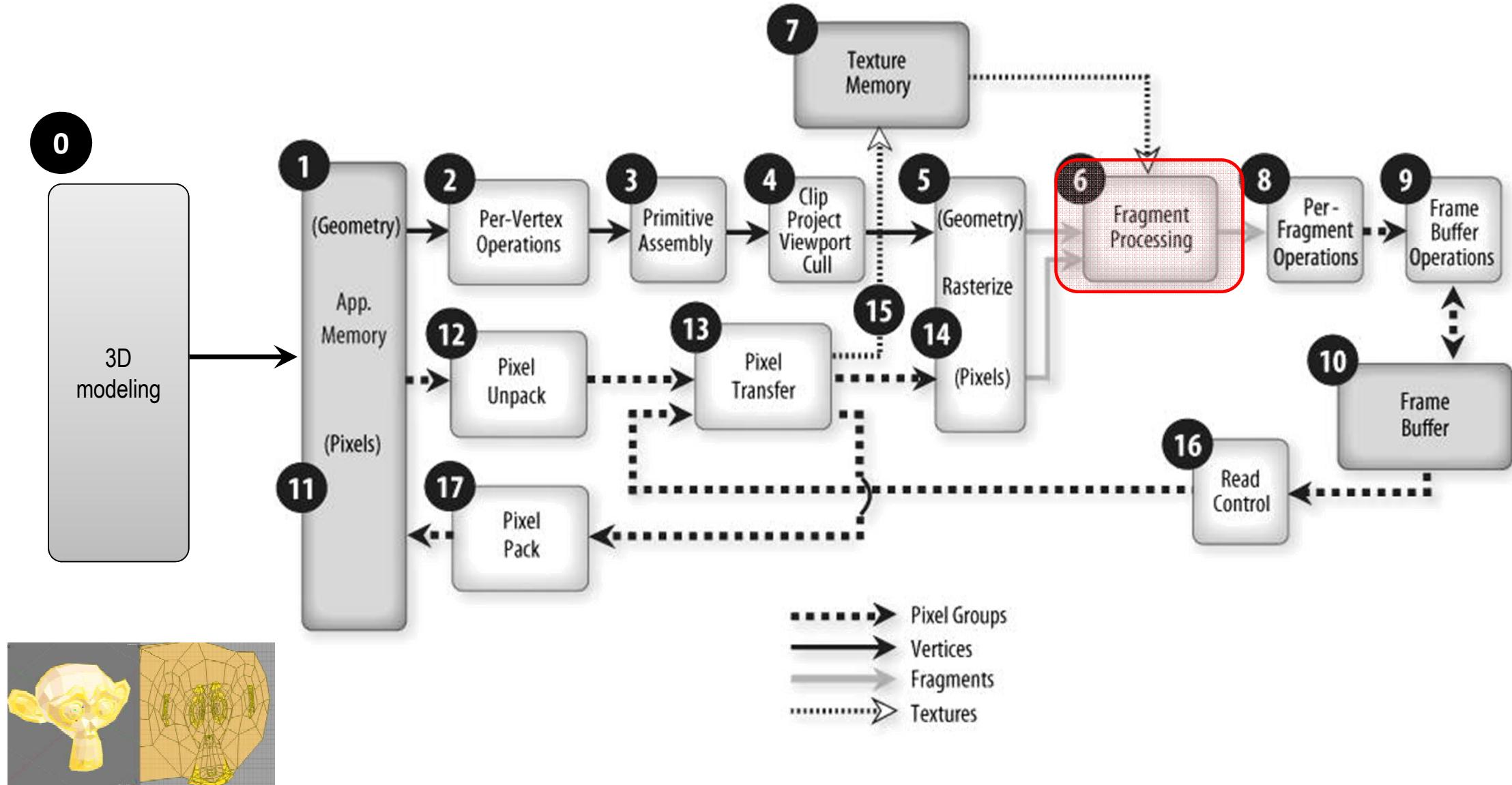


- > Pixel Groups
- > Vertices
- > Fragments
- > Textures



```
// 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);
...
```

ÚS DE TEXTURA AL FRAGMENT SHADER



FS – exemple

```
uniform sampler2D colorMap;  
in vec2 vtexcoord;
```

...

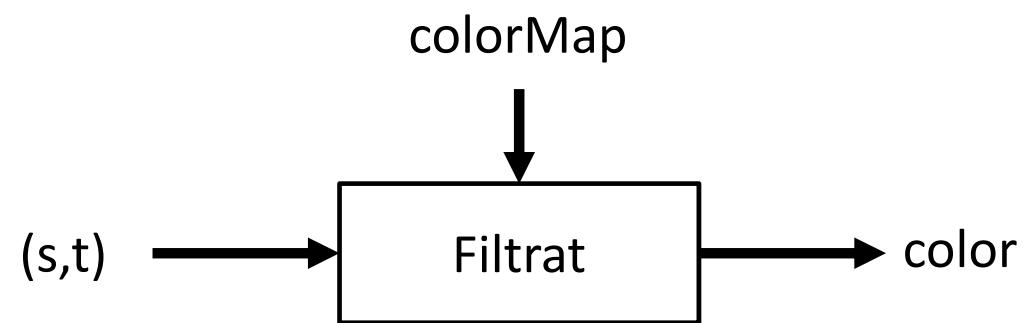
```
vec4 color = texture(colorMap, vtexcoord);
```

...

Magnification filters, Minification filters, Mipmapping

FILTRAT

Accés a textura



Necessitat del filtrat



Objeto texturado



Textura

Necessitat del filtrat



Textura



Magnification \approx upsampling



Minification \approx downsampling

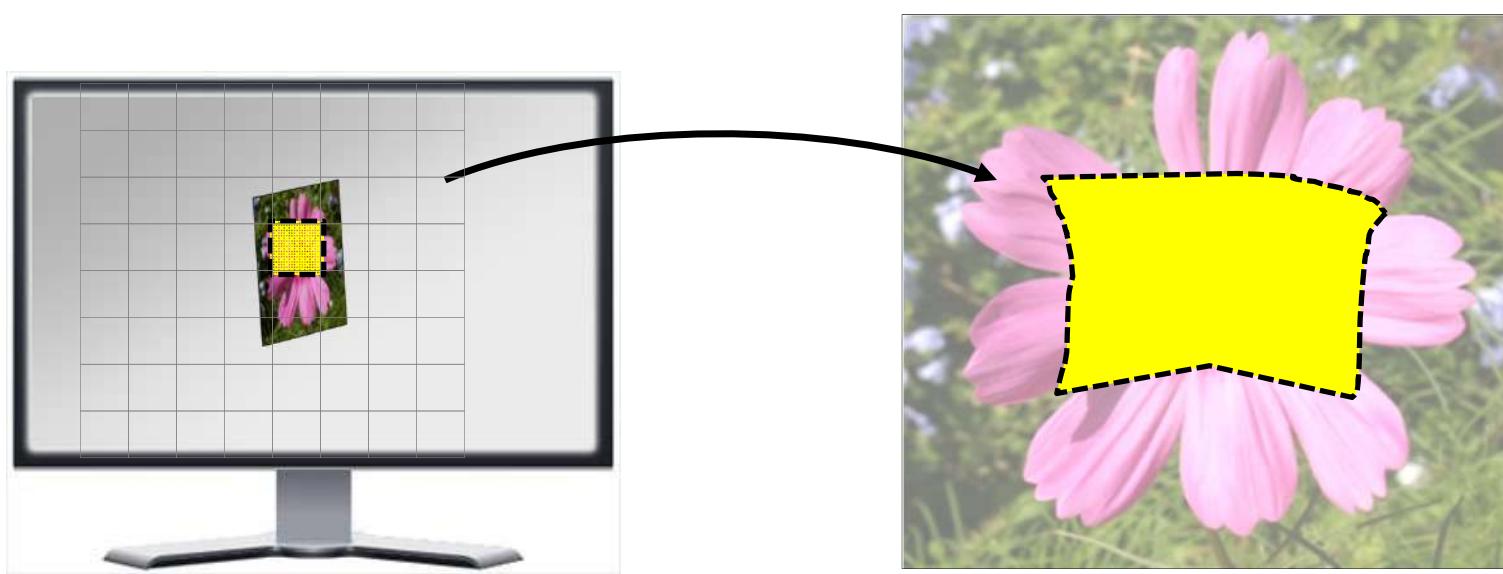
Magnification (naïve)



Minification (naïve)

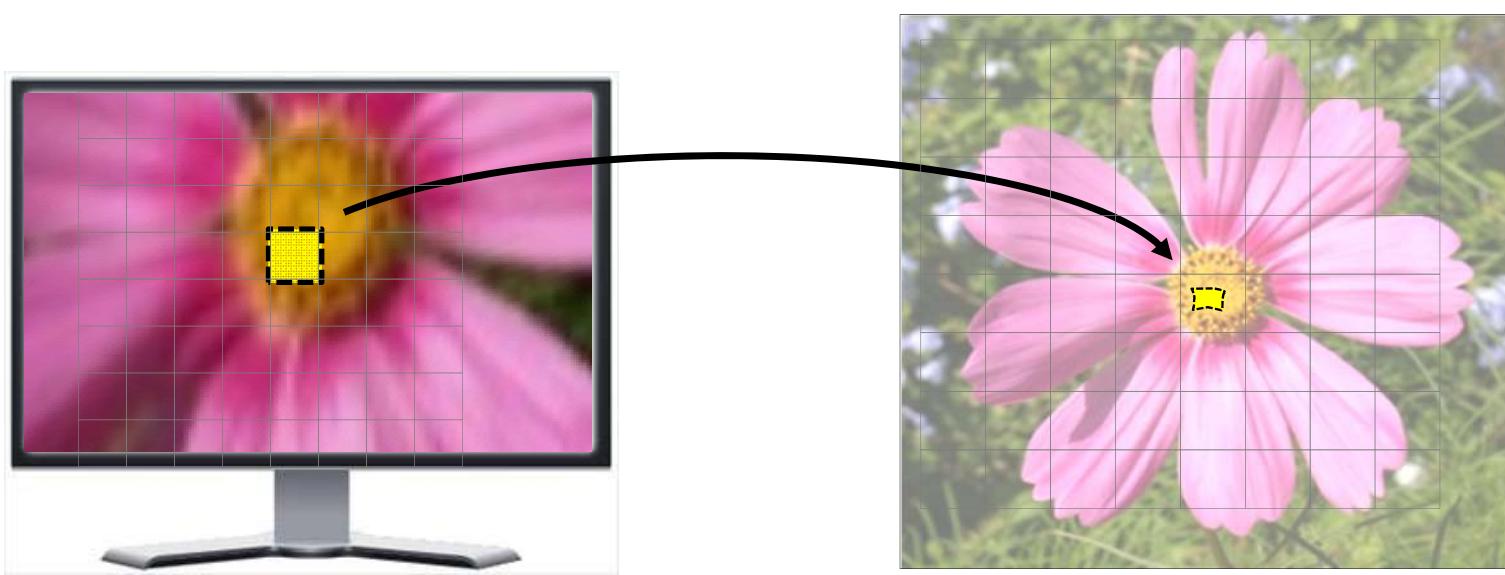


Preimatge d'un fragment



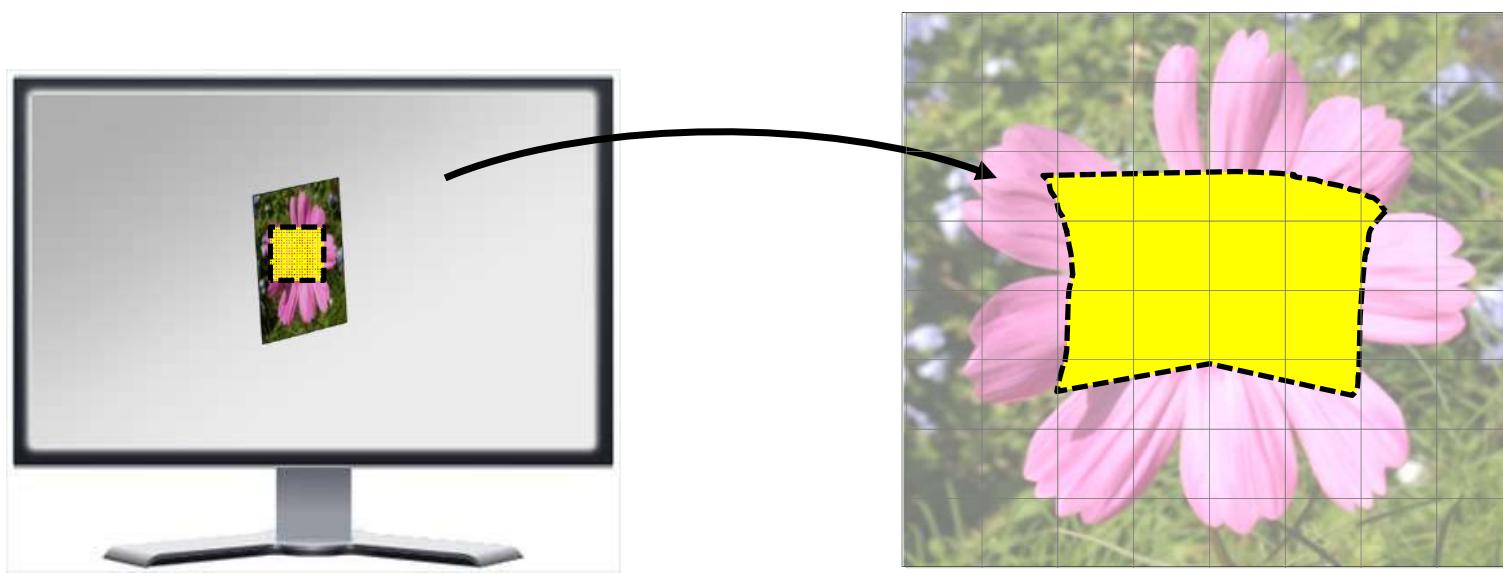
Ideal: color d'un pixel → color de la seva *preimatge* a la textura

Magnification



Magnification → la preimatge és < texel

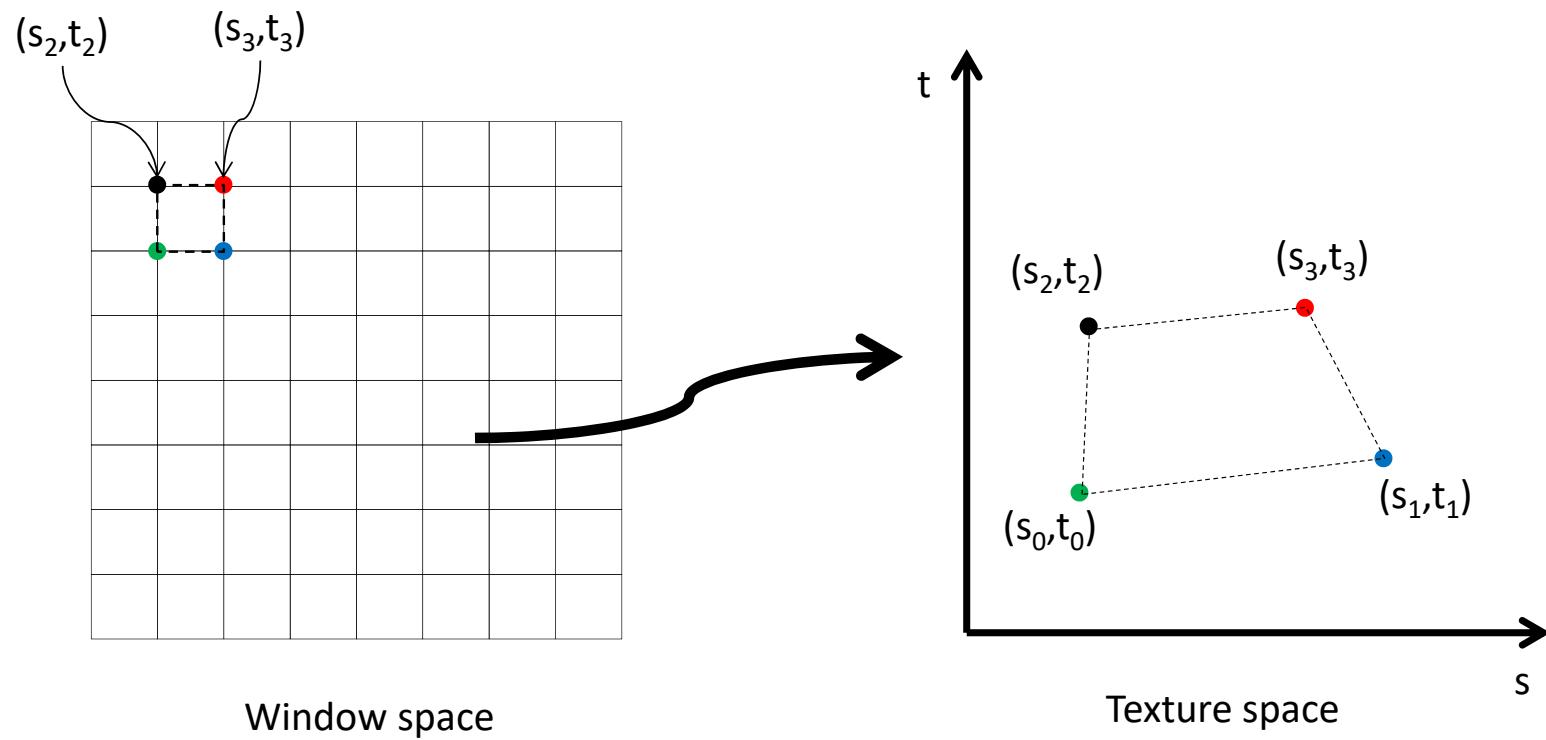
Minification



Minification → la preimatge és > texel

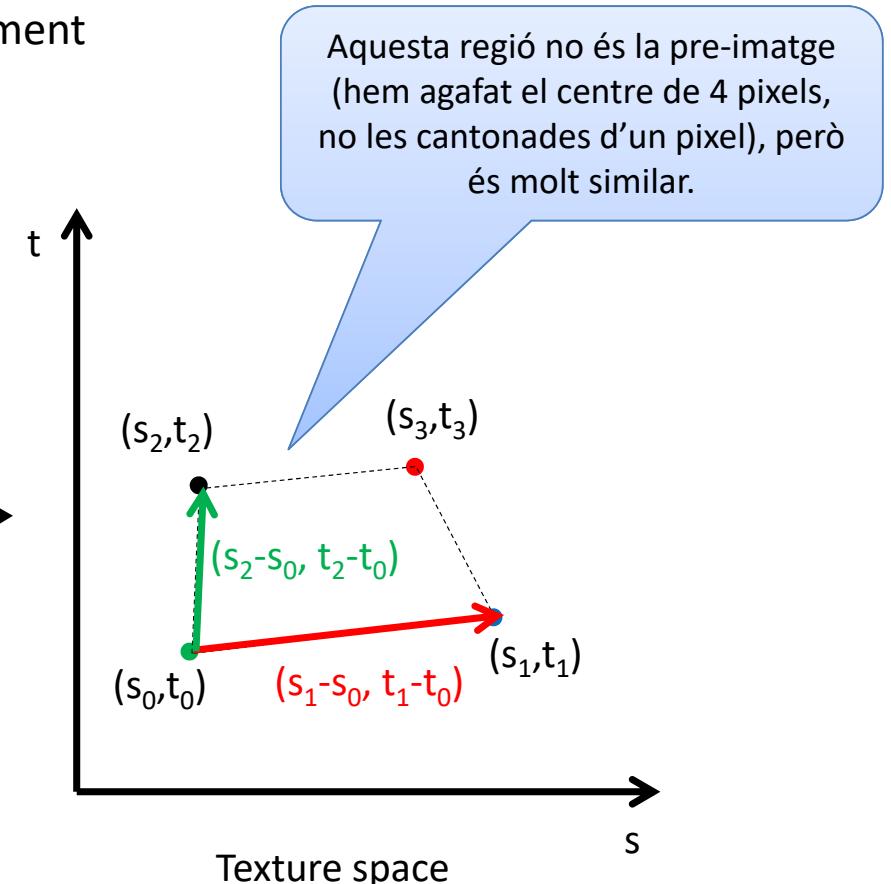
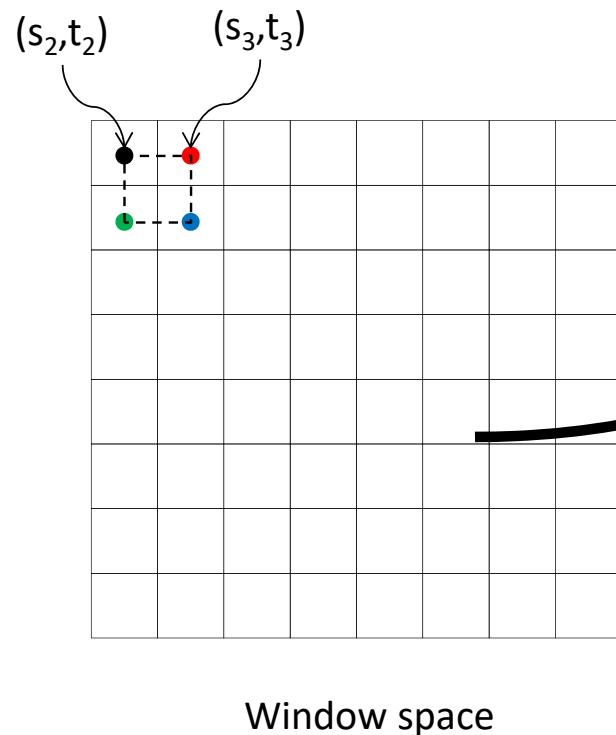
MAGNIFICATION O MINIFICATION?

Idealment



Aproximació que fa OpenGL

Usem les coords (s, t) del centre de cada fragment



Aproximació que fa OpenGL

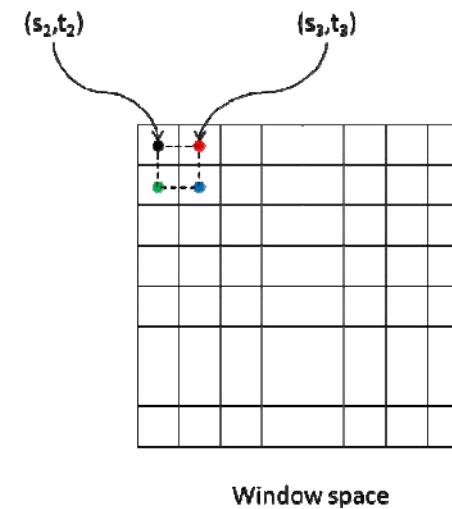
- Siguin $s(x,y)$, $t(x,y)$ les coordenades s,t del fragment (x,y)
- Derivades parcials de $s(x,y)$:

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

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

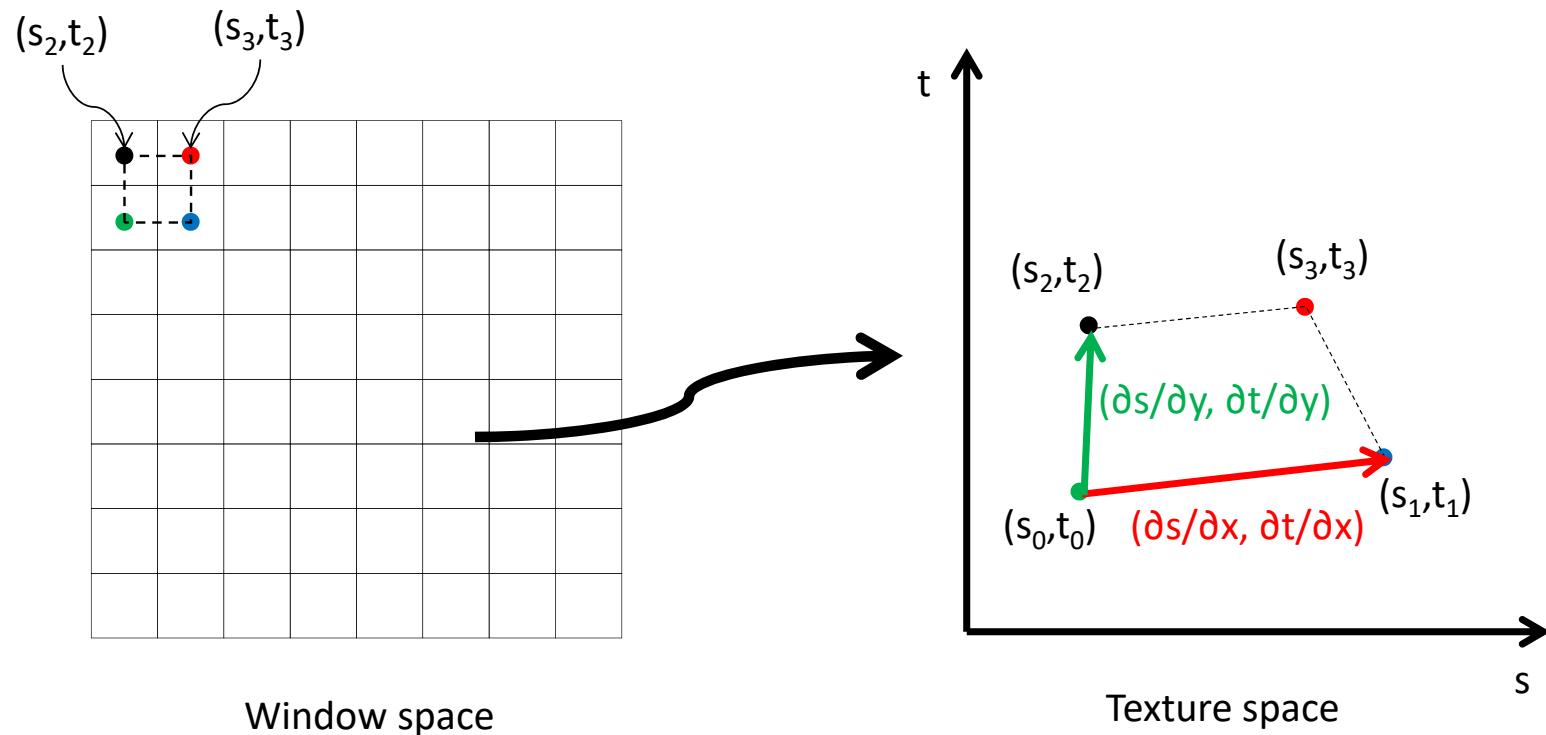
$$\begin{aligned}\frac{\partial s}{\partial x} &\approx dFdx(\text{texCoord}.s) \\ \frac{\partial s}{\partial y} &\approx dFdy(\text{texCoord}.s)\end{aligned}$$

(anàlogament per t)



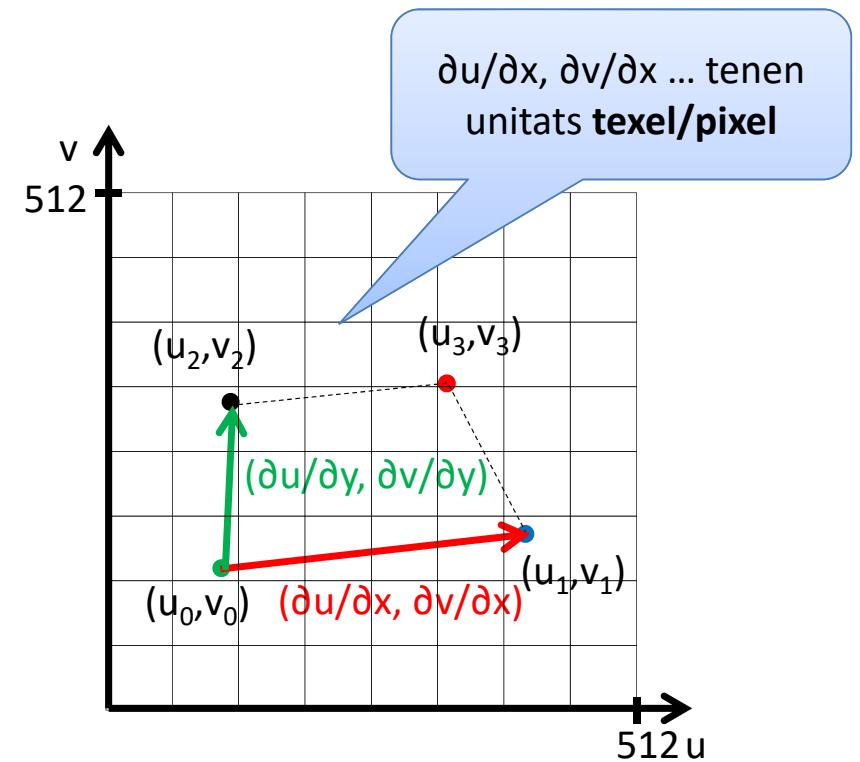
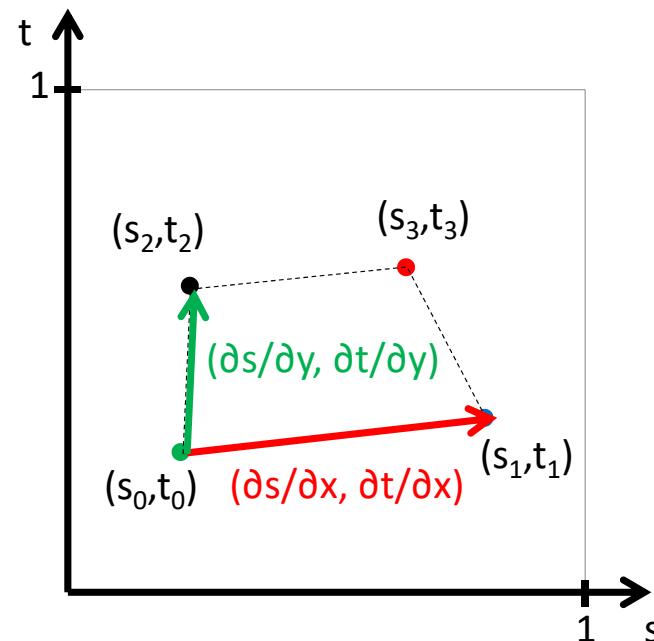
Aproximació que fa OpenGL

Aproximació de la mida de la pre-imatge (**en espai normalitzat de textura**)



Aproximació que fa OpenGL

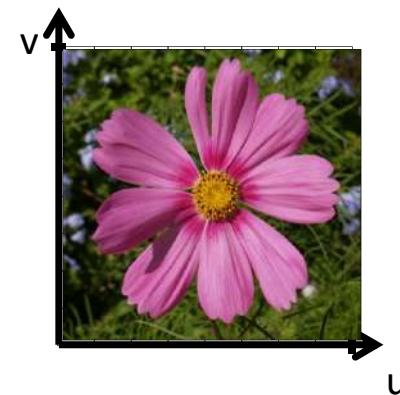
Mida de la pre-imatge (**en texels**) amb una textura 512x512



Exemple 1 (mapping 1:1)



Polígon projectat en WxH pixels



Textura WxH texels

En aquest cas un pixel correspon a un texel:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} = 1 \quad \frac{\partial v}{\partial x} = \frac{\partial u}{\partial y} = 0$$

Exemple 2 (magnification x2)



Fragments veïns
tenen coordenades
(u,v) properes



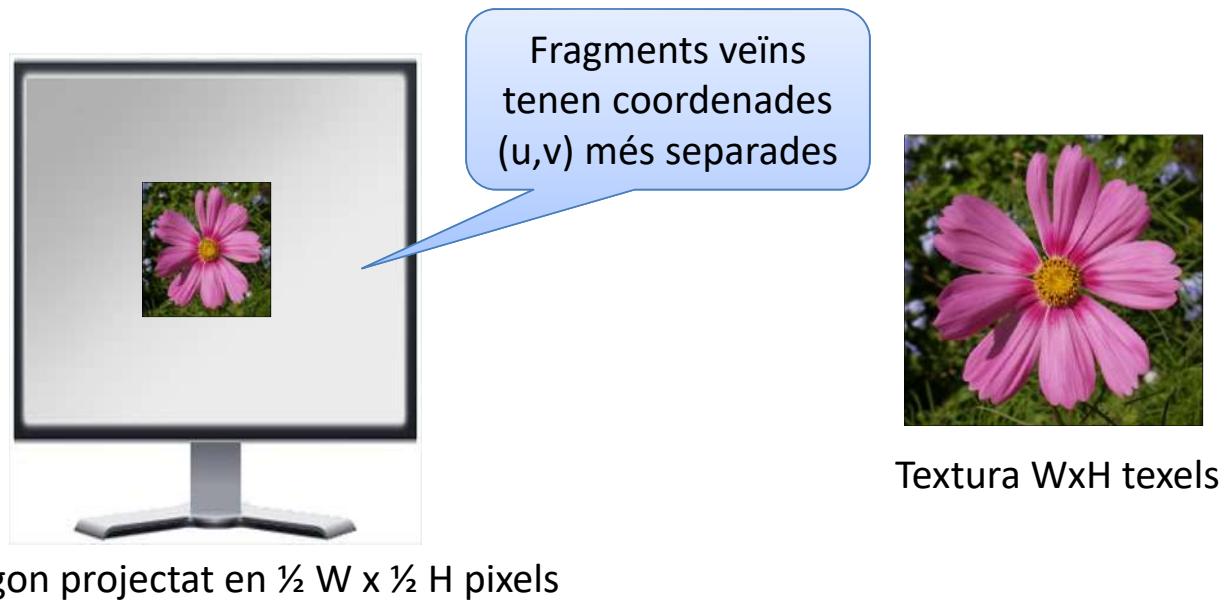
Textura WxH texels

Polígon projectat en 2Wx2H pixels

En aquest cas un pixel correspon a mig texel:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} = \frac{1}{2} \quad \frac{\partial v}{\partial x} = \frac{\partial u}{\partial y} = 0$$

Exemple 3 (minification x2)



En aquest cas un pixel correspon a dos texels:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} = 2 \quad \frac{\partial v}{\partial x} = \frac{\partial u}{\partial y} = 0$$

Exemple 4 (anisotròpic)



Textura WxH texels

En direcció horitzontal → magnification
En direcció vertical → minification

Exemple 4 (anisotròpic)

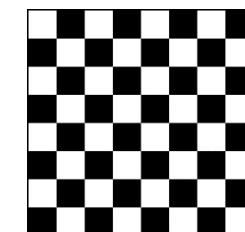
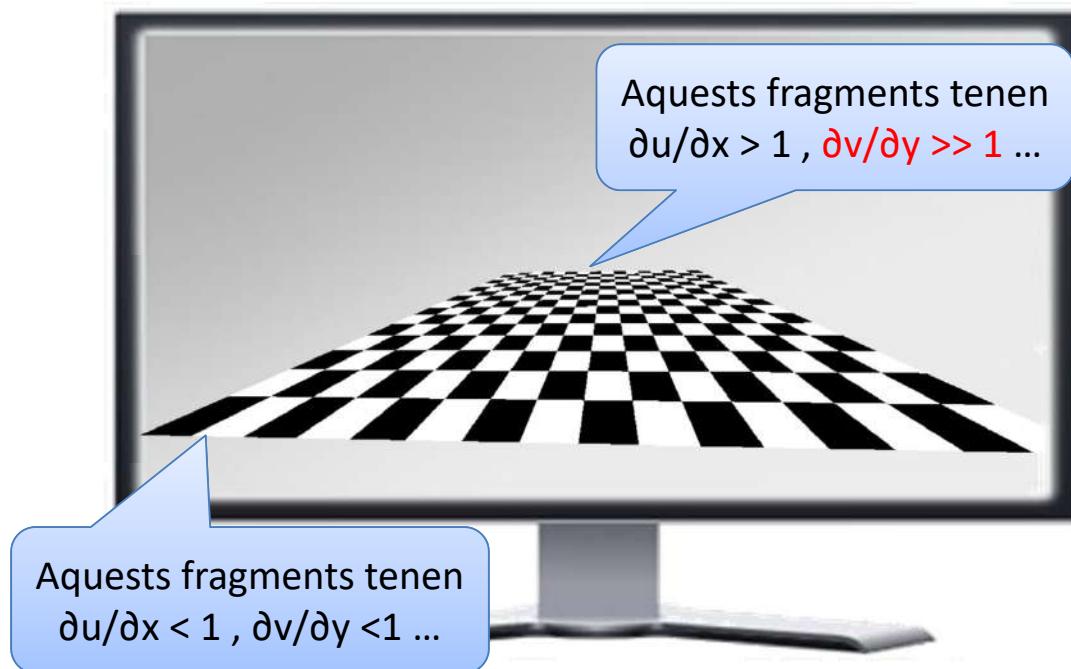


Textura WxH texels

En direcció horitzontal → magnification
En direcció vertical → minification

$$\begin{aligned}\partial u / \partial x &= \frac{1}{2} & \partial v / \partial x &= 0 \\ \partial u / \partial y &= 0 & \partial v / \partial y &= 2\end{aligned}$$

Exemple 5

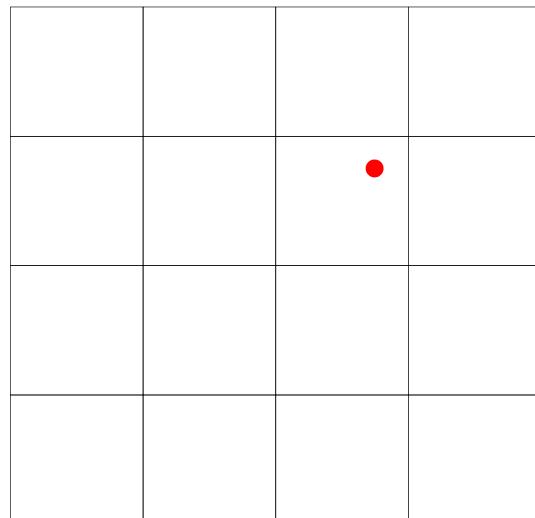


Textura

TEXTURE FILTERS

Texture filters

Determinen com s'avalua **texture(sampler, texCoord)**



Textura WxH texels

MAGNIFICATION

Magnification filters

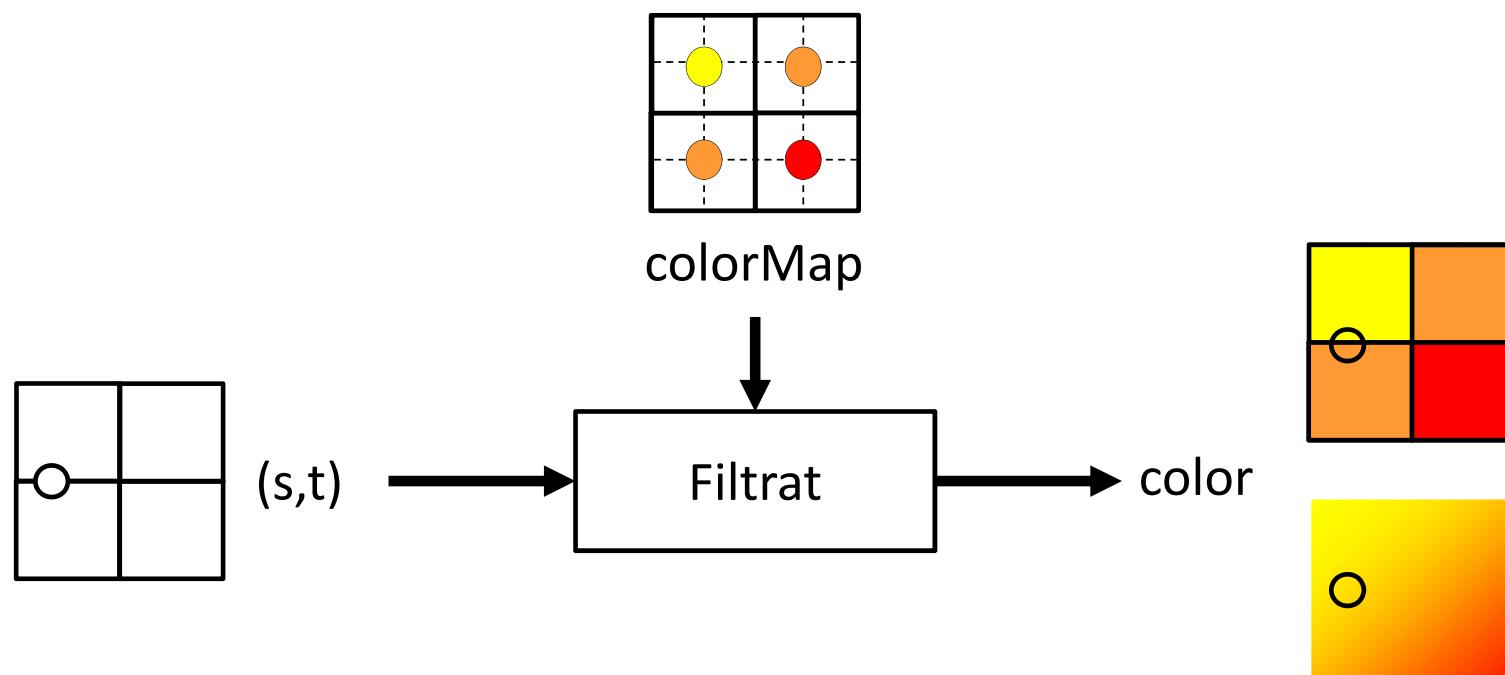
- GL_NEAREST
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
- GL_LINEAR
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

```
// 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());
glTexParameterf(...)

// Bind textures, set uniforms...
g.glActiveTexture(GL_TEXTURE0);
g glBindTexture(GL_TEXTURE_2D, textureId0);
program->bind();
program->setUniformValue("colorMap", 0);

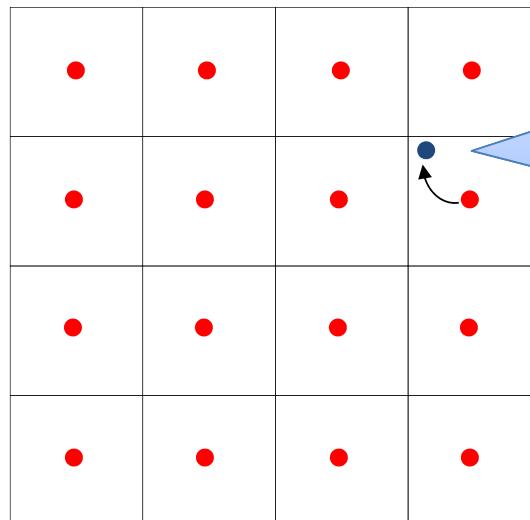
...
```

Magnification filters



Magnification filters

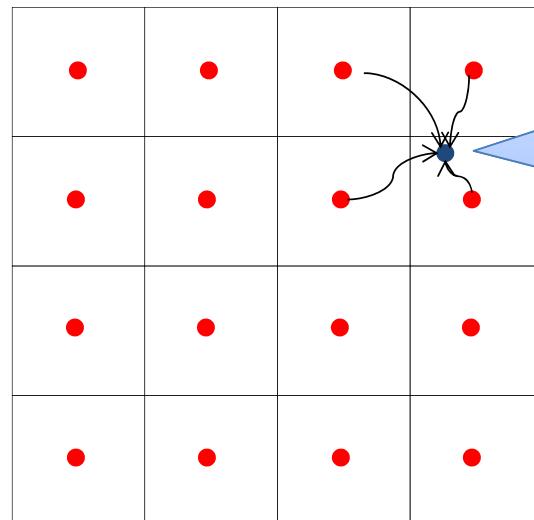
GL_NEAREST: nearest neighbor sampling



Amb nearest neighbor sampling, el color d'aquesta mostra és el color del veí més proper

Magnification filters

GL_LINEAR: bilinear interpolation

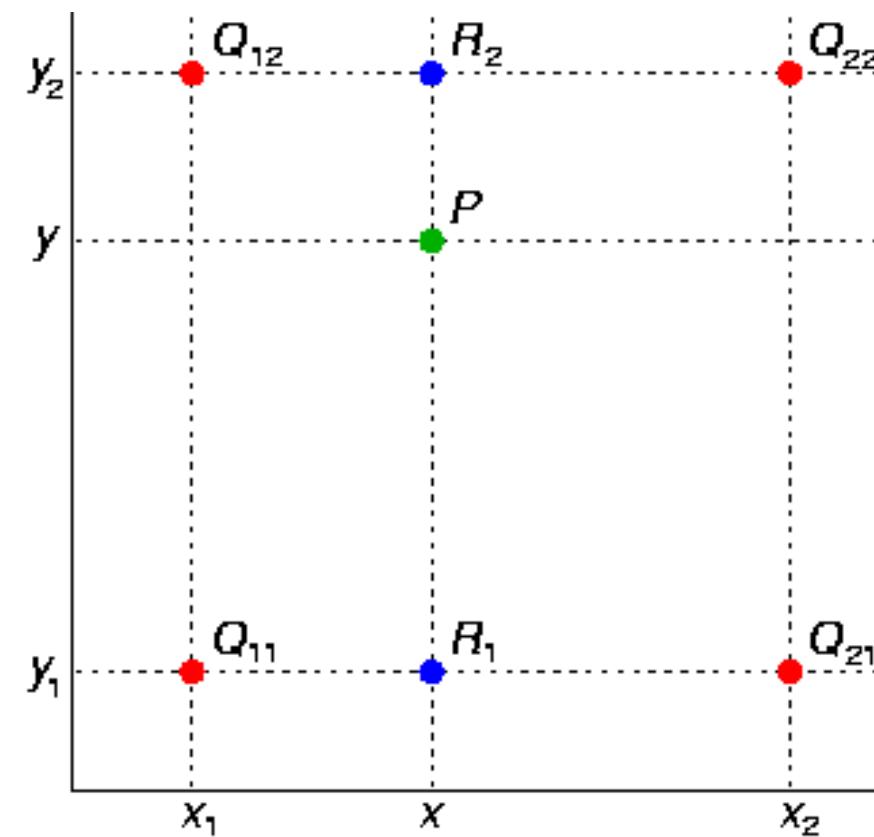
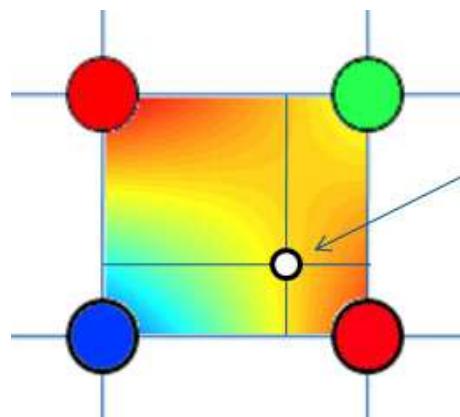


Amb interpolació bilineal,
el color d'aquesta mostra
és una mitjana ponderada
dels colors dels quatre
veïns més propers

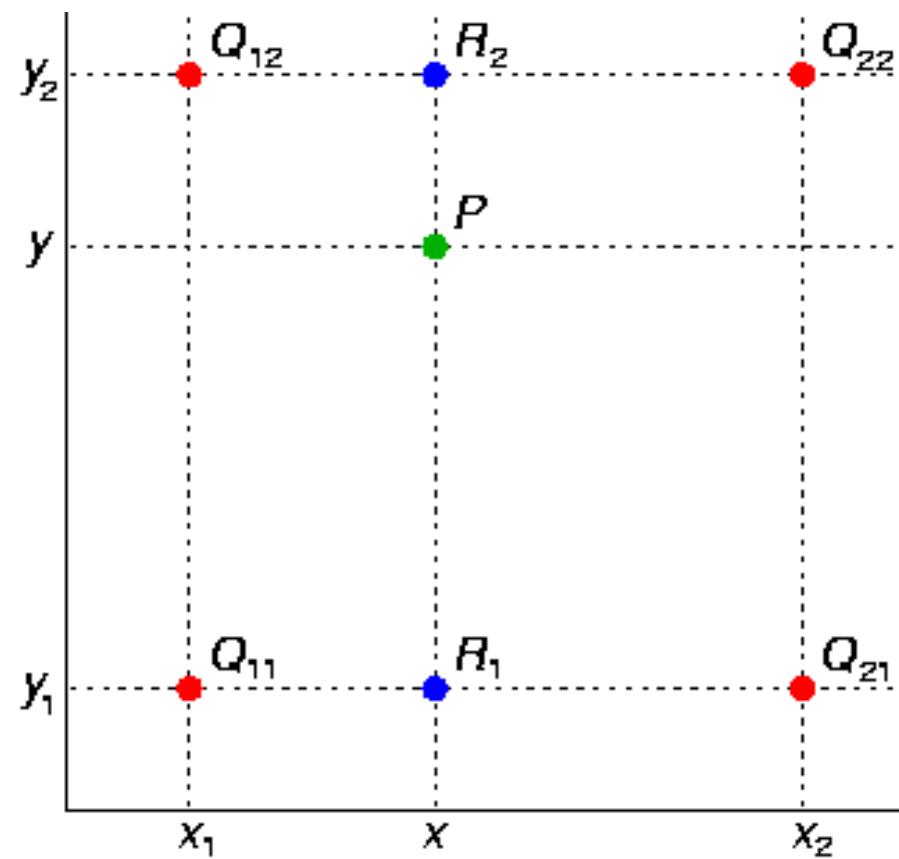
Comparació



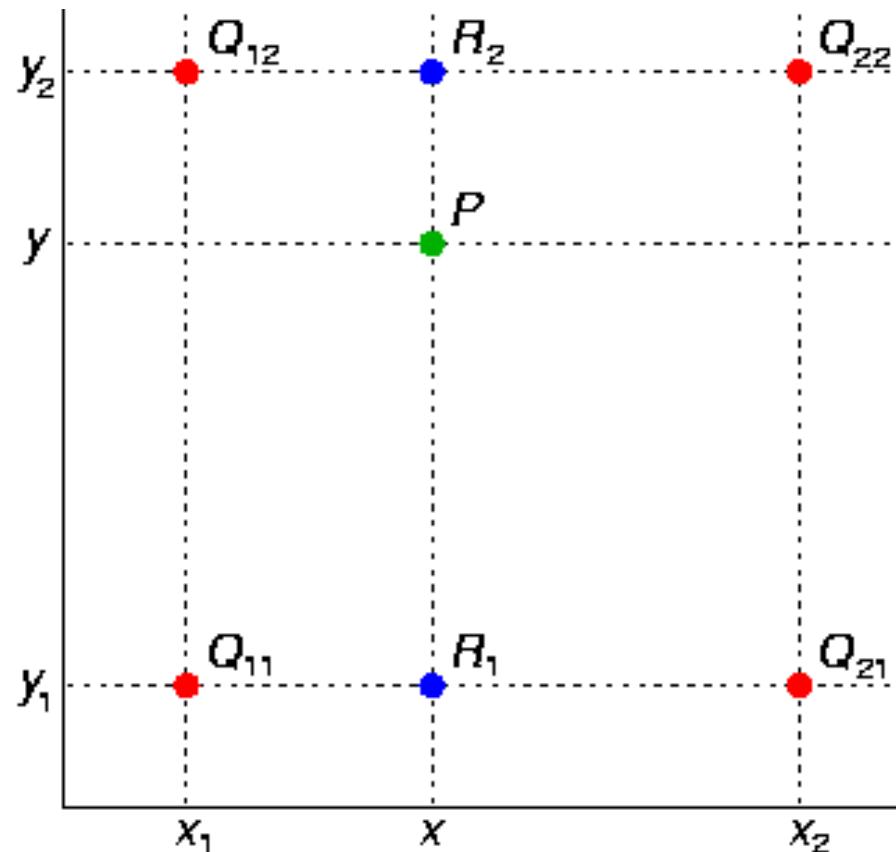
Bilinear interpolation



Bilinear interpolation



Bilinear interpolation



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

MINIFICATION

Minification filters

- **Magnification** → la preimatge d'un pixel en espai textura és petita → n'hi ha prou filtrant amb 2x2 texels.
- **Minification** → la preimatge d'un pixel és arbitràriament gran → no n'hi ha prou amb 2x2 texels!



Mipmapping

Idea bàsica: cada textura està representada amb diferents resolucions (level-of-details, LODs)

1x1 (LOD 8)

:

:

■ 16x16 (LOD 4)

■

32x32 (LOD 3)



64x64 (LOD 2)



128x128 (LOD 1)



256x256 (LOD 0)

Mipmapping

En alguns casos el LOD més adient λ és fàcil de calcular

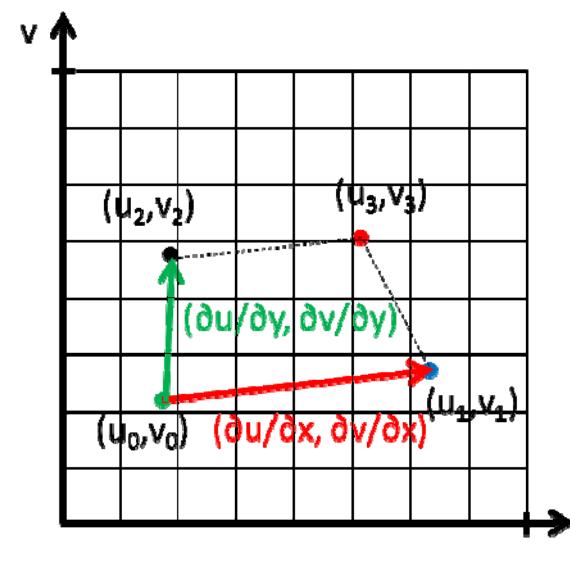
Minification	$\partial u / \partial x, \partial v / \partial y$	LOD
	1x	1
	2x	2
	4x	4
	8x	8
■	$2^\lambda x$	2^λ

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

Mipmapping

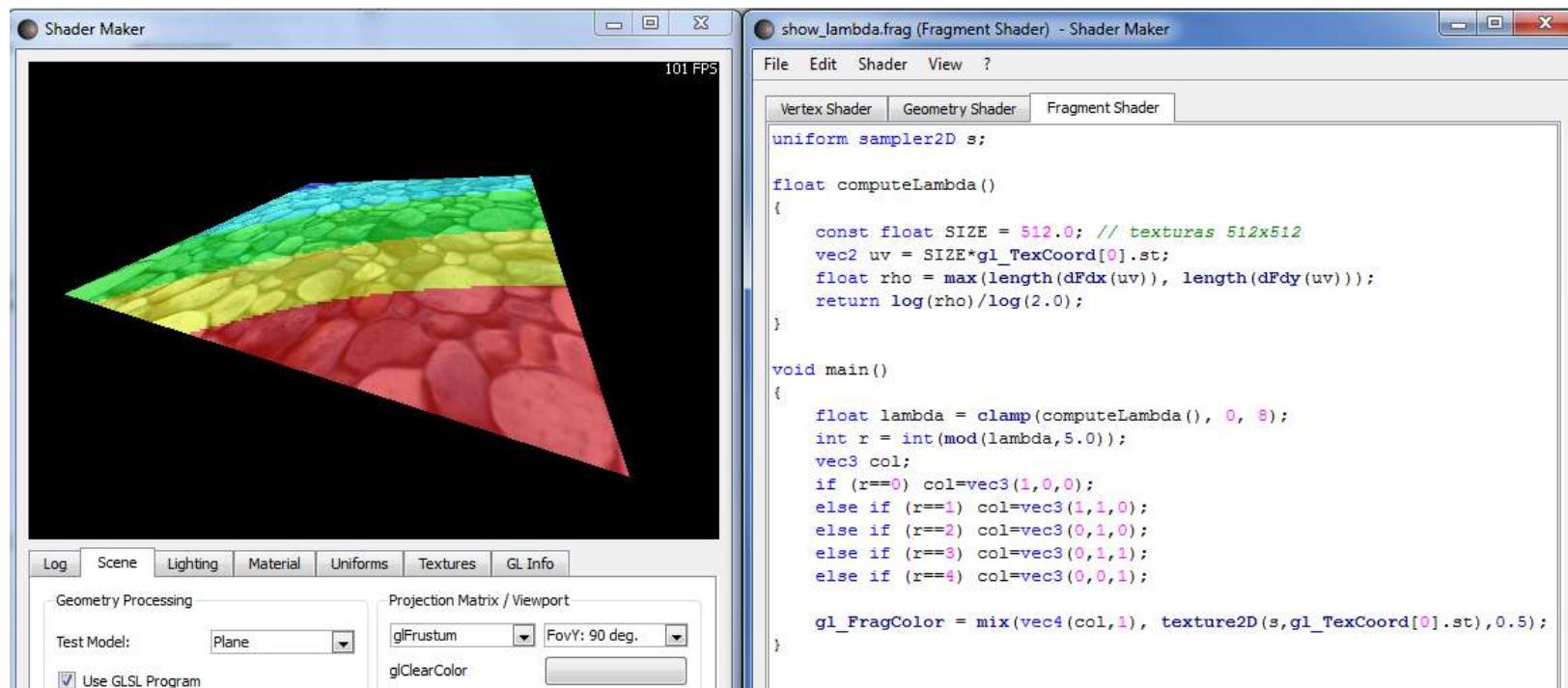
En general:

- 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 $\lambda = \log_2(\rho)$



Mipmapping

Demo: show_lambda.frag



Minification filters

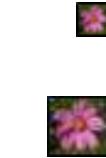
Without mipmapping

- GL_NEAREST // Nearest neighbor sampling on LOD 0
- GL_LINEAR // Bilinear interpolation on LOD 0

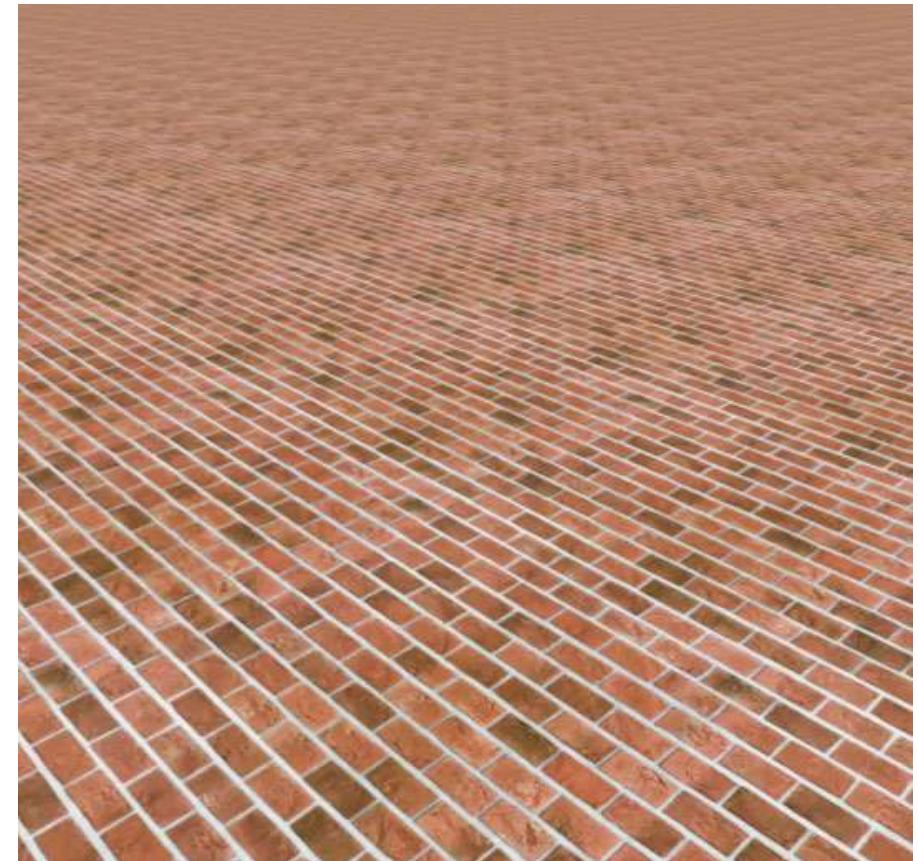
With mipmapping

- GL_NEAREST_MIPMAP_NEAREST // Nearest neighbor sampling on LOD $\text{int}(\lambda)$
- GL_LINEAR_MIPMAP_NEAREST // Bilinear sampling on LOD $\text{int}(\lambda)$
- GL_NEAREST_MIPMAP_LINEAR // c_0 = nearest neighbor on LOD $\text{int}(\lambda)$
// c_1 = nearest neighbor on LOD $\text{int}(\lambda+1)$
// mix($c_0, c_1, \text{fract}(\lambda)$)
- GL_LINEAR_MIPMAP_LINEAR // c_0 = bilinear sampling on LOD $\text{int}(\lambda)$
// c_1 = bilinear sampling on LOD $\text{int}(\lambda+1)$
// mix($c_0, c_1, \text{fract}(\lambda)$)

GL_<samplingWithinTheLODs>_MIPMAP_<oneOrTwoLODs>



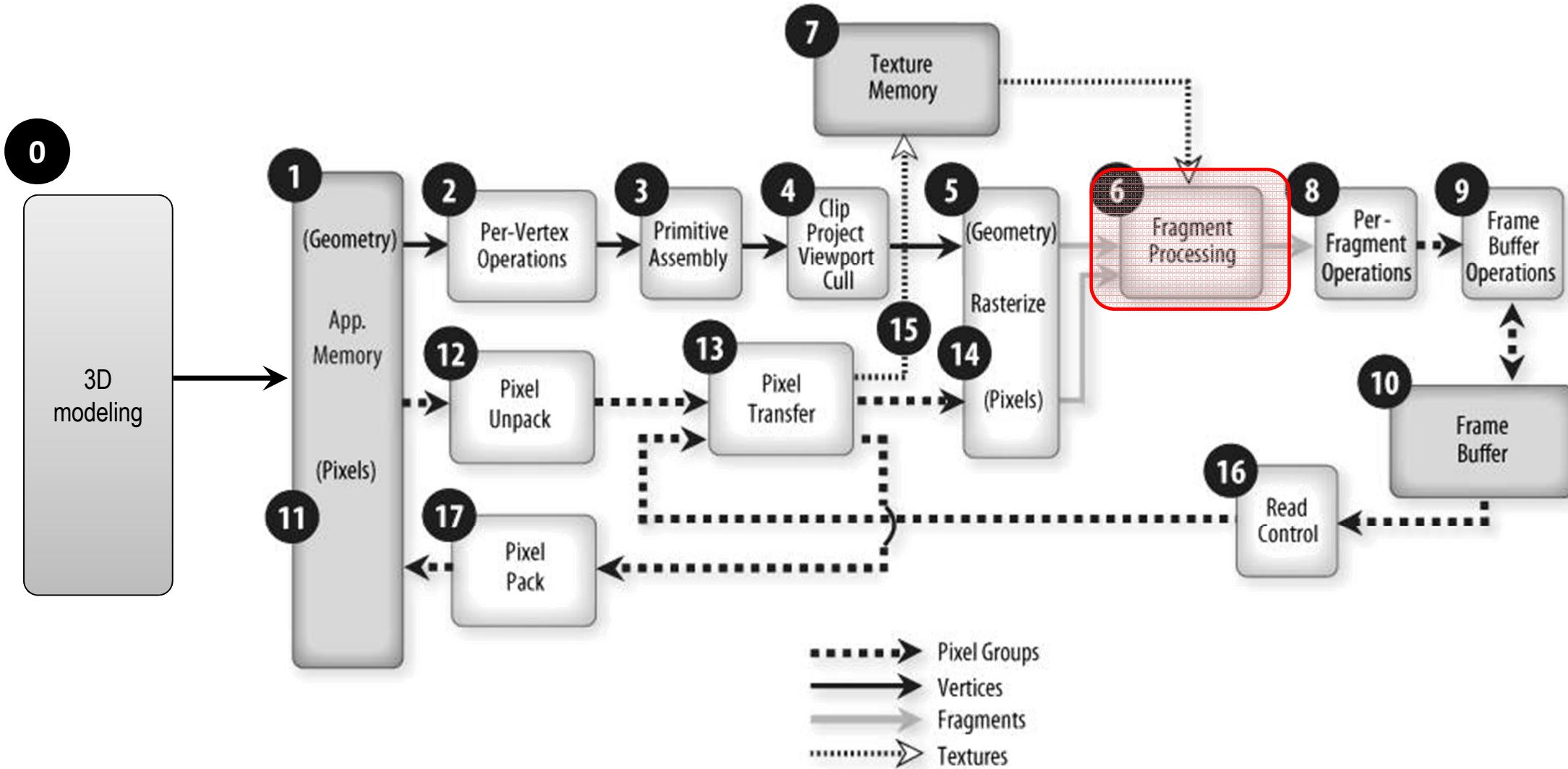
Comparació



Minification filters (2)

[Demo mipmapping]

COMBINACIÓ



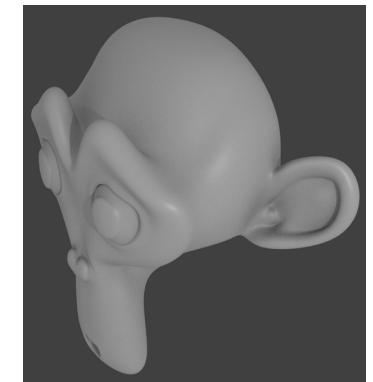
FS – exemple

```
uniform sampler2D colorMap;  
in vec2 vtexcoord;  
in vec4 frontColor;  
  
vec4 texColor = texture(colorMap, vtexcoord);  
...  
fragCoord = ???
```

Modes habituels

REPLACE: `fragColor = texColor;`

$$K_d I_d (N \cdot L) + K_s I_s (R \cdot V)^s$$



Modes habituels

MODULATE: `fragColor = texColor * frontColor;`

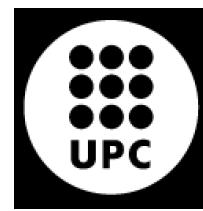
$$K_d I_d (N \cdot L) + K_s I_s (R \cdot V)^s$$



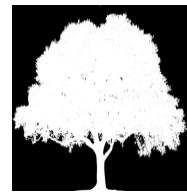
Modes habituales

DECAL:

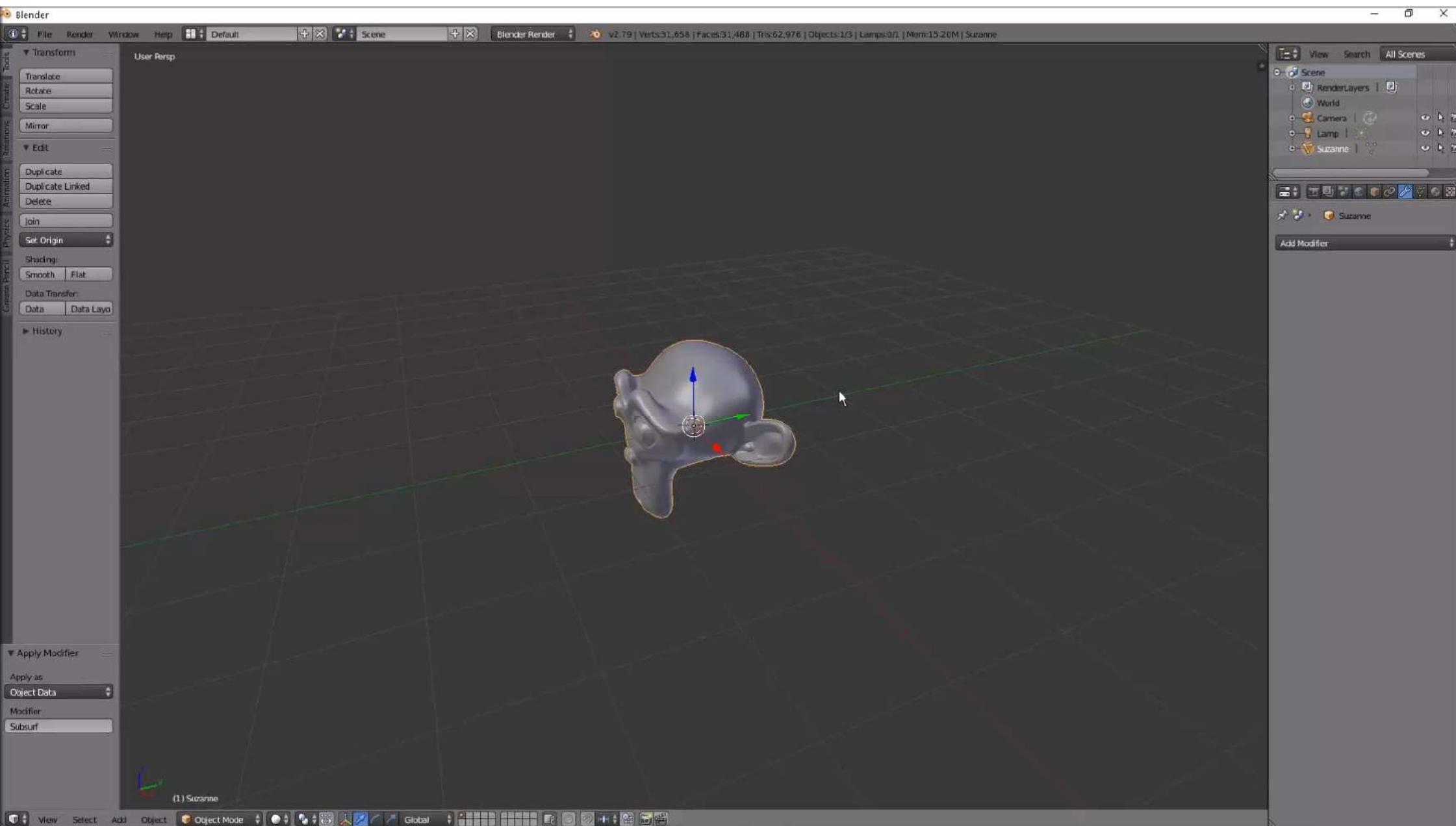
```
fragColor = mix(frontColor, texColor, texColor.a);
```



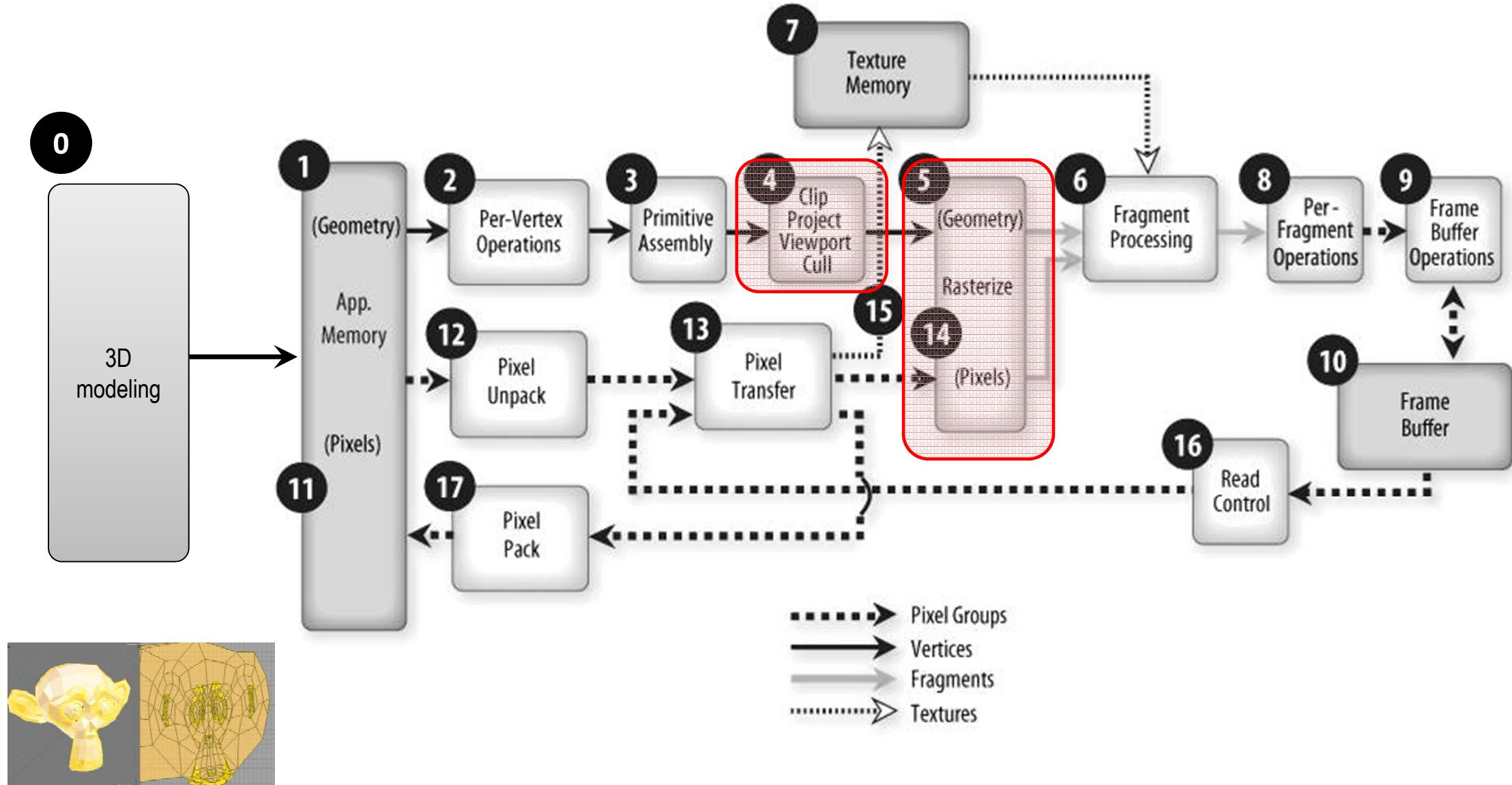
```
if (texColor.a < alphaThreshold)  
    discard;
```



GENERACIÓ D'UN LIGHT MAP



INTERPOLACIÓ DE COORDS DE TEXTURA

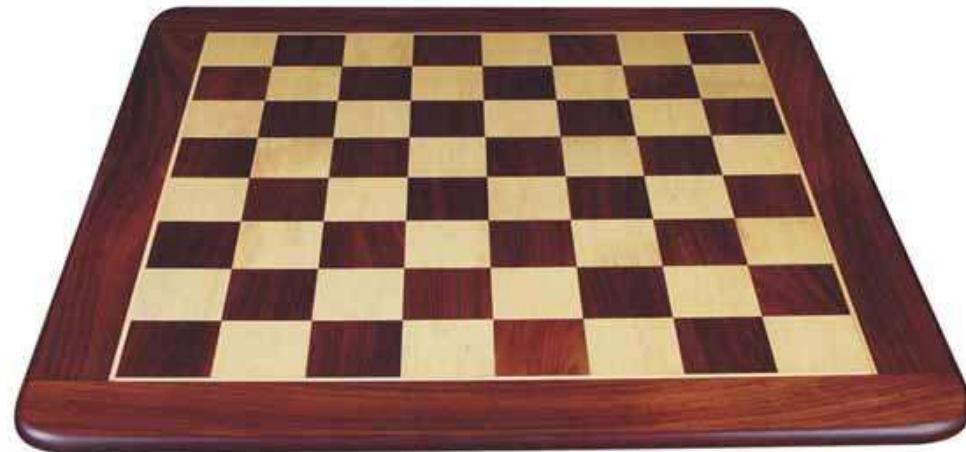
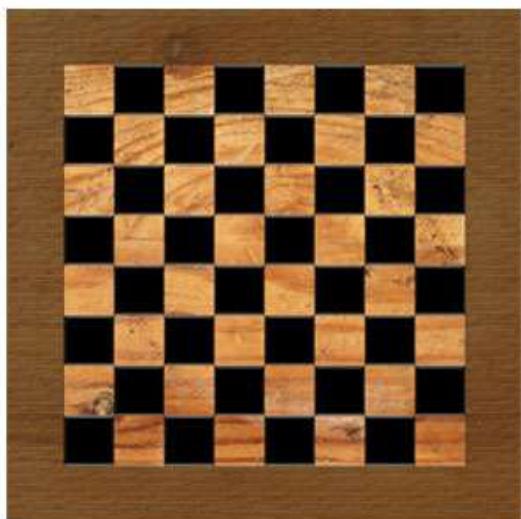


PERSPECTIVE-CORRECT INTERPOLATION

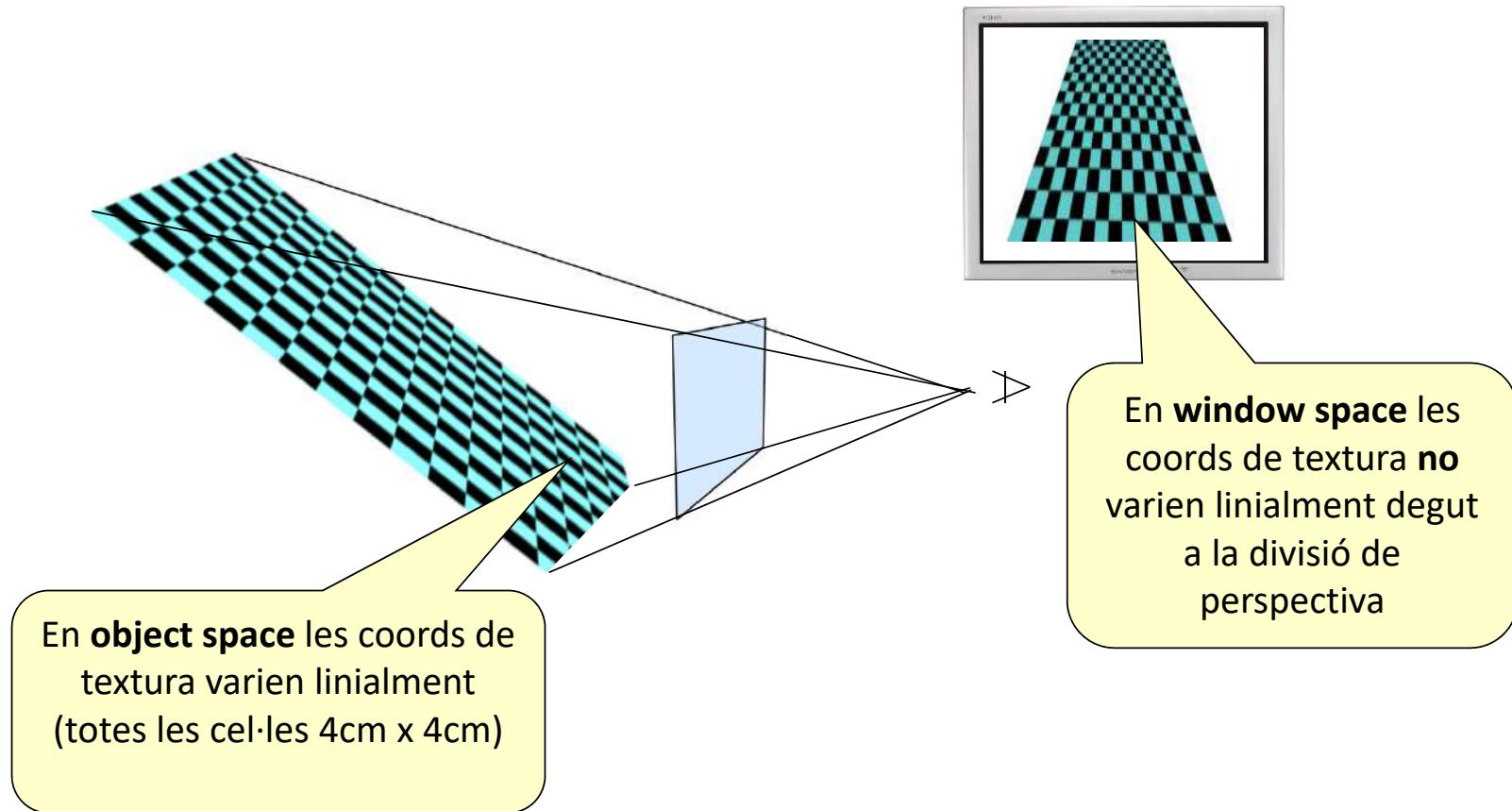
Perspective deformation



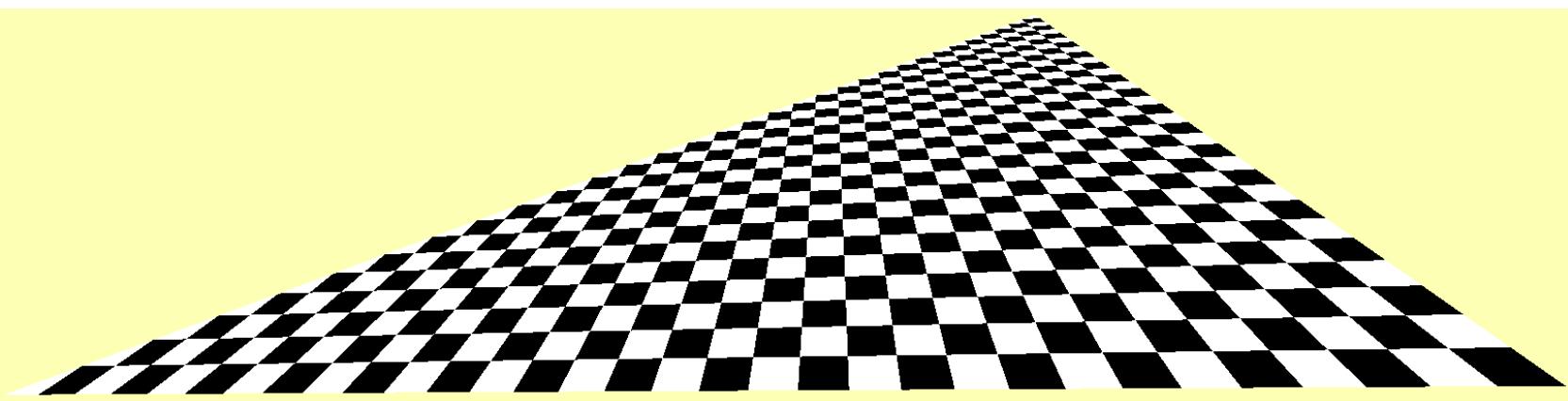
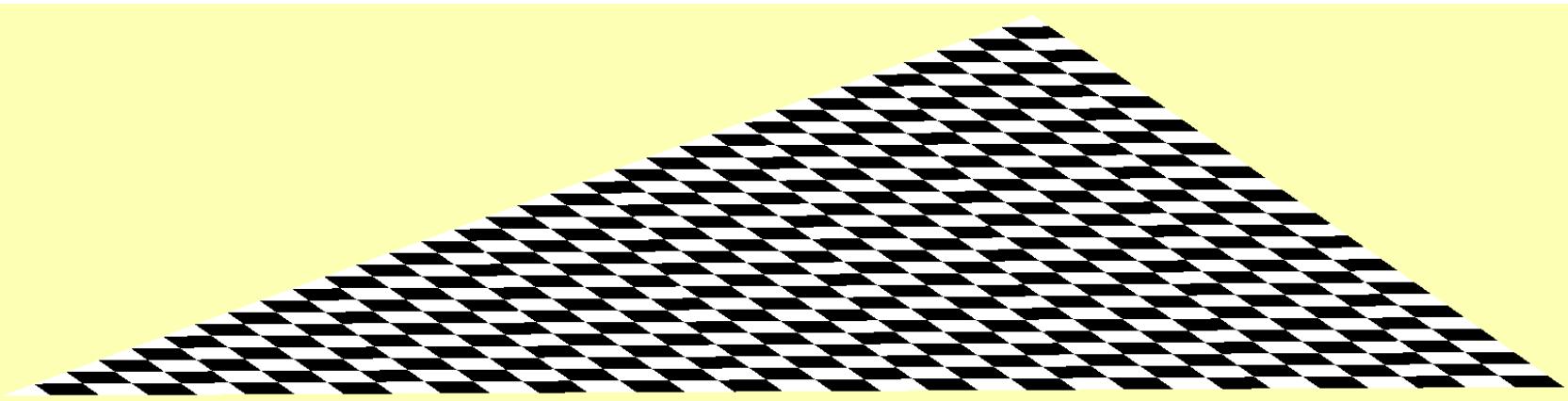
Perspective deformation



Perspective-correct interpolation



Perspective-correct interpolation



[Demo: pers.vert, pers.frag]

Perspective-correct interpolation

Solucions:

- a) Interpolem (s, t) en **object space** (també valdria en **world, eye i clip space**, perquè són abans de la div. de perspectiva)
- b) Interpolem (sw, tw, w) en **window space**, obtenint un texel (s, t, q) ; accedirem a la textura amb $(s/q, t/q)$

Recordeu que $w = 1/w_c = -1/z_e$

Perspective-Correct Interpolation

[Demostració de (b): consulteu per exemple l'article de Kok-Lim Low]

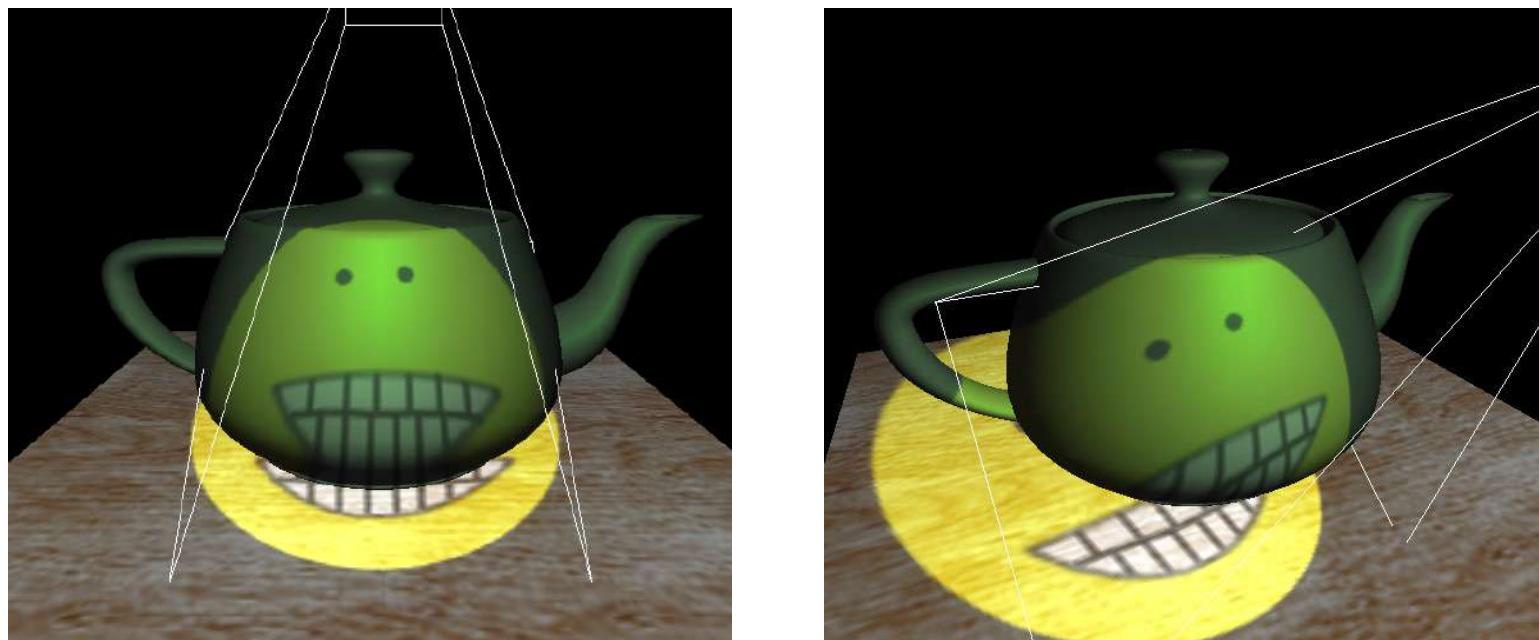
Kok-Lim Low

Department of Computer Science
University of North Carolina at Chapel Hill
Email: lowk@cs.unc.edu

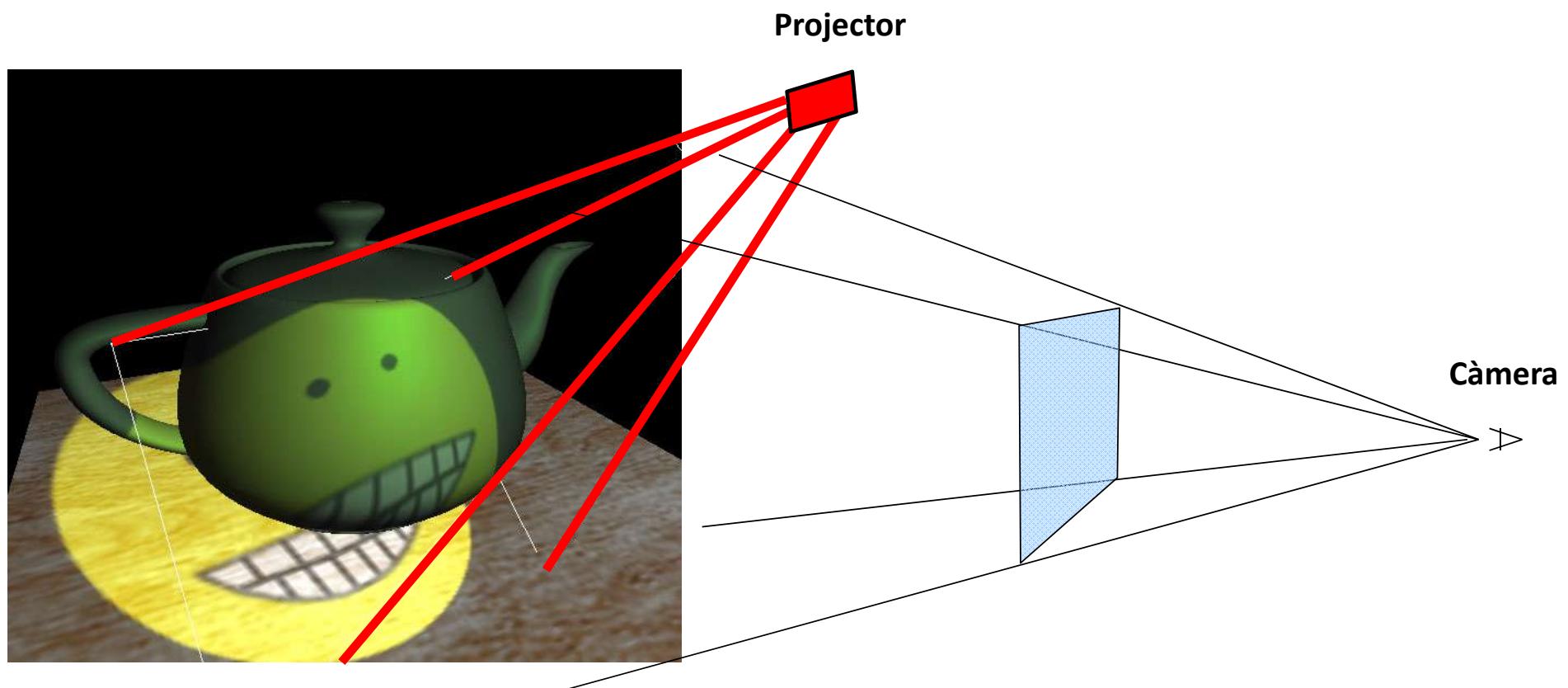
March 12, 2002

PROJECTIVE TEXTURE MAPPING

Projective texture mapping



Projective texture mapping



Opció 1 (incorrecta)

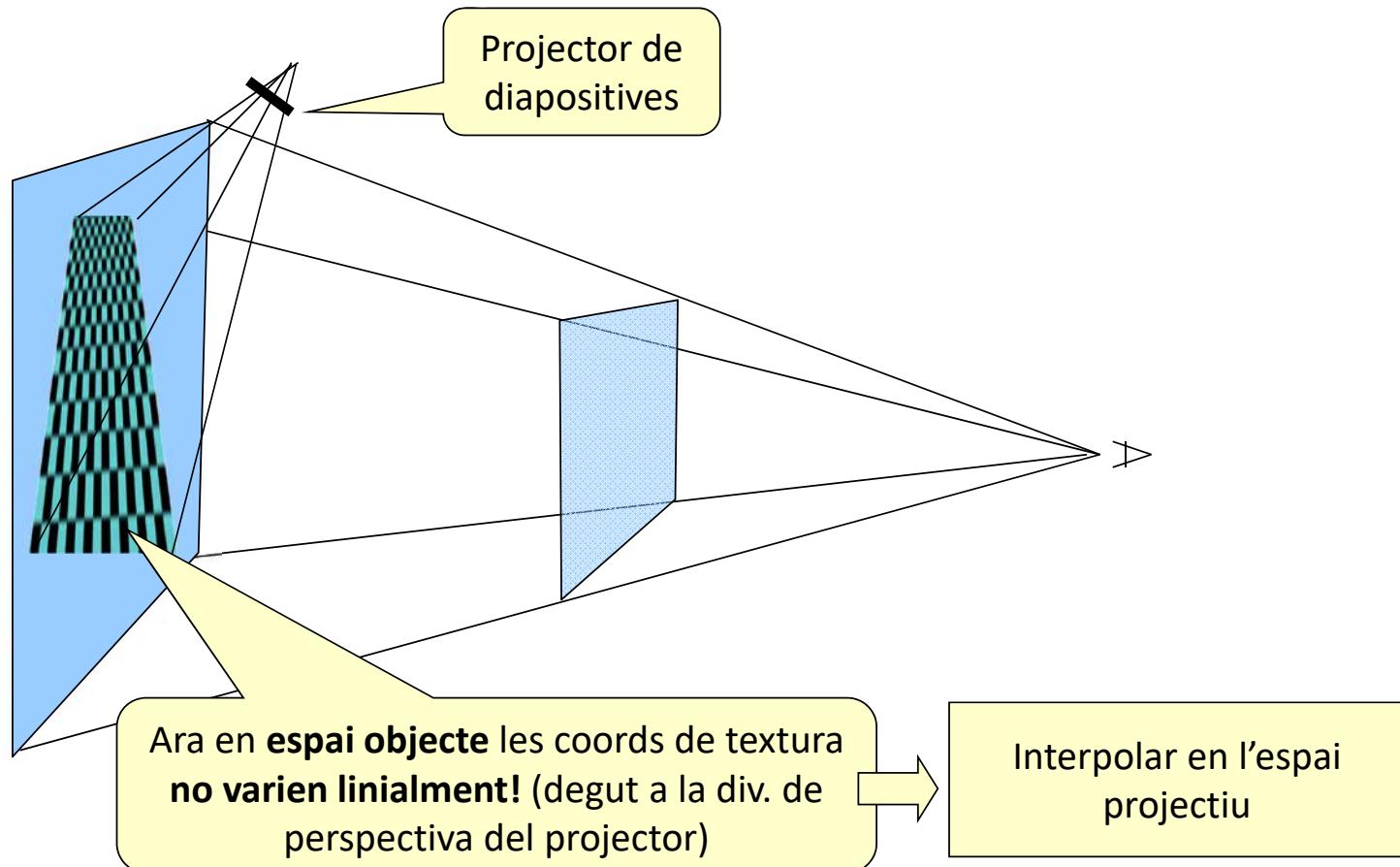
VS – generació coords de textura

Passa el vèrtex de *object space* a *window space* (viewport 1x1)
Calcula (s,t) com (x,y) del resultat

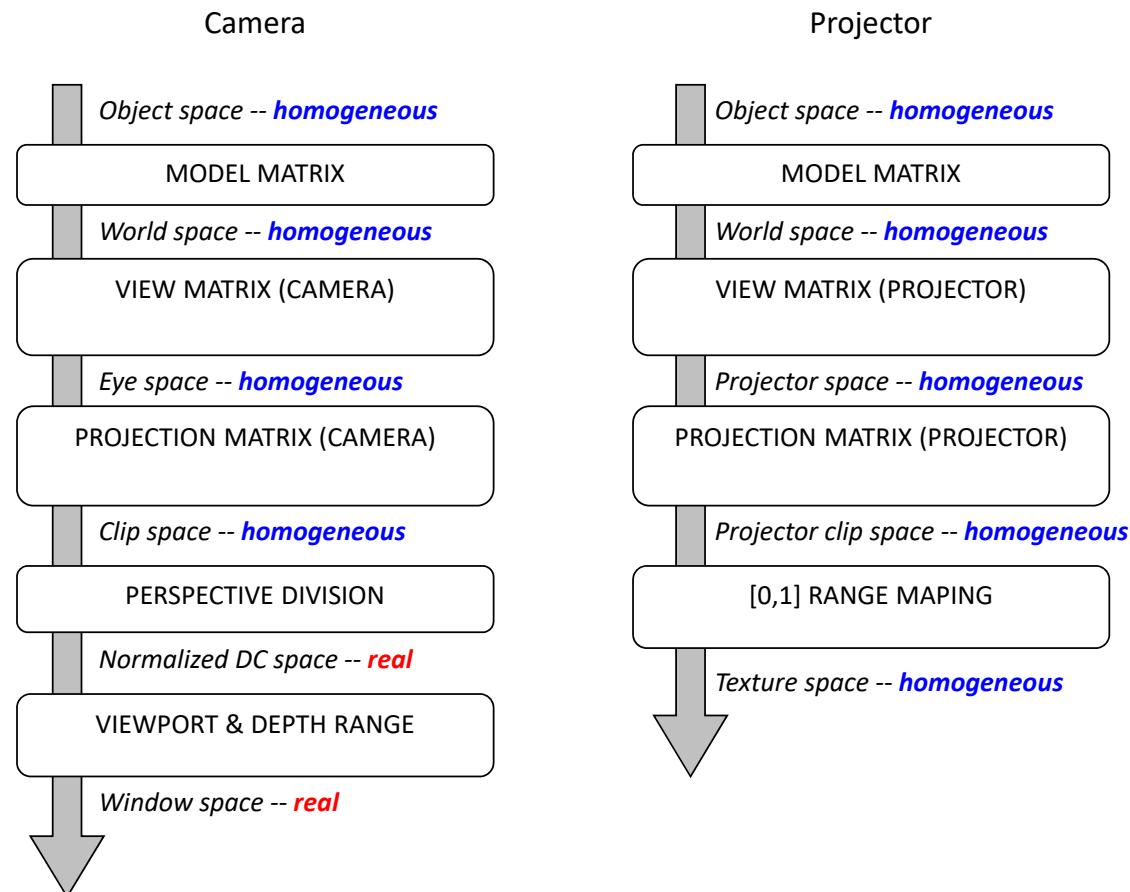
FS – accés a textura

Usa (s,t) per accedir a la textura

Projective-space interpolation



Projective texture mapping



Opció 2 (correcta)

VS – generació coords de textura

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

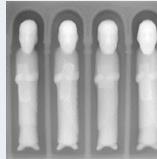
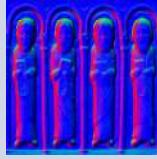
Calcula (s, t, p, q) com (x, y, z, w) del resultat

FS – accés a textura

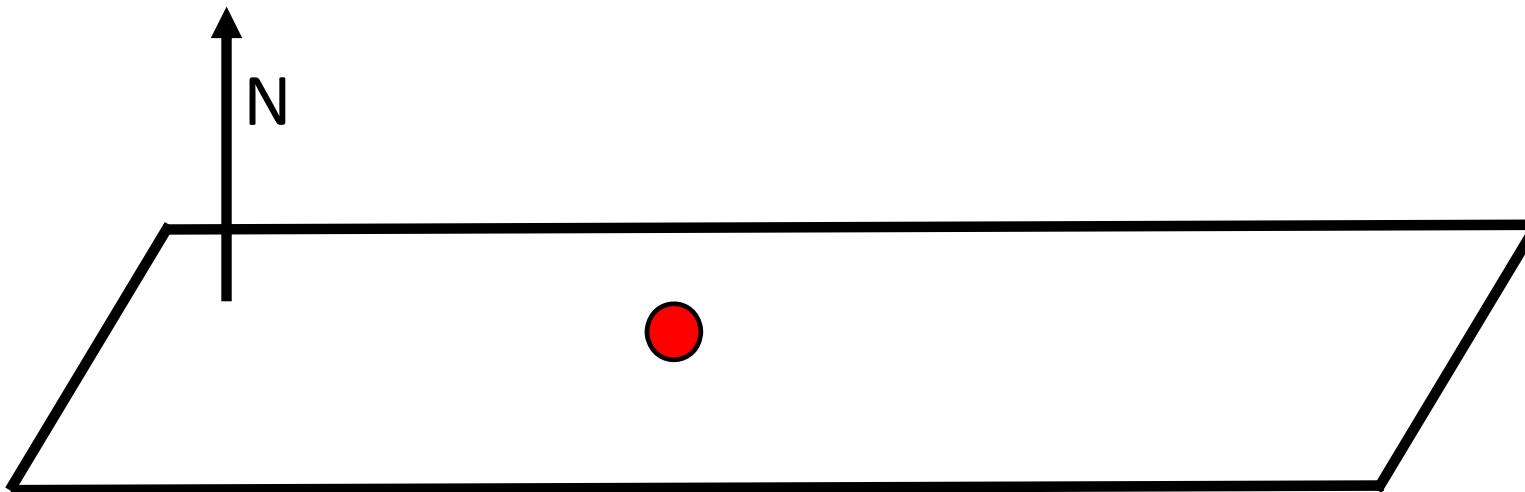
Usa $(s/q, t/q)$ per a accedir a la textura

Color, bump, parallax, relief i displacement mapping

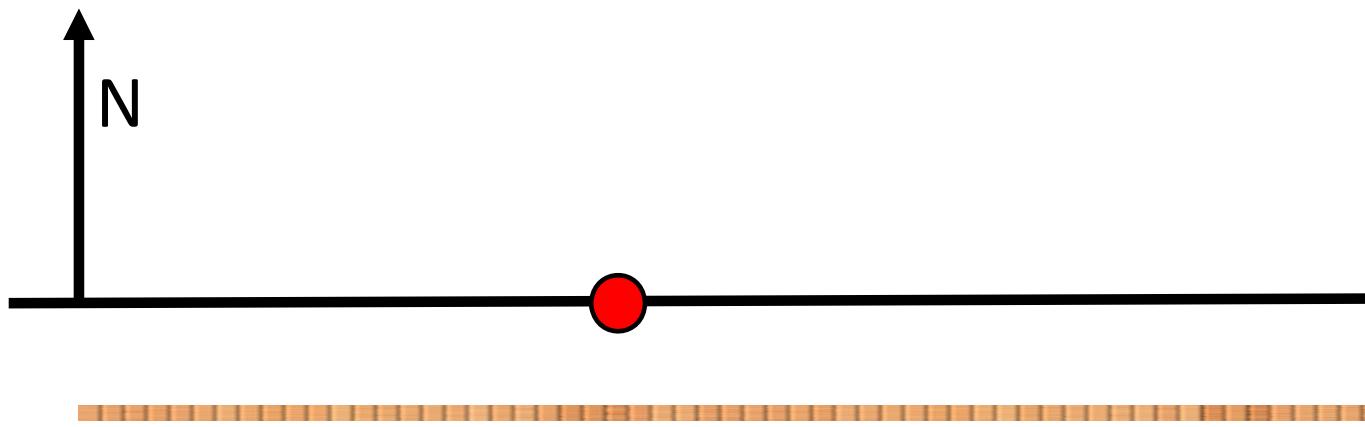
APLICACIONES

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	$(s+d_s, t+d_t)$	S	N	N	FS	
Relief mapping	Normal + D 3+1 o 4	Modificar la normal; descartar fragments	$(s+d_s, t+d_t)$	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

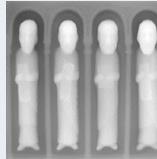
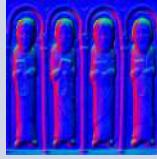


Color mapping

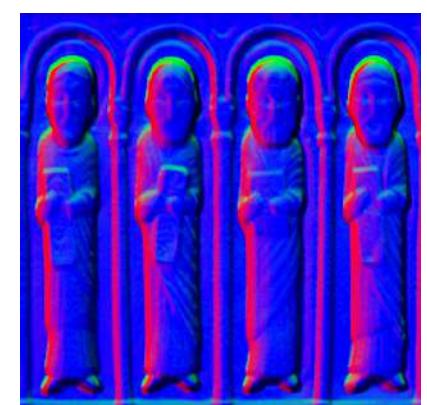
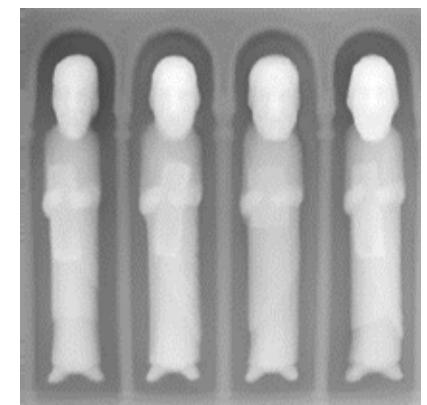
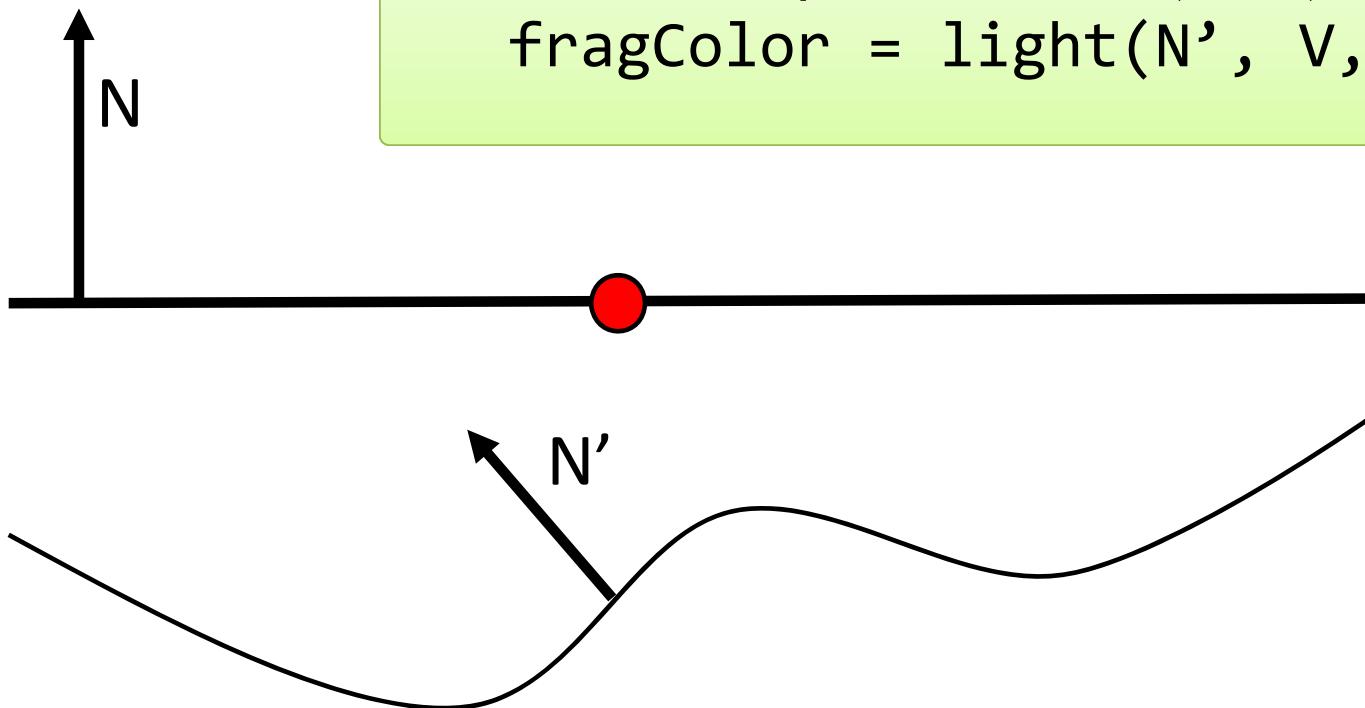


Color mapping

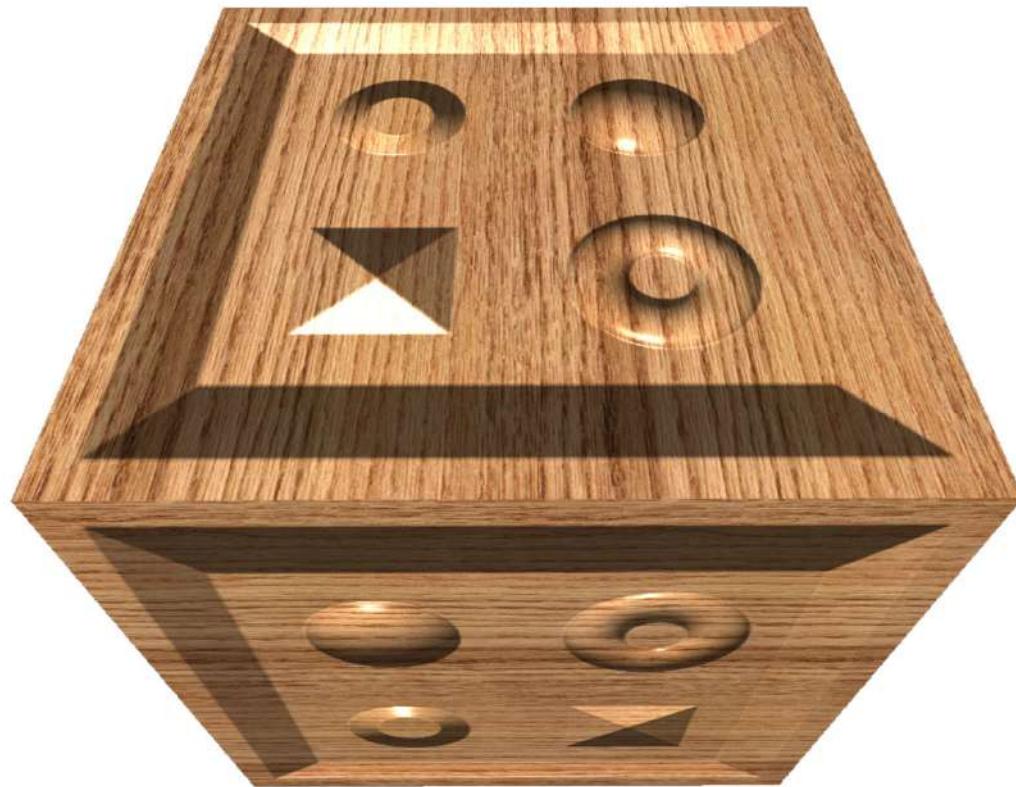


Tècnica	Info a la textura	Ús de la textura	Coords de textura	View parallax	Self-occlusion	Detailed silhouette	On s'aplica	
Color mapping	RGB 3	Kd del material	(s,t)	-	-	-	FS	
Bump mapping	D 1		Modificar la normal	(s,t)	N	N	N	FS
Normal mapping	Normal 3		Modificar la normal	(s,t)	N	N	N	FS
Parallax mapping	Normal + D 3+1 o 4	Modificar la normal	$(s+d_s, t+d_t)$	S	N	N	FS	
Relief mapping	Normal + D 3+1 o 4	Modificar la normal; descartar fragments	$(s+d_s, t+d_t)$	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	

Bump mapping/Normal mapping

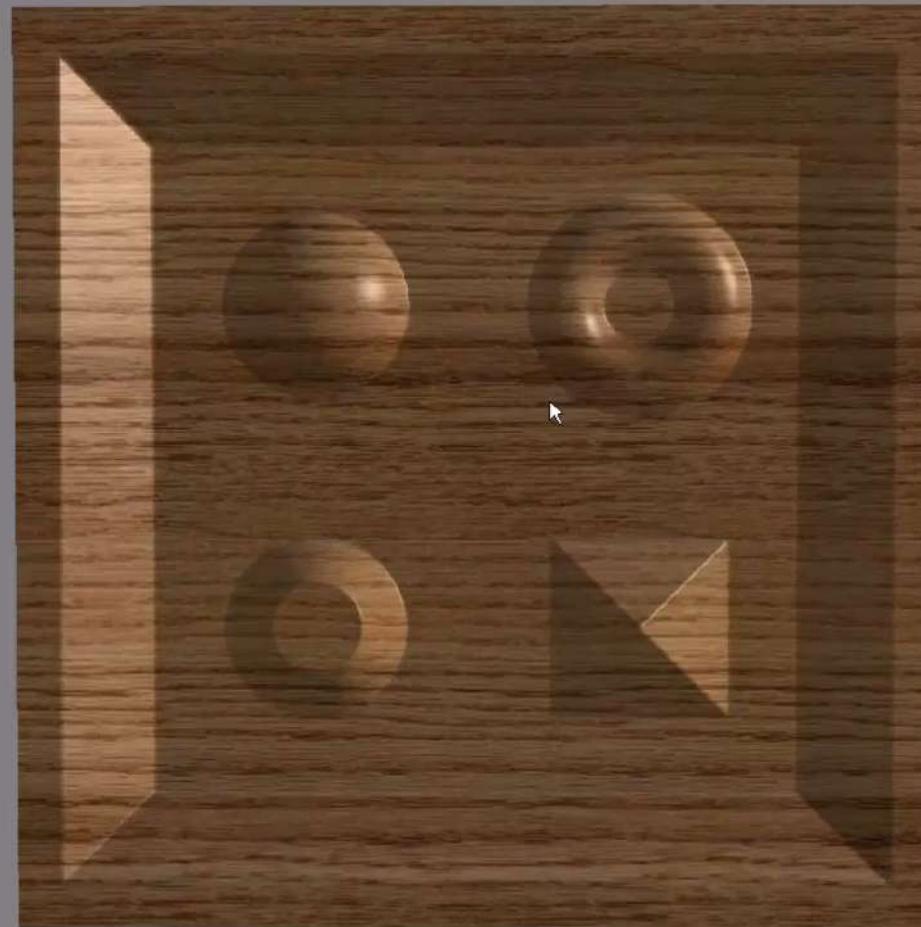


Bump mapping/Normal mapping

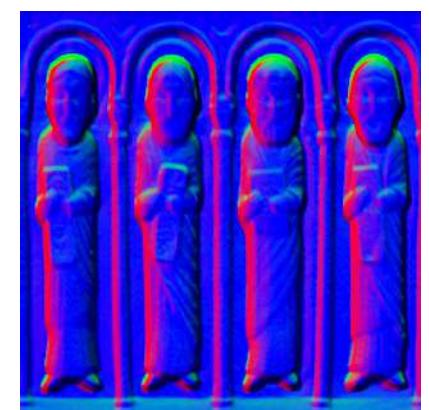
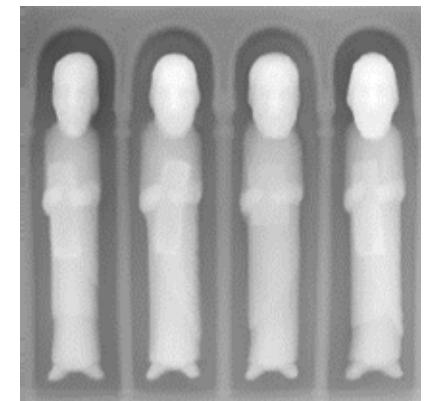
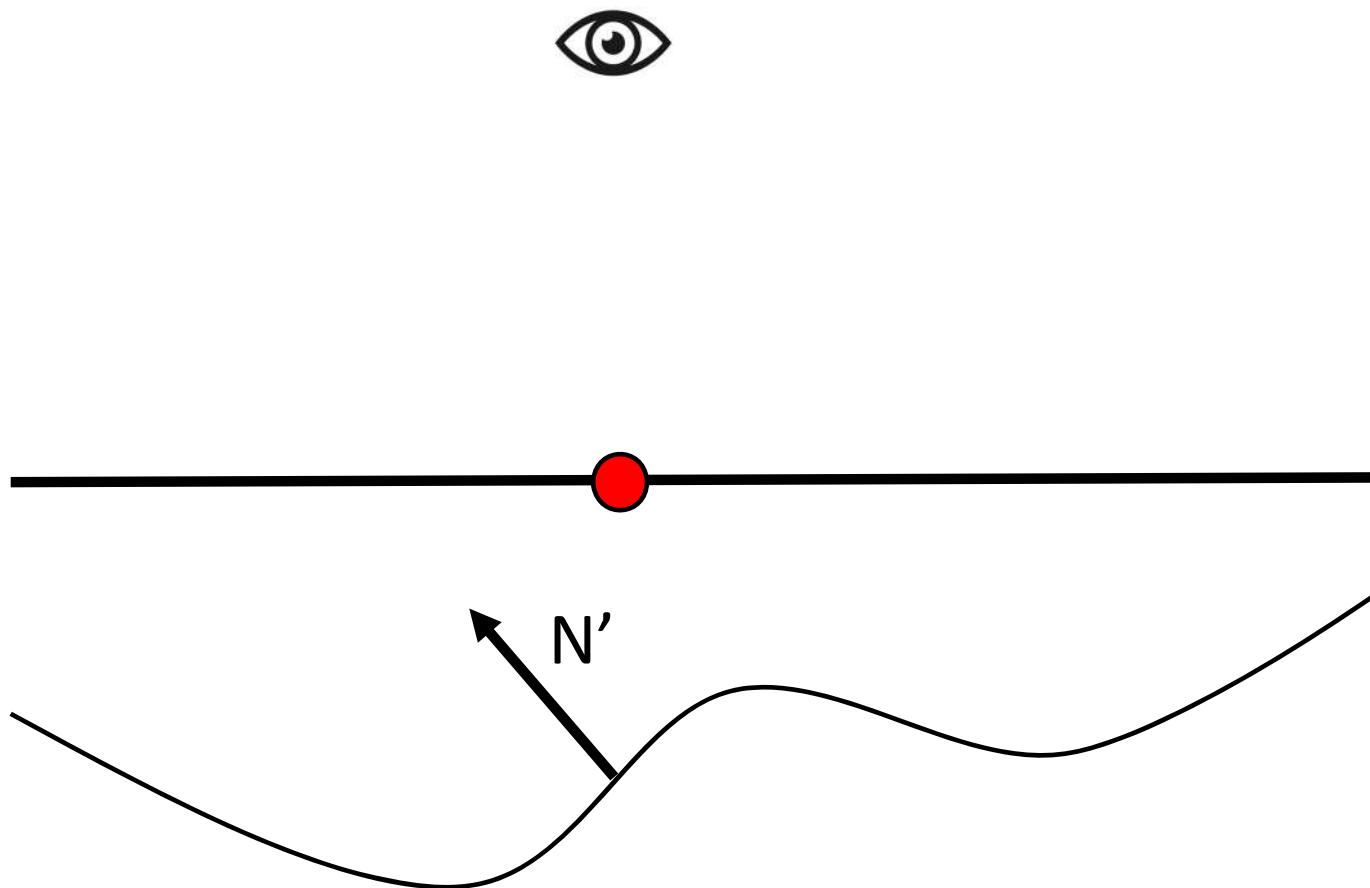


Relief Mapping
File View Render About...

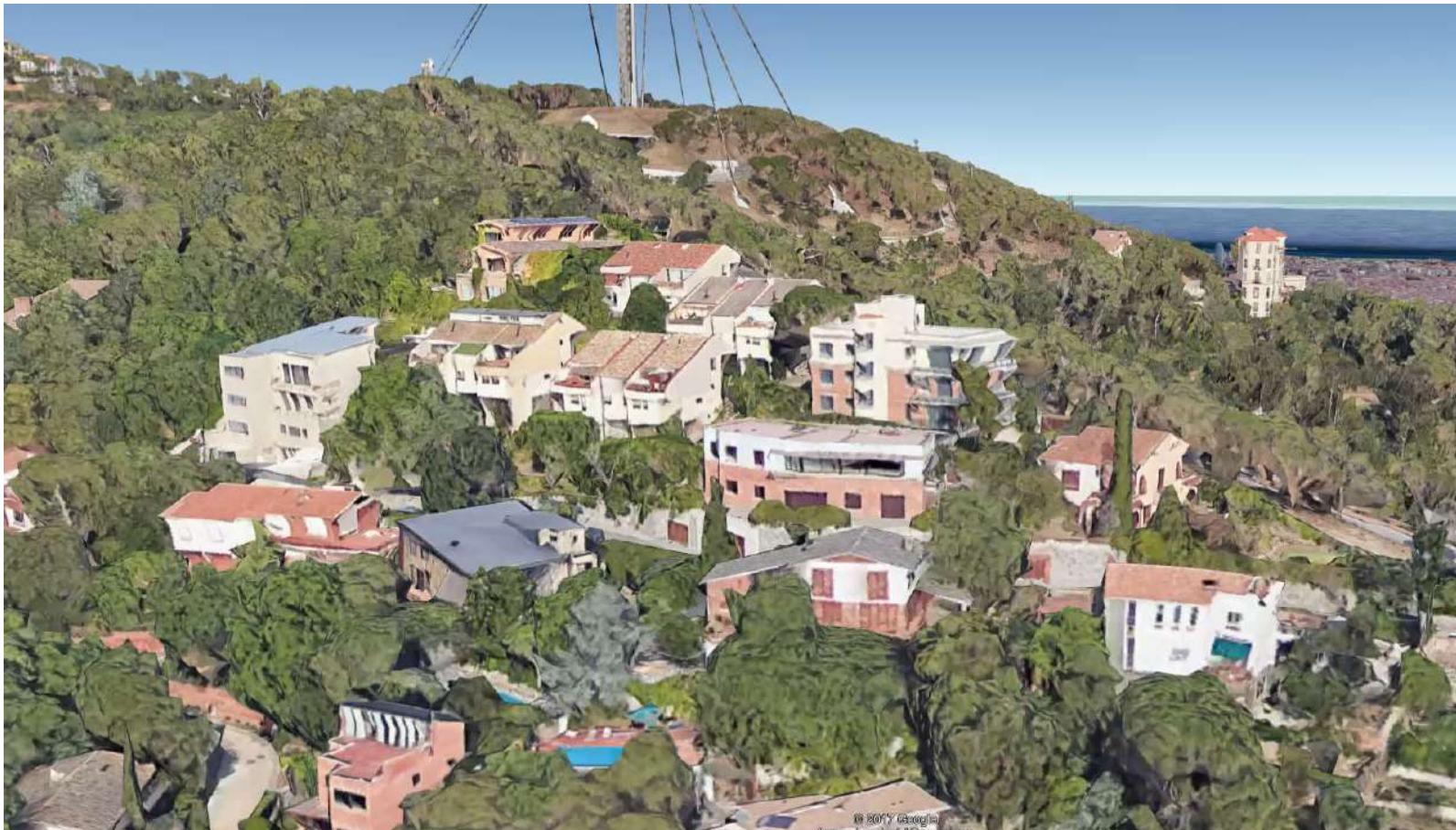
Normal Mapping
Border Clamp
Depth Bias

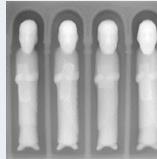
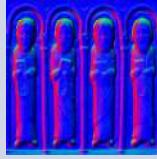


Problema Bump mapping/Normal mapping

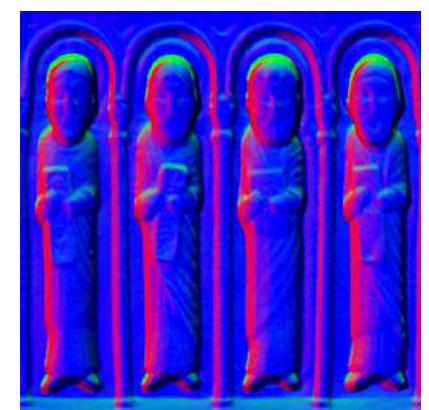
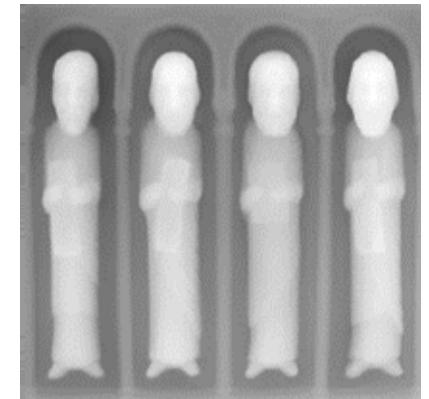
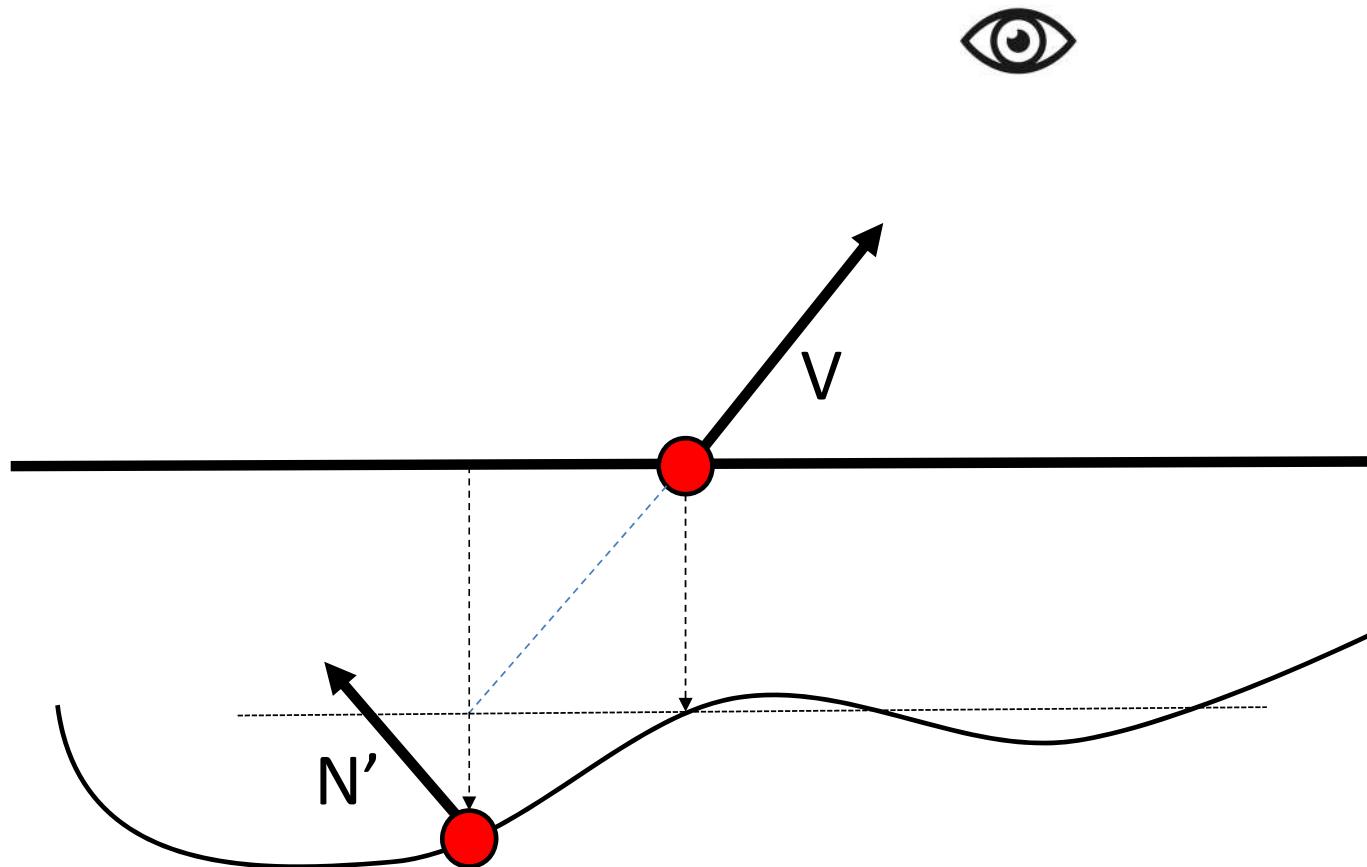


View-motion parallax

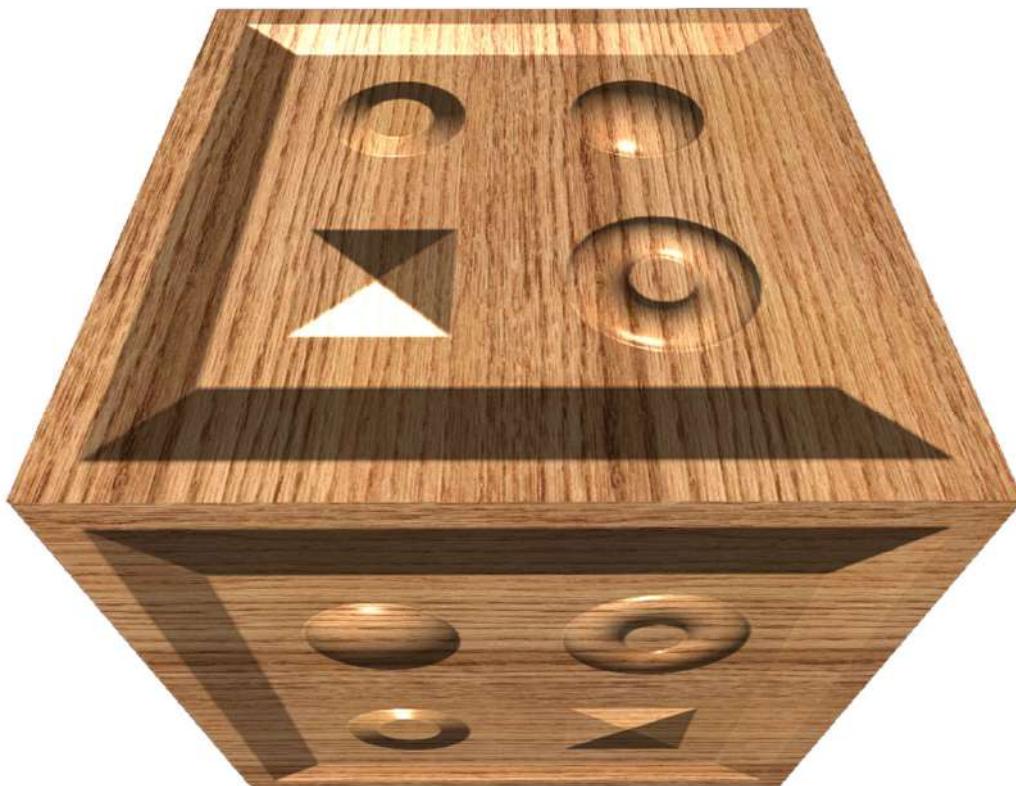


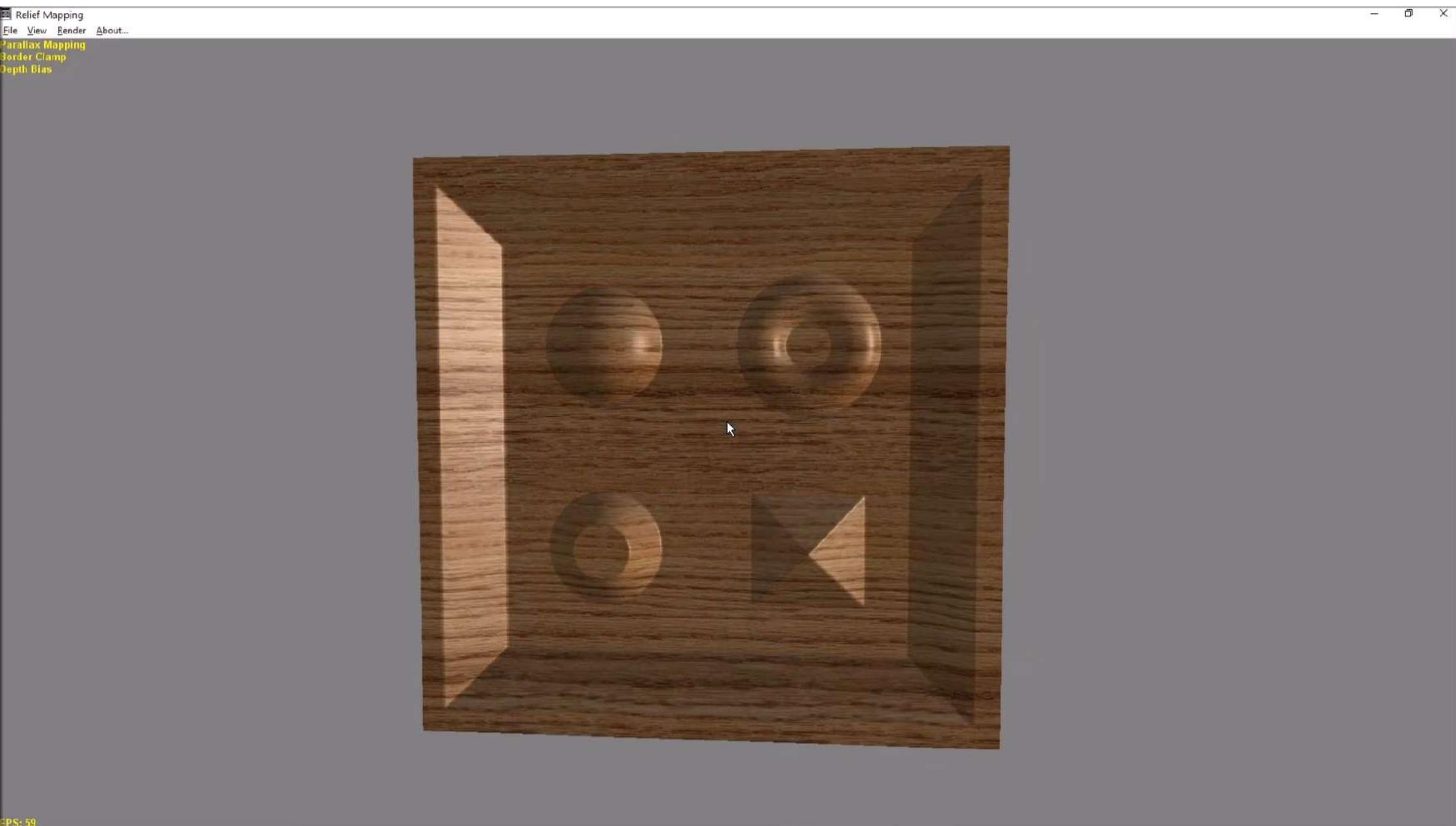
Tècnica	Info a la textura	Ús de la textura	Coords de textura	View parallax	Self-occlusion	Detailed silhouette	On s'aplica	
Color mapping	RGB 3	Kd del material	(s,t)	-	-	-	FS	
Bump mapping	D 1		Modificar la normal	(s,t)	N	N	N	FS
Normal mapping	Normal 3		Modificar la normal	(s,t)	N	N	N	FS
Parallax mapping	Normal + D 3+1 o 4	Modificar la normal	$(s+d_s, t+d_t)$	S	N	N	FS	
Relief mapping	Normal + D 3+1 o 4	Modificar la normal; descartar fragments	$(s+d_s, t+d_t)$	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	

Parallax mapping

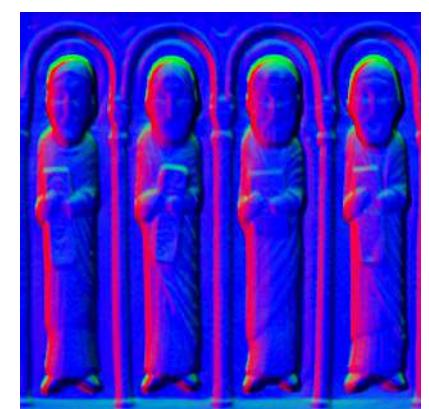
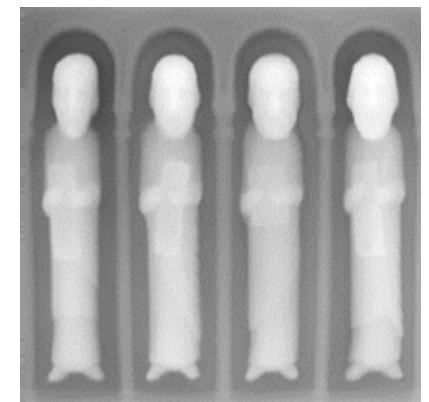
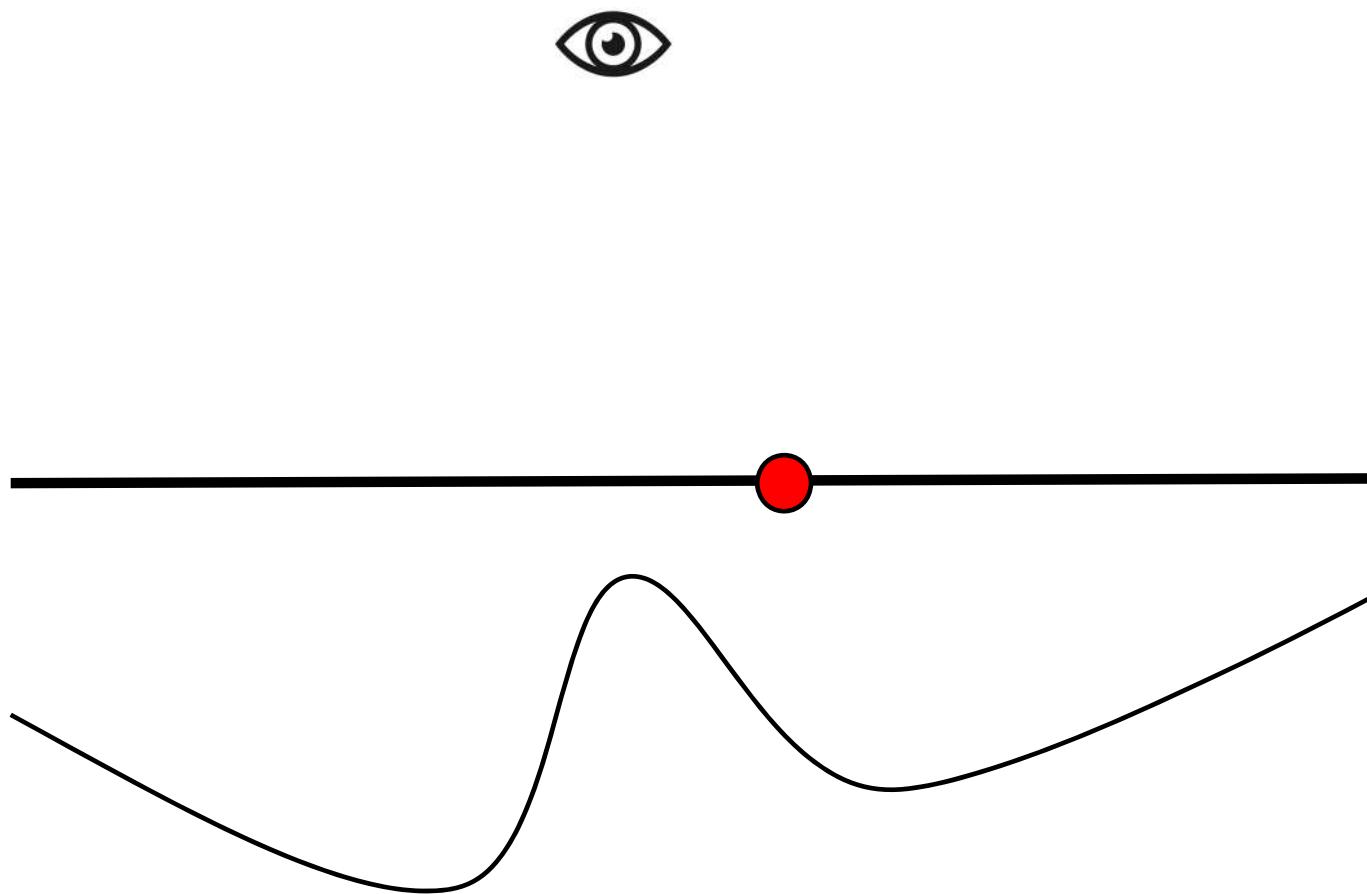


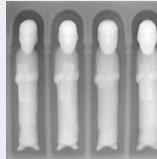
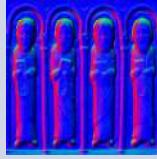
Parallax mapping



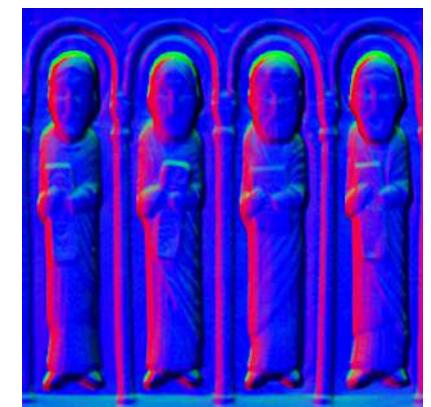
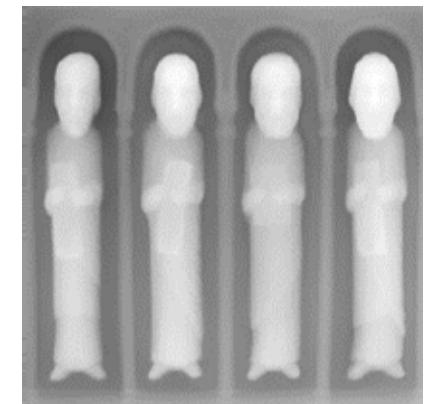
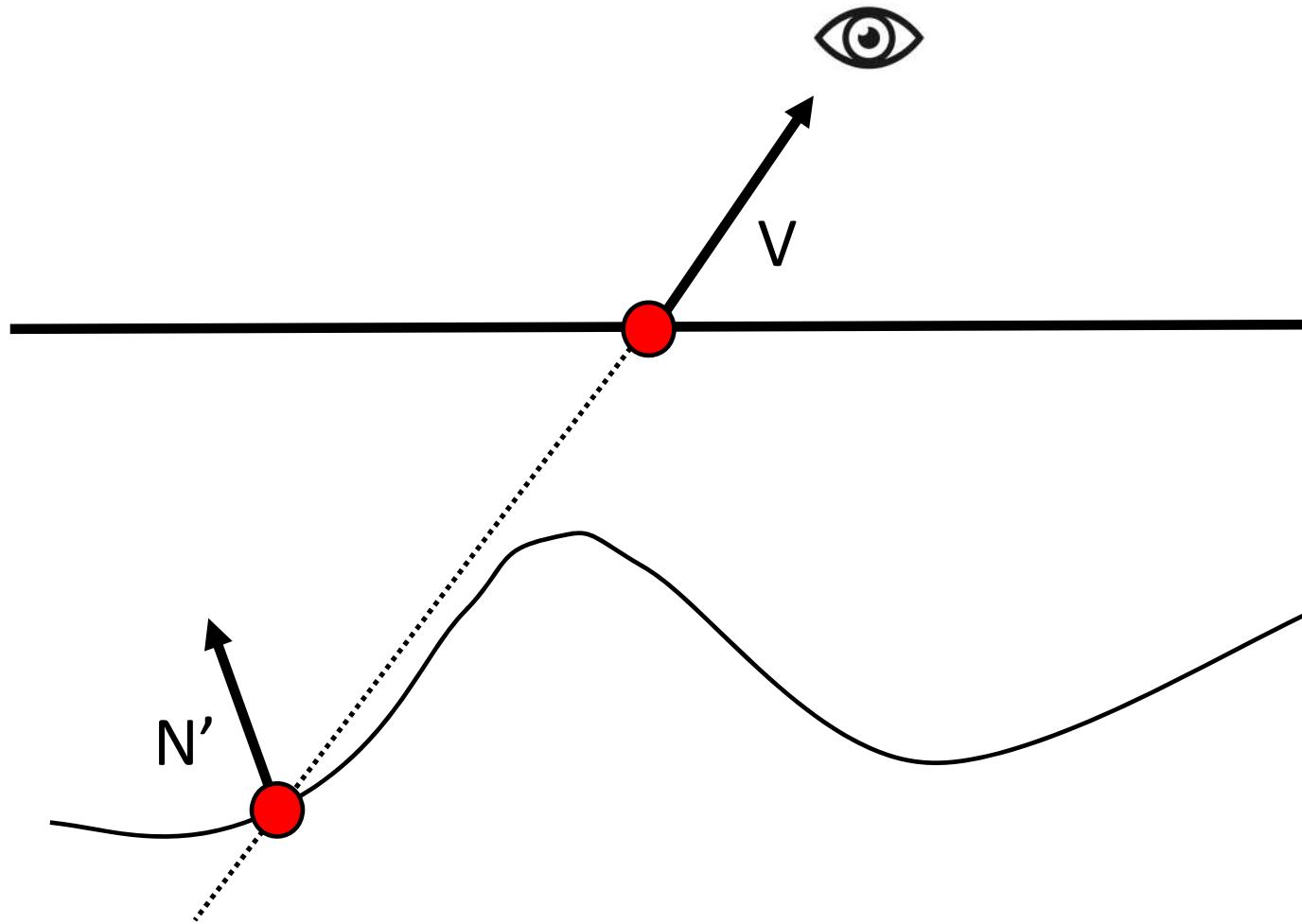


Problema Parallax mapping

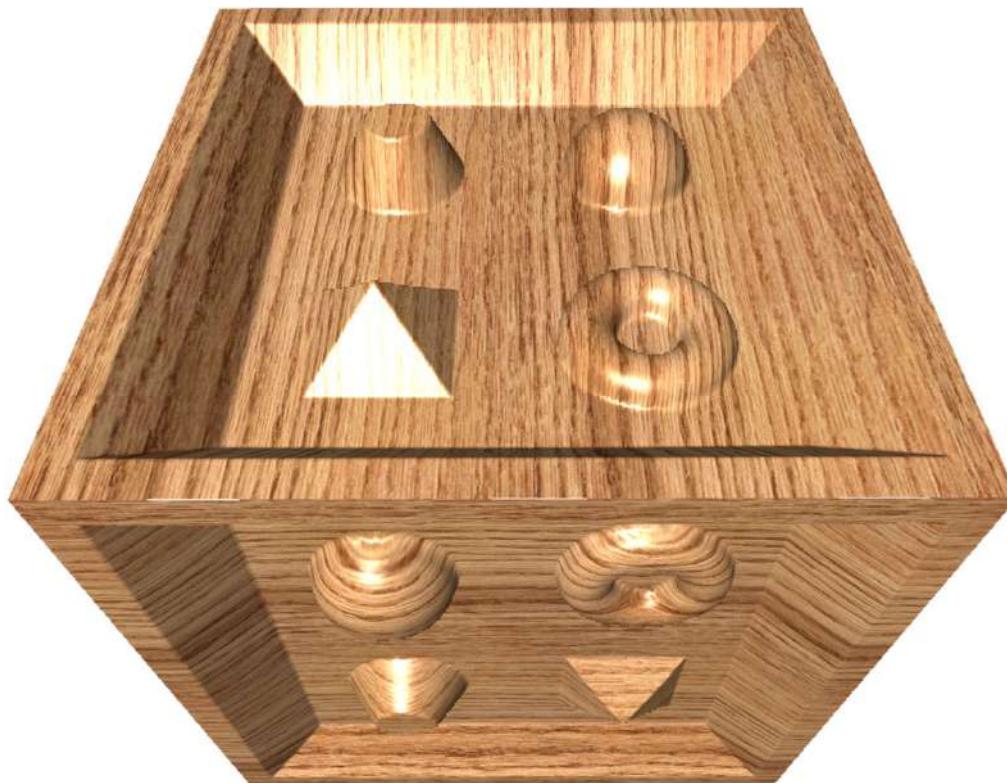


Tècnica	Info a la textura	Ús de la textura	Coords de textura	View parallax	Self-occlusion	Detailed silhouette	On s'aplica	
Color mapping	RGB 3	Kd del material	(s,t)	-	-	-	FS	
Bump mapping	D 1		Modificar la normal	(s,t)	N	N	N	FS
Normal mapping	Normal 3		Modificar la normal	(s,t)	N	N	N	FS
Parallax mapping	Normal + D 3+1 o 4	Modificar la normal	$(s+d_s, t+d_t)$	S	N	N	FS	
Relief mapping	Normal + D 3+1 o 4	Modificar la normal; descartar fragments	$(s+d_s, t+d_t)$	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	

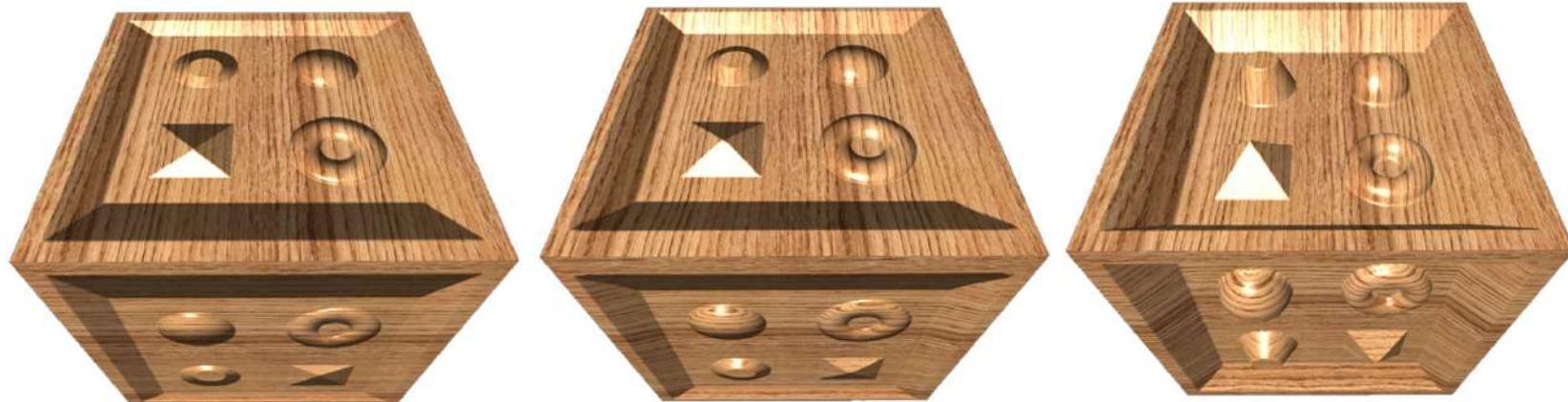
Relief mapping

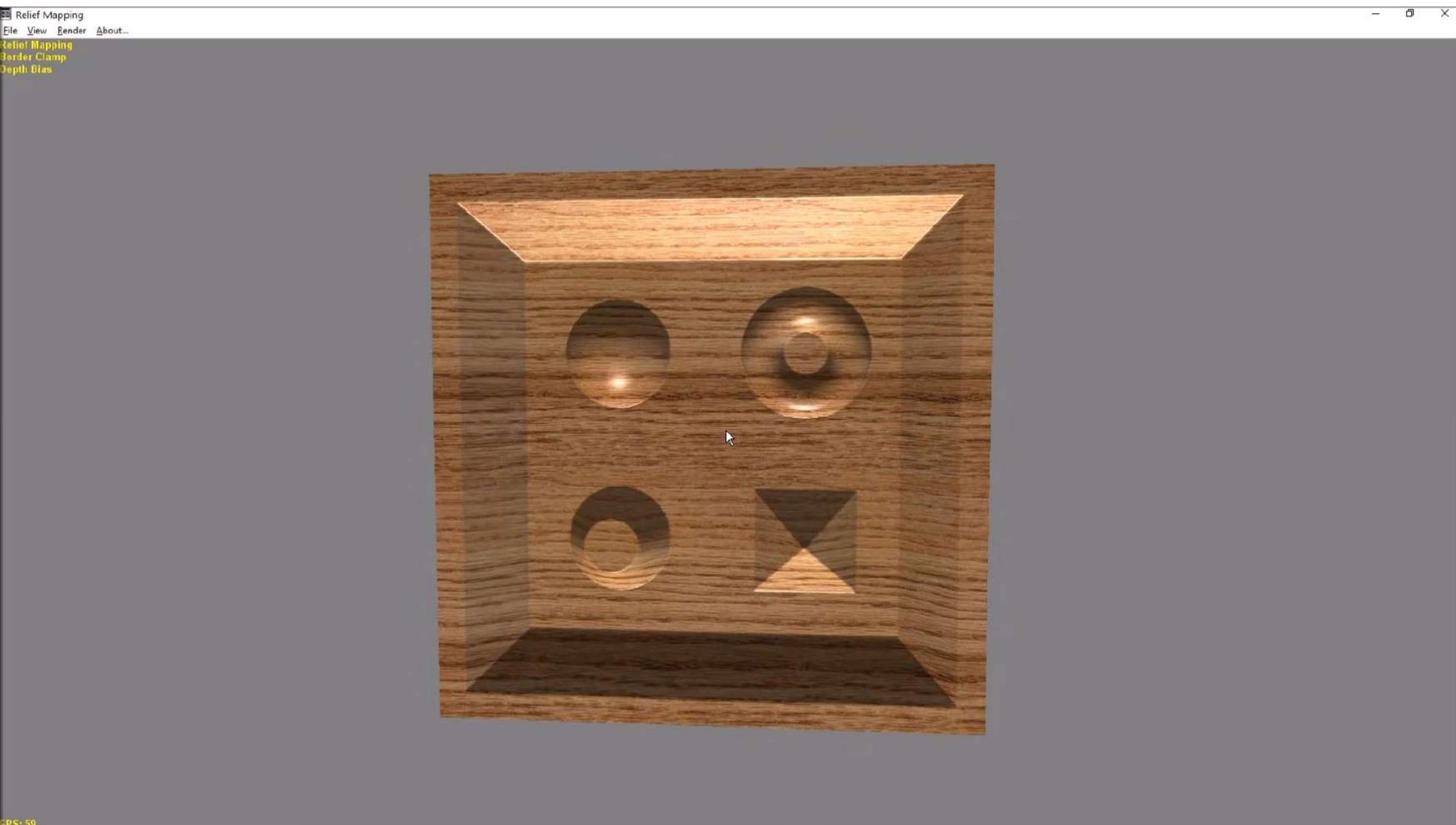


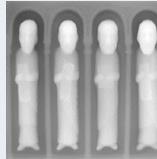
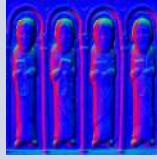
Relief mapping



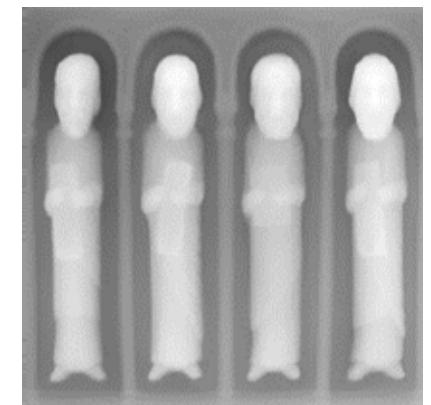
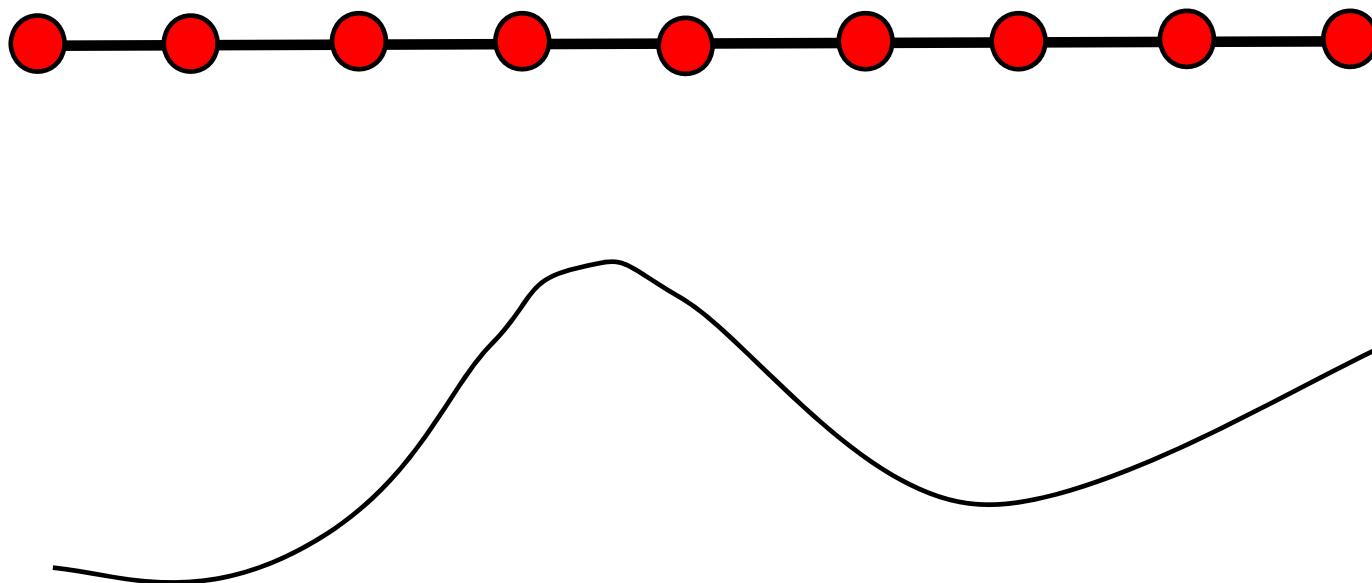
Comparació





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	$(s+d_s, t+d_t)$	S	N	N	FS	
Relief mapping	Normal + D 3+1 o 4	Modificar la normal; descartar fragments	$(s+d_s, t+d_t)$	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	

Displacement mapping



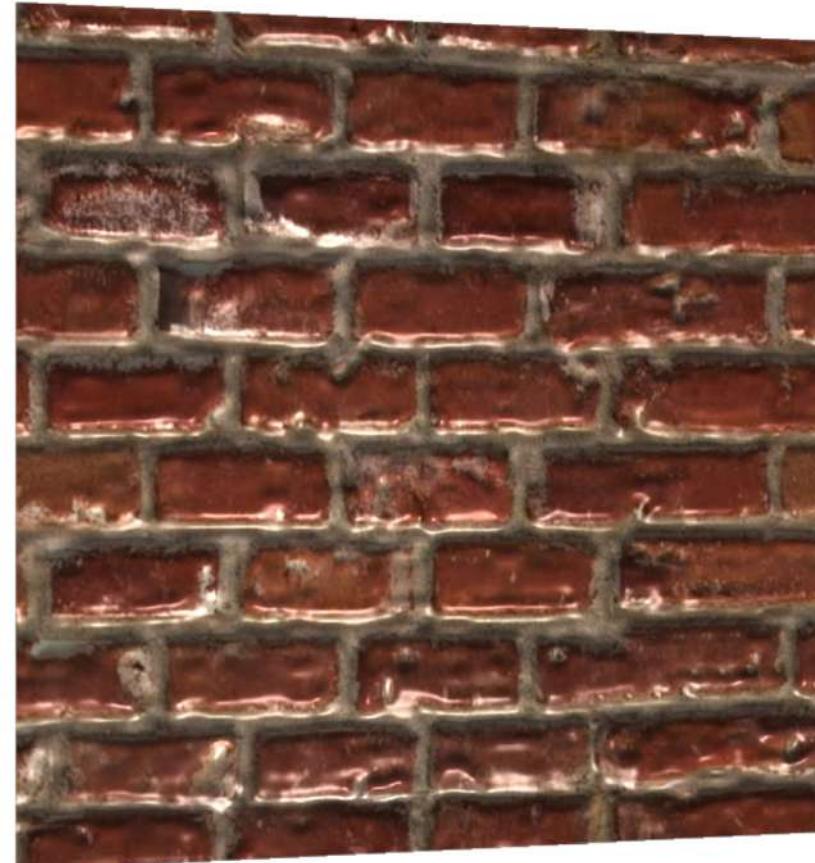
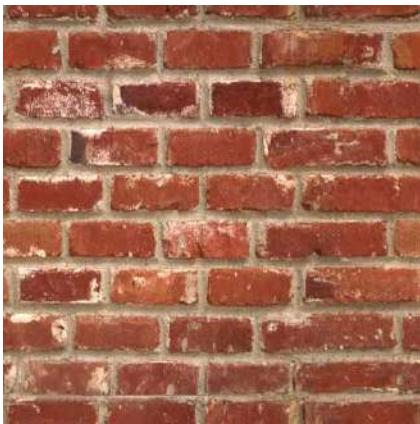
Comparació



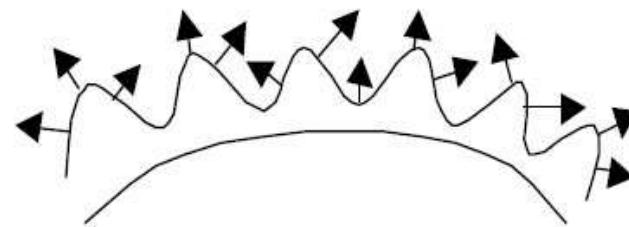
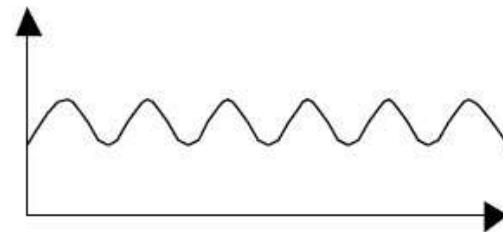
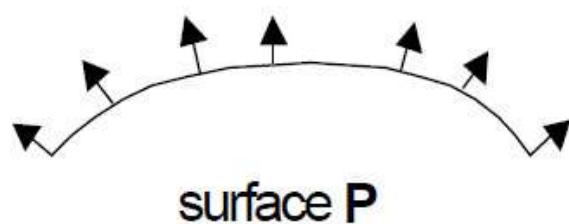
Mark Kilgard. A Practical and Robust Bump-mapping Technique for Today's GPUs. GDC 2000

BUMP MAPPING

Bump mapping

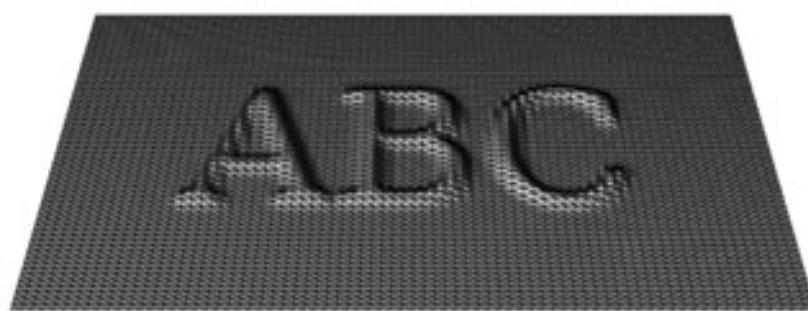


Elements bàsics

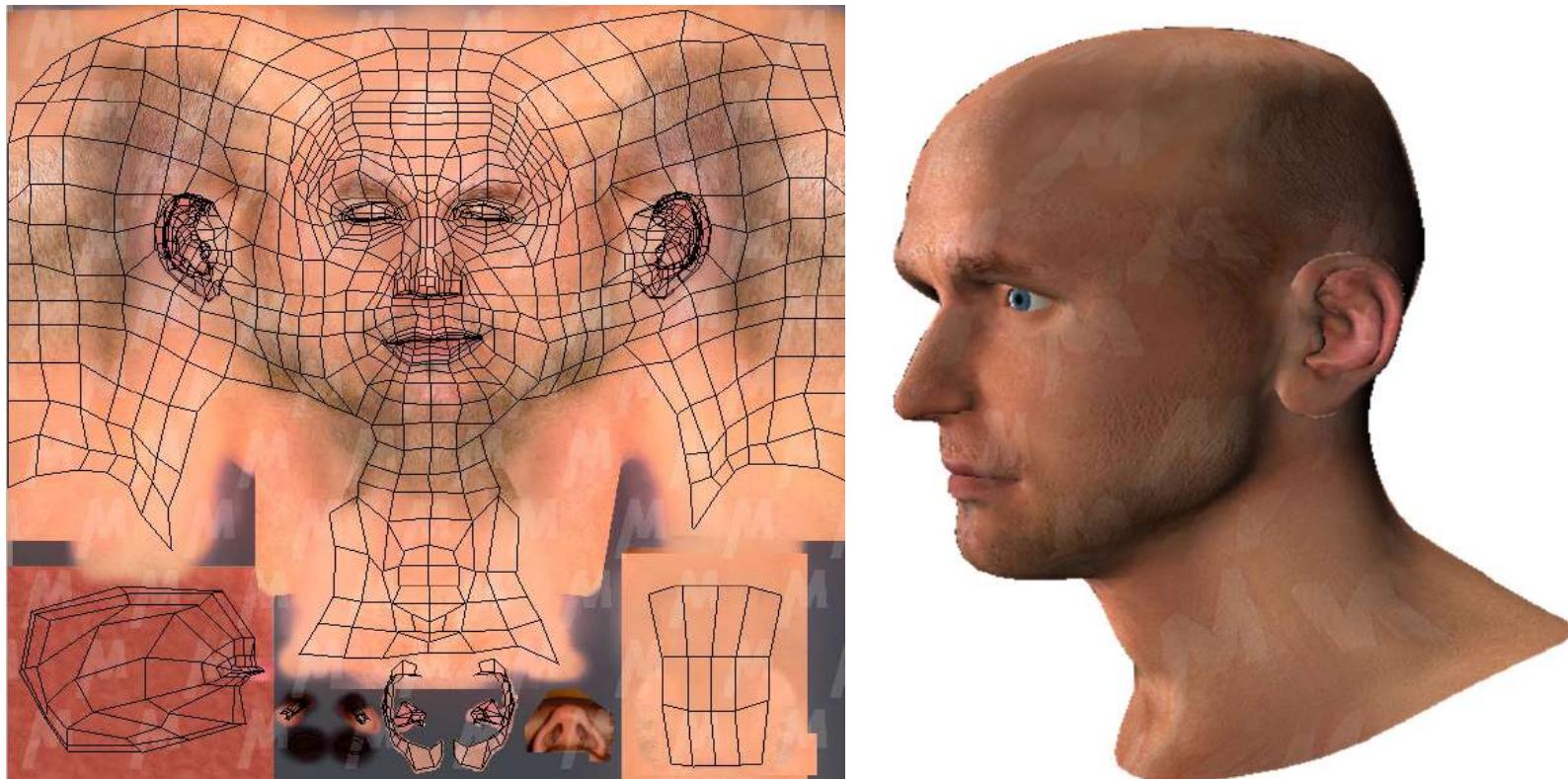


perturbed surface P'

Height field



Malla amb coordenades de textura



Eqüacions

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

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

$$\mathbf{P}'(u, v) = \mathbf{P}(u, v) + F(u, v) \frac{\mathbf{N}(u, v)}{|\mathbf{N}(u, v)|}$$

Eqüacions

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

$$\mathbf{N}' = \left(\frac{\partial \mathbf{P}}{\partial u} + \frac{\partial F}{\partial u} \left(\frac{\mathbf{N}}{|\mathbf{N}|} \right) \right) \times \left(\frac{\partial \mathbf{P}}{\partial v} + \frac{\partial F}{\partial v} \left(\frac{\mathbf{N}}{|\mathbf{N}|} \right) \right) \quad \frac{\partial \mathbf{P}'}{\partial u} = \frac{\partial \mathbf{P}}{\partial u} + \frac{\partial F}{\partial u} \left(\frac{\mathbf{N}}{|\mathbf{N}|} \right) + F \left(\frac{\partial \frac{\mathbf{N}}{|\mathbf{N}|}}{\partial u} \right)$$

$$\mathbf{N}' = \frac{\partial \mathbf{P}}{\partial u} \times \frac{\partial \mathbf{P}}{\partial v} + \frac{\frac{\partial F}{\partial u} \left(\mathbf{N} \times \frac{\partial \mathbf{P}}{\partial v} \right)}{|\mathbf{N}|} + \frac{\frac{\partial F}{\partial v} \left(\frac{\partial \mathbf{P}}{\partial u} \times \mathbf{N} \right)}{|\mathbf{N}|} + \frac{\frac{\partial F}{\partial u} \frac{\partial F}{\partial v} (\mathbf{N} \times \mathbf{N})}{|\mathbf{N}|^2}$$

$$\mathbf{N}' = \mathbf{N} + \frac{\frac{\partial F}{\partial u} \left(\mathbf{N} \times \frac{\partial \mathbf{P}}{\partial v} \right) - \frac{\partial F}{\partial v} \left(\mathbf{N} \times \frac{\partial \mathbf{P}}{\partial u} \right)}{|\mathbf{N}|}$$

Eqüacions - resum

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

$$\mathbf{P}'(u, v) = \mathbf{P}(u, v) + F(u, v) \frac{\mathbf{N}(u, v)}{|\mathbf{N}(u, v)|}$$

$$\mathbf{N}' = \mathbf{N} + \frac{\frac{\partial F}{\partial u} \left(\mathbf{N} \times \frac{\partial \mathbf{P}}{\partial v} \right) - \frac{\partial F}{\partial v} \left(\mathbf{N} \times \frac{\partial \mathbf{P}}{\partial u} \right)}{|\mathbf{N}|}$$

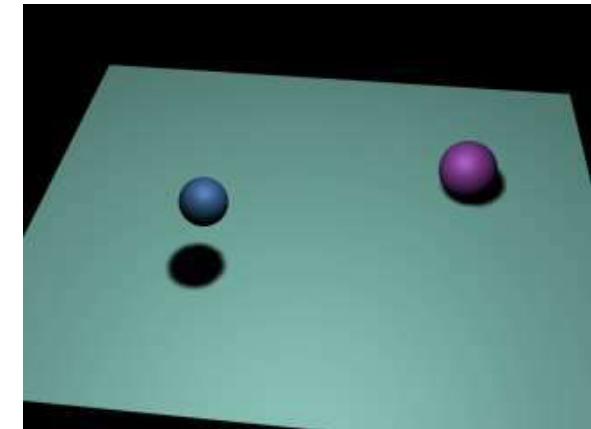
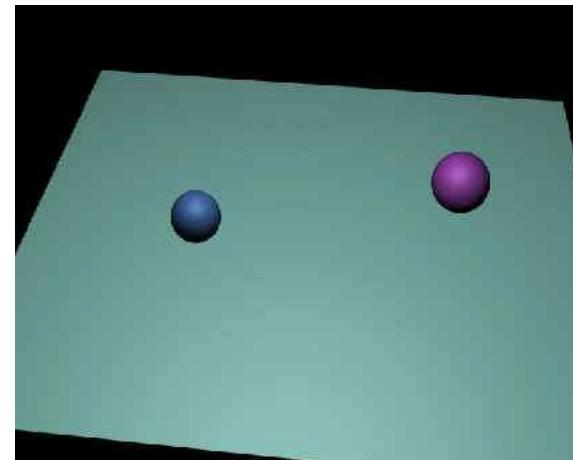
Simulació d'ombres

Carlos Andújar

Maig 2022

Avantatges de simular ombres

- Més realisme
- Indicació visual de profunditat

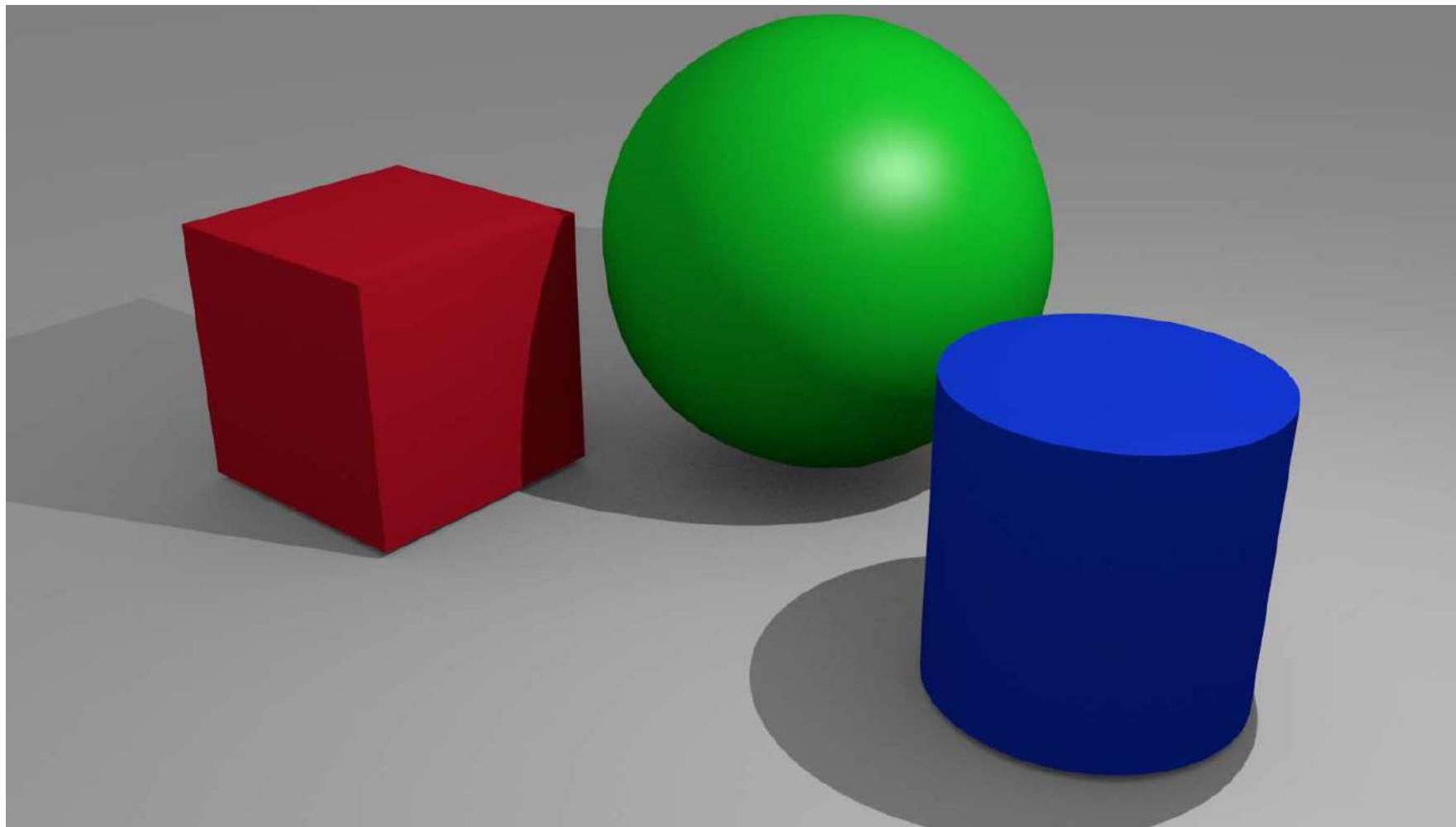


Avantatges de simularombres

- Informació adicional



Percepció d'ombres



ECVP

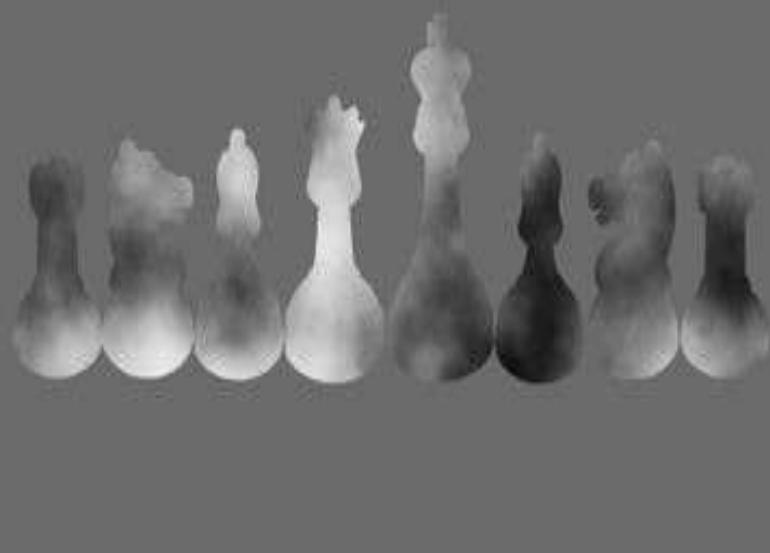


2005

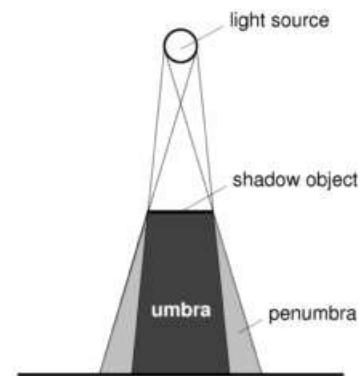
ECVP



2005

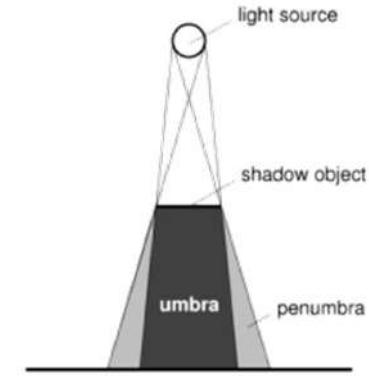


Umbra i penumbra



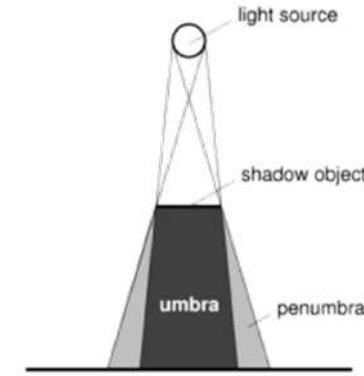
Propietats

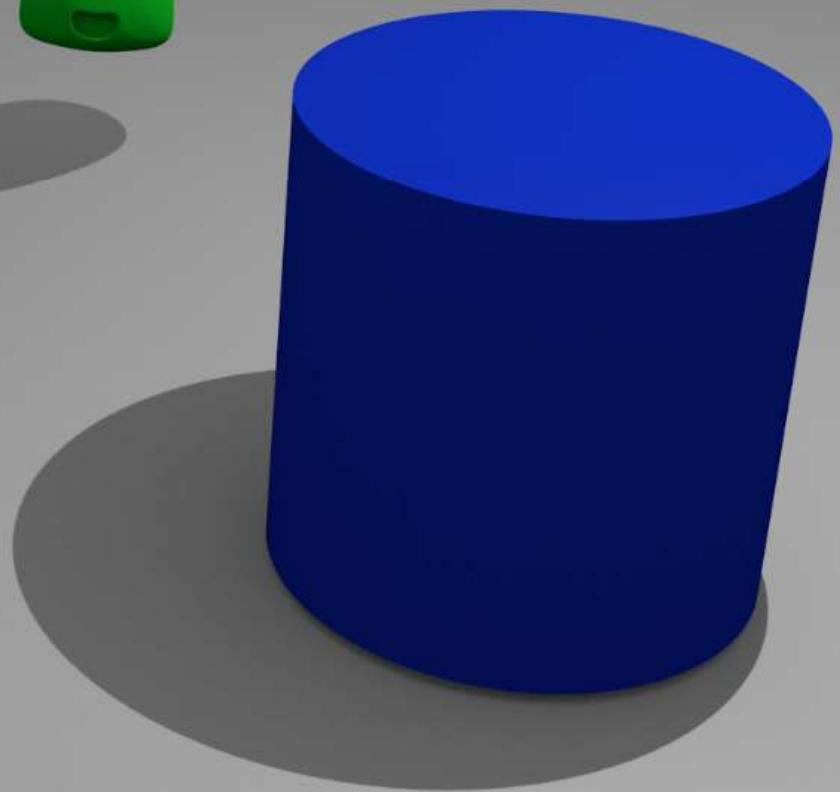
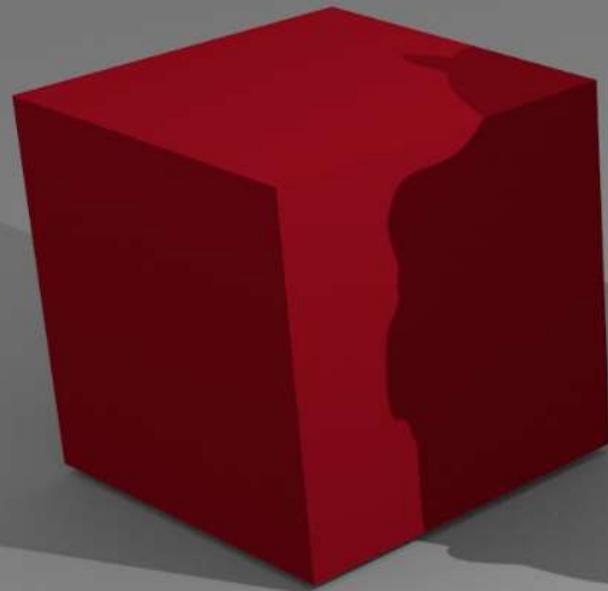
- Si la font de llum és puntual →
- Si augmenta la mida de la font de llum...
- Si apropiem oclusor i receptor...

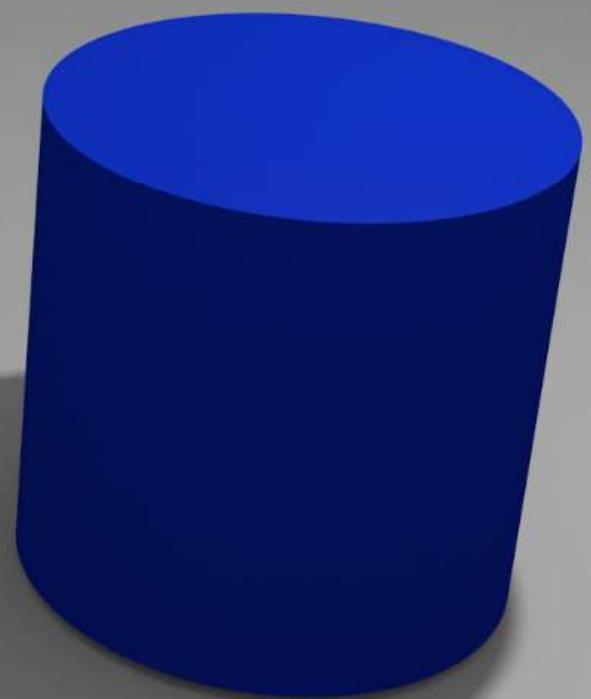
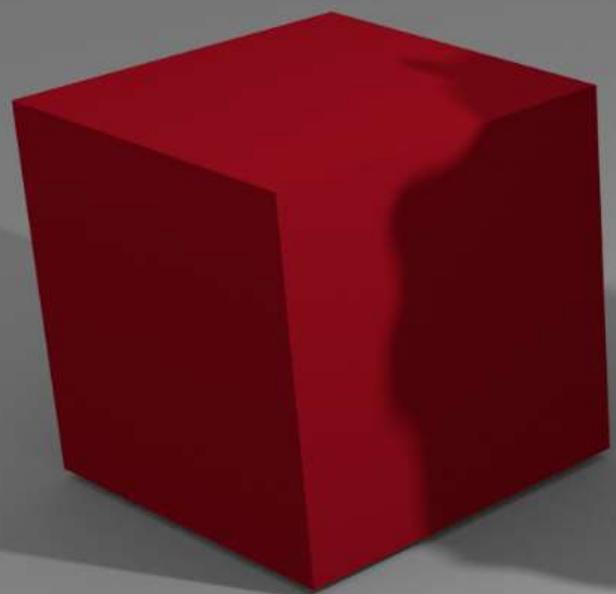


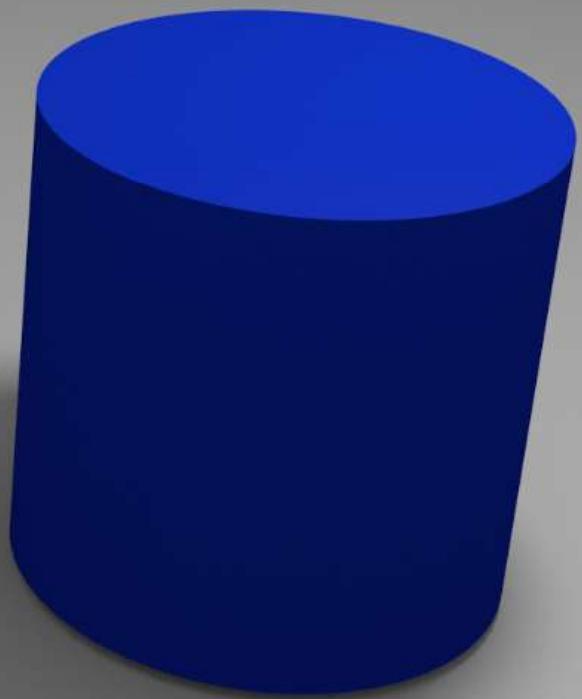
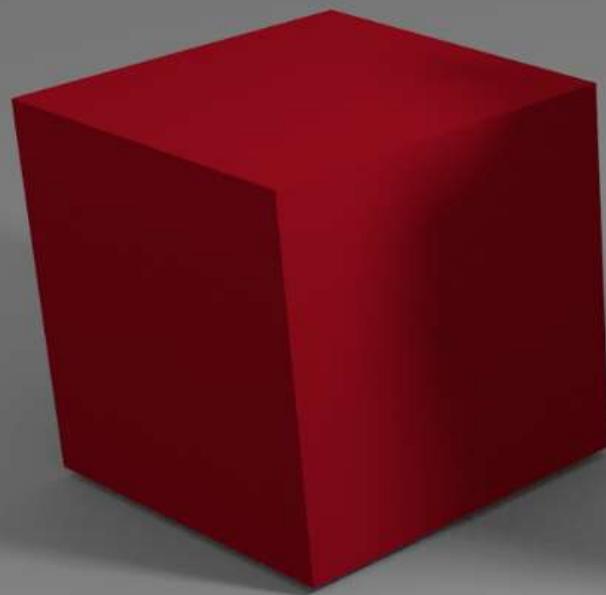
Propietats

- Si la font de 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 apropiem oclusor i receptor...
 - Disminueix la penombra

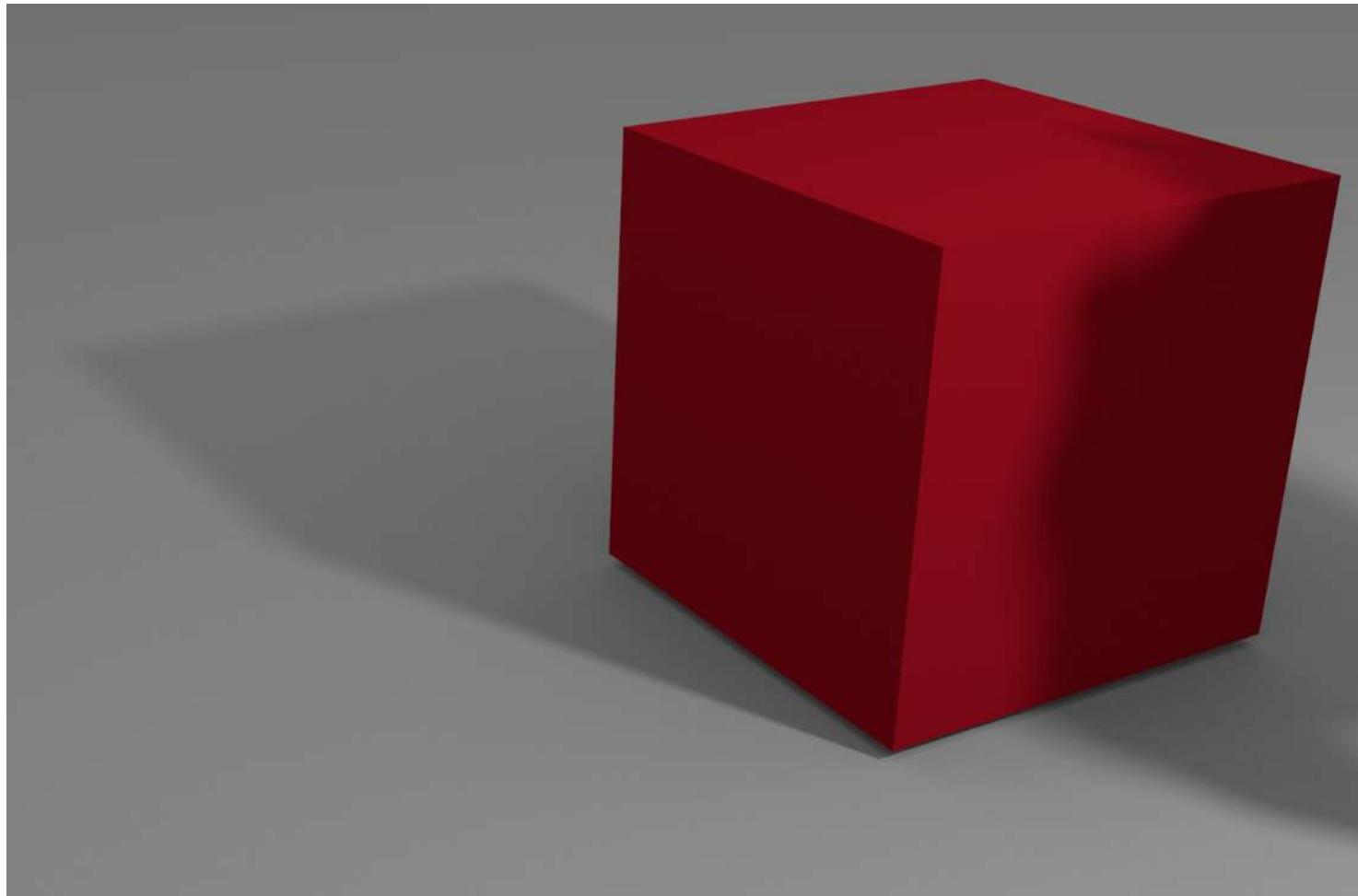






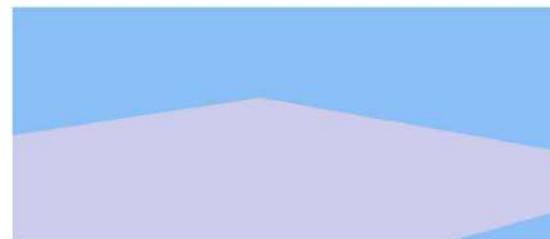


Propietats

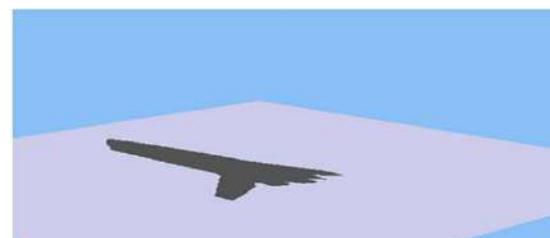


Ombres per projecció (un pla)

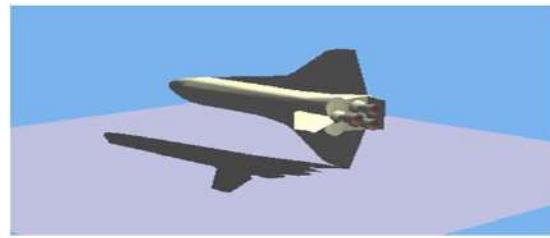
// 1. Dibuixar **receptor**



// 2. Dibuixar l'**oclusor** projectat (ombra)

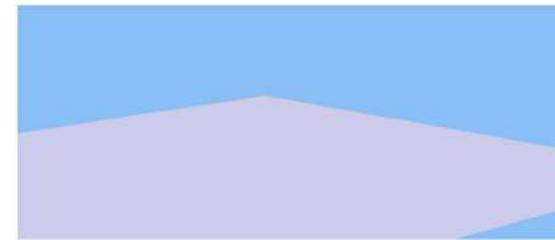


// 3. Dibuixar **oclusor**

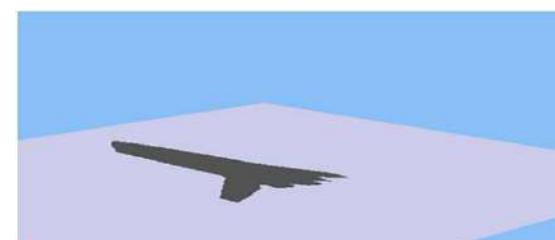


Ombres per projecció (un pla)

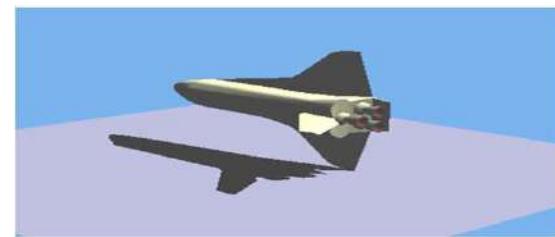
```
// 1. Dibuixar receptor  
dibuixa(receptor)
```



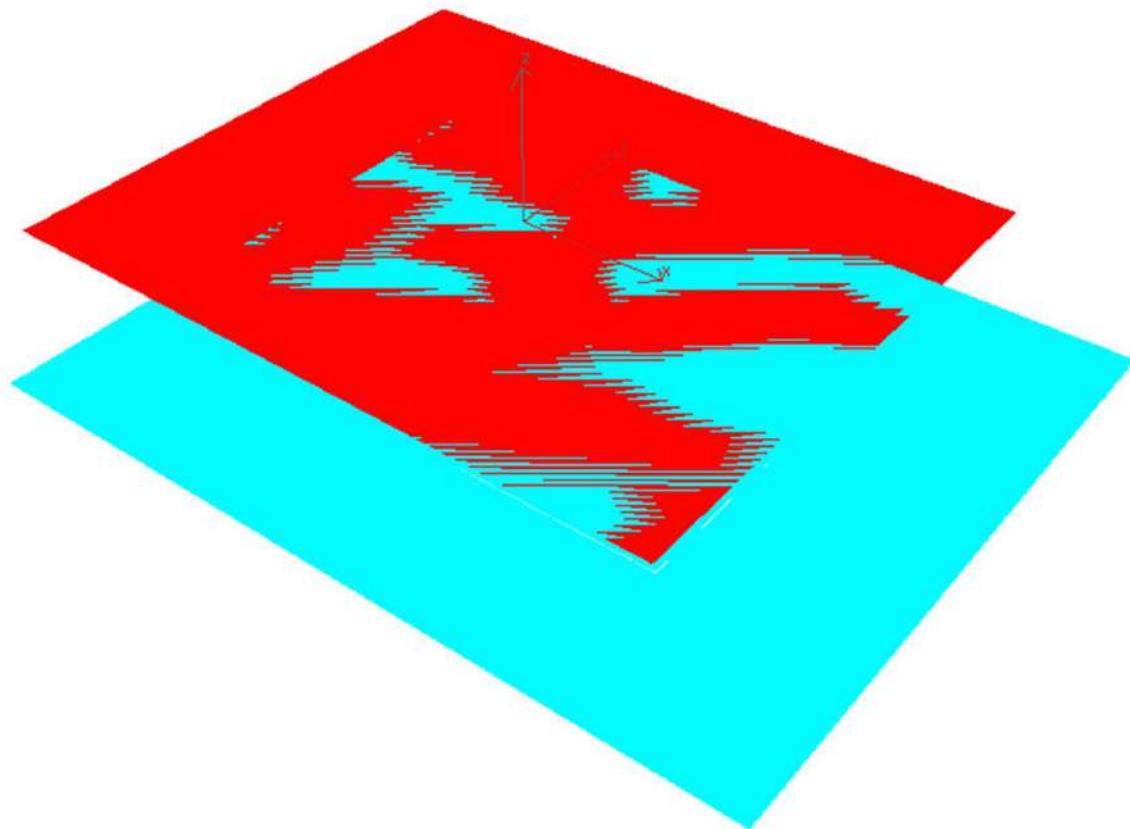
```
// 2. Dibuixar l'oclusor projectat (ombra)  
glDisable (GL_LIGHTING);  
glDisable (GL_DEPTH_TEST);  
glMatrixMode (GL_MODELVIEW);  
glPushMatrix ();  
glMultMatrixf (MatriuProjeccio);  
dibuixa (oclusor);  
glPopMatrix();
```



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



Z-fighting



Evitar problemes de z-fighting

`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

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

on

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

→ El paràmetre *units* permet introduir un **offset constant**

Evitar problemes de z-fighting

`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

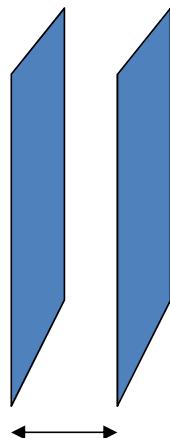
$$\partial z = \max(\partial z / \partial x, \partial z / \partial y)$$

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

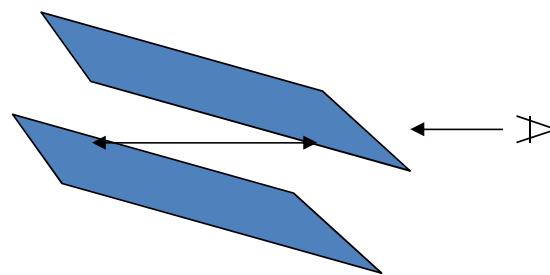
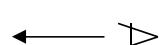
- El paràmetre *factor* permet introduir un **offset variable** (depén de la inclinació del polígon)
- El paràmetre *units* permet introduir un **offset constant**

Evitar problemes de z-fighting

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



Offset = $r \cdot \text{units}$ ($dz=0$)



Offset = $3 \cdot \text{factor} + r \cdot \text{units}$ ($dz=3$)

Evitar problemes de z-fighting

Valors típics: `glPolygonOffset(1, 1);`

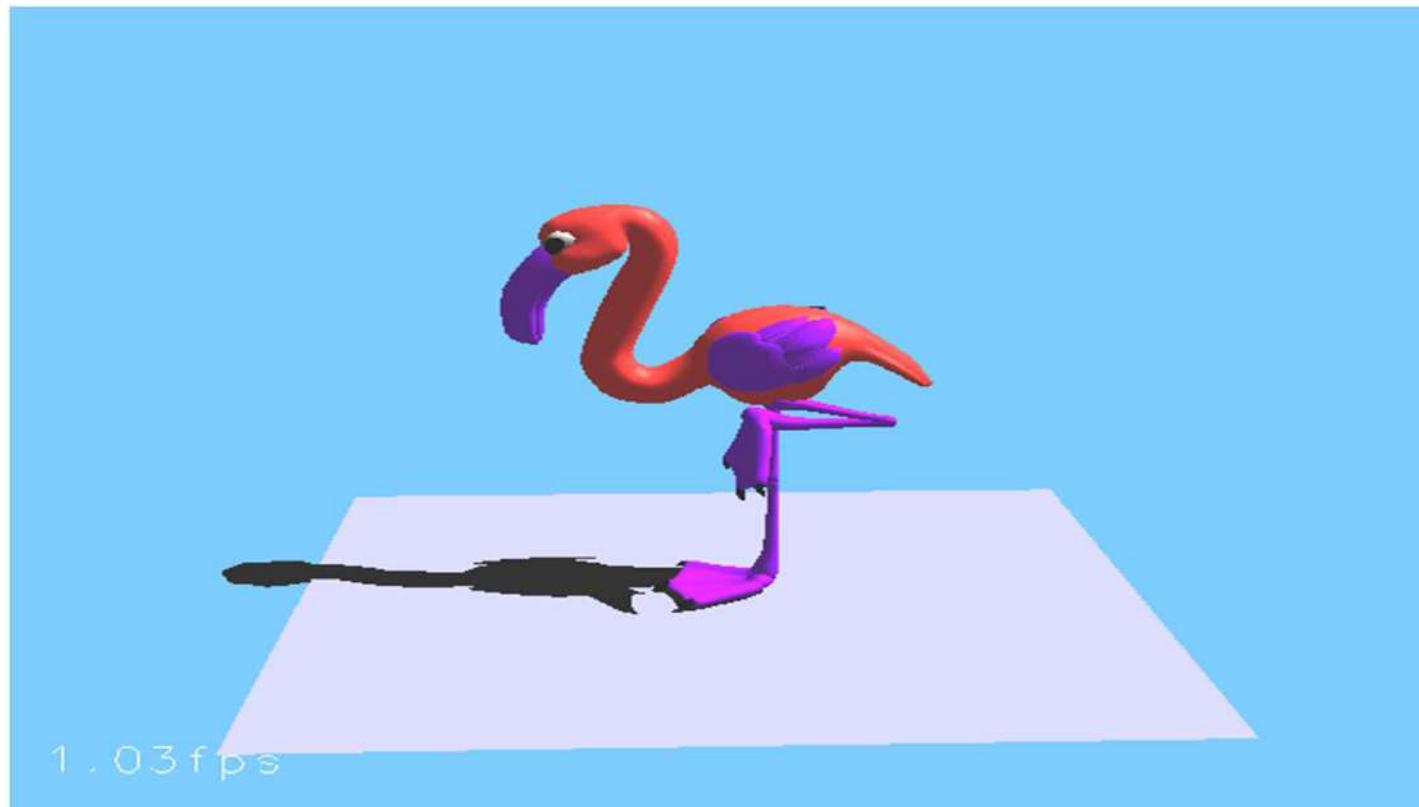
Offset positiu → increment de la z (en window coordinates) → es calcula la z com si estigués més lluny

Ombres per projecció (un pla)

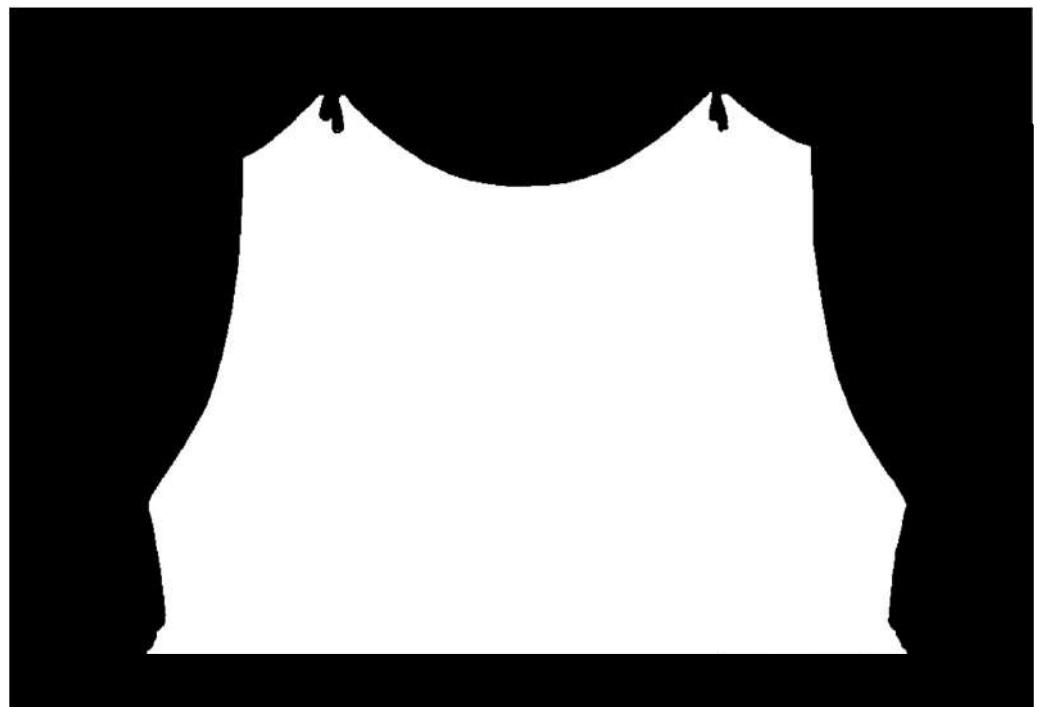
```
void CastShadows::paintGL(...) {  
    // 1. Dibuixar receptor  
    phongShader->bind();  
    drawReceiver();  
  
    // 2. Dibuixar l'oclusor projectat (ombra)  
    phongShader->setUniform("shadow", true);  
    phongShader->setUniform("modelMatrix", ...);  
    glEnable(GL_POLYGON_OFFSET_FILL);  
    glPolygonOffset(-1, -1);  
    drawOccluder();  
  
    // 3. Dibuixar oclusor  
    phongShader->setUniform("shadow", false);  
    phongShader->setUniform("modelMatrix", identity);  
    glDisable(GL_POLYGON_OFFSET_FILL);  
    drawOccluder();  
}
```



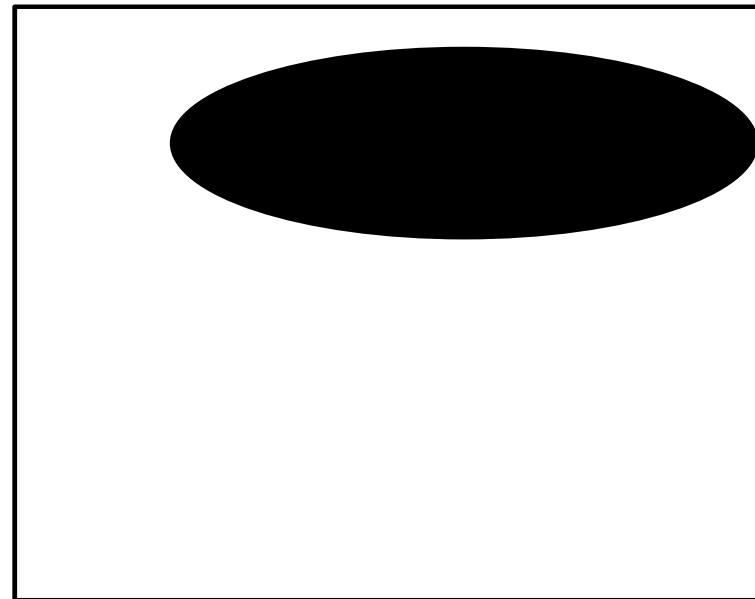
Ombres per projecció (un pla)



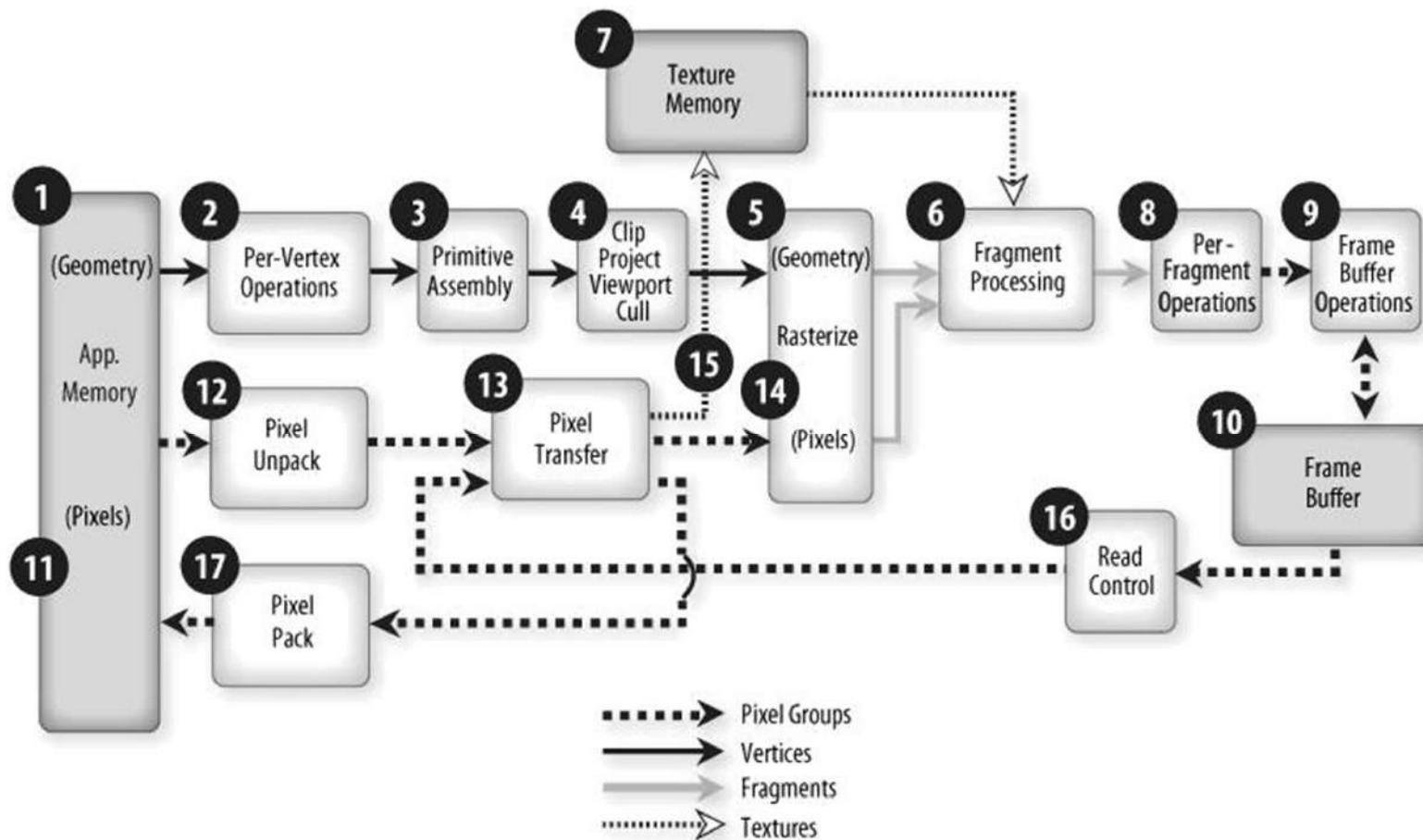
Stencil buffer



Stencil buffer



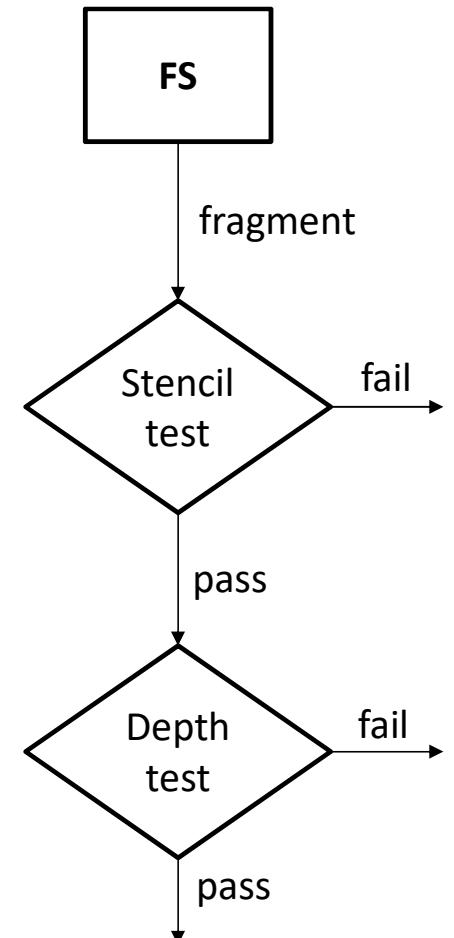
Pipeline OpenGL



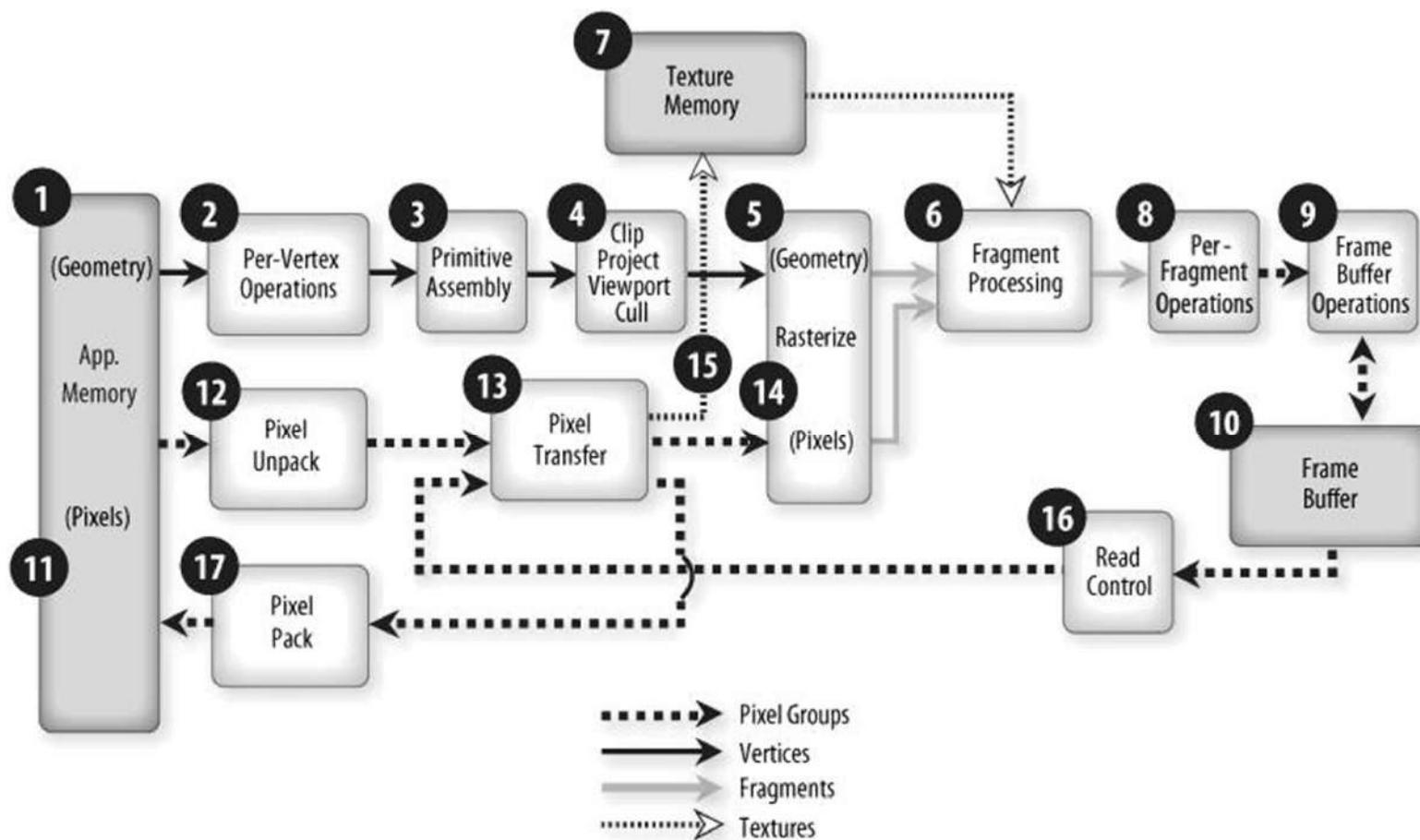
Pipeline OpenGL

8. Per-fragment operations (“raster operations”)

- Pixel ownership
- Scissor test
- Alpha test
- **Stencil test**
- **Depth test (test Z-buffer)**
- Blending
- Dithering
- Logical Ops (glLogicOp)



Pipeline OpenGL



Pipeline OpenGL

9. Frame buffer operations

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

Stencil buffer

El stencil buffer guarda, per cada pixel, un enter entre $0..2^n-1$.

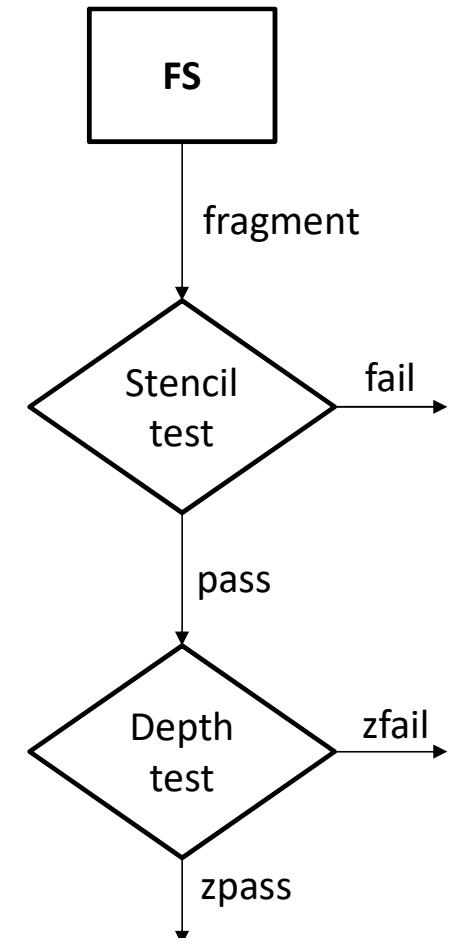
- Demanar una finestra OpenGL amb stencil:
 - QOpenGLformat f;
 - f.setStencil(true);
 - QOpenGLformat::setDefaultFormat(f);
- Obtenir el núm. de bits del stencil:
 - glGetIntegerv(GL_STENCIL_BITS, &nbits);
- Esborrar stencil (no li afecta glStencilFunc(), sí glStencilMask):
 - glClearStencil(0);
 - glClear(GL_STENCIL_BUFFER_BIT);

Stencil buffer

- 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 refència)

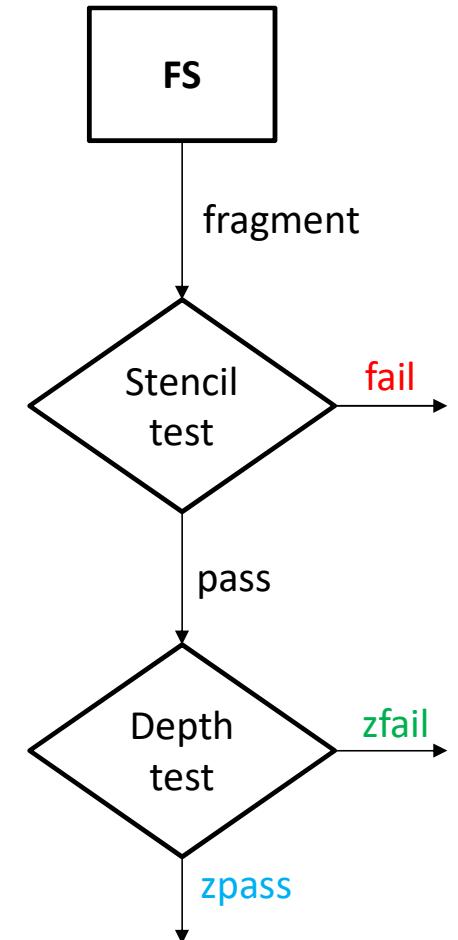
Stencil buffer

- 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 refència)



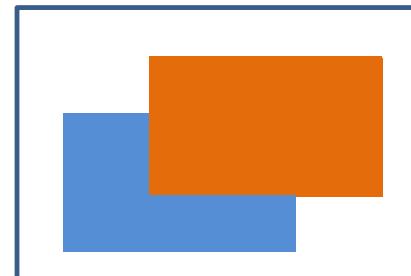
Stencil buffer

```
glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_ALWAYS, 1, 255);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
drawQuad()
```

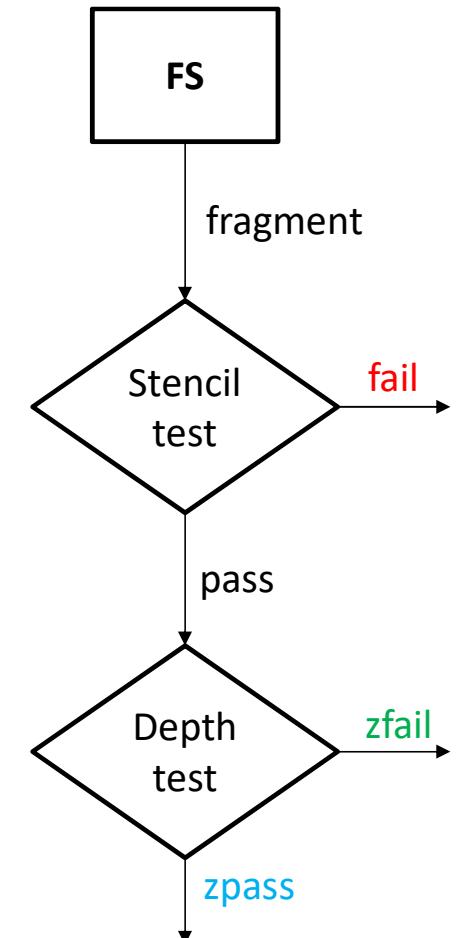


Stencil buffer

```
glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_ALWAYS, 1, 255);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
drawQuad()
```



```
glStencilFunc(GL_EQUAL, 0, 255);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
drawQuad();
```

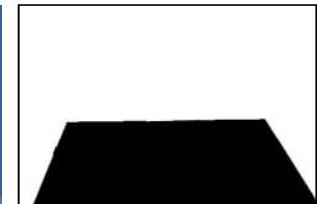


Targets

Ombres per projecció (amb stencil)

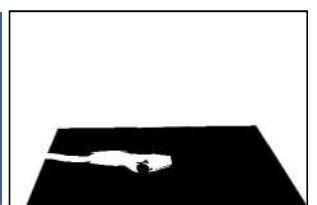
Color, Z, Stencil

// 1. Dibuixa el **receptor** al color buffer i al stencil buffer



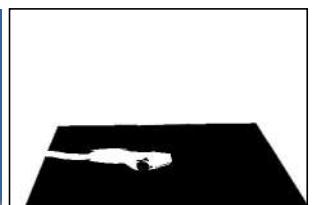
Stencil

// 2. Dibuixa **occlusor** per netejar l'stencil a les zones a l'ombra



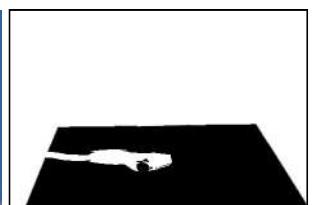
Color, Z

// 3. Dibuixa la part fosca del **receptor**



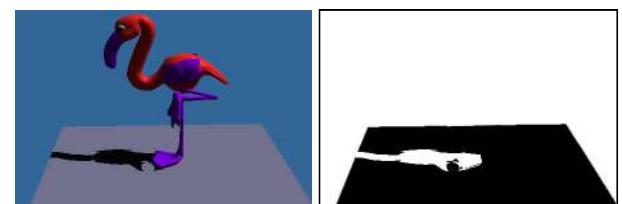
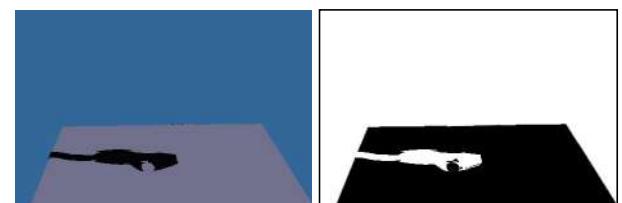
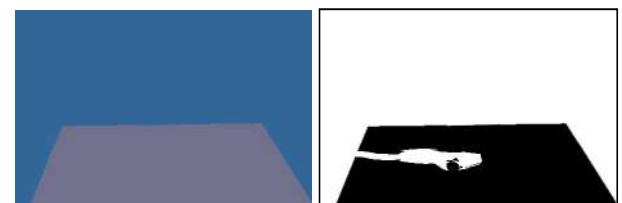
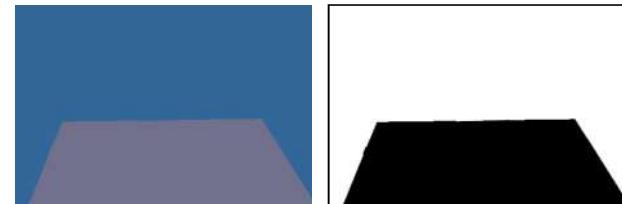
Color, Z

// 4. Dibuixa l'**occlusor**



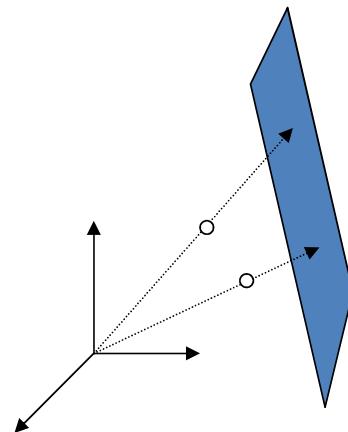
Ombres per projecció (amb stencil)

```
// 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_LEQUAL);  
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);
```



Projecció respecte l'origen

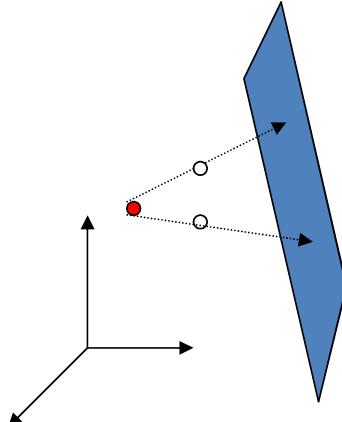
- Donats els coeficients (a, b, c, d) d'un pla, la matriu de projecció respecte l'origen és:



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

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:

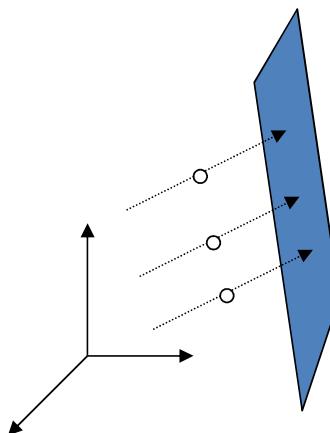


$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -(d+ax+by+cz) & 0 & 0 & 0 \\ 0 & -(d+ax+by+cz) & 0 & 0 \\ 0 & 0 & -(d+ax+by+cz) & 0 \\ a & b & c & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0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

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

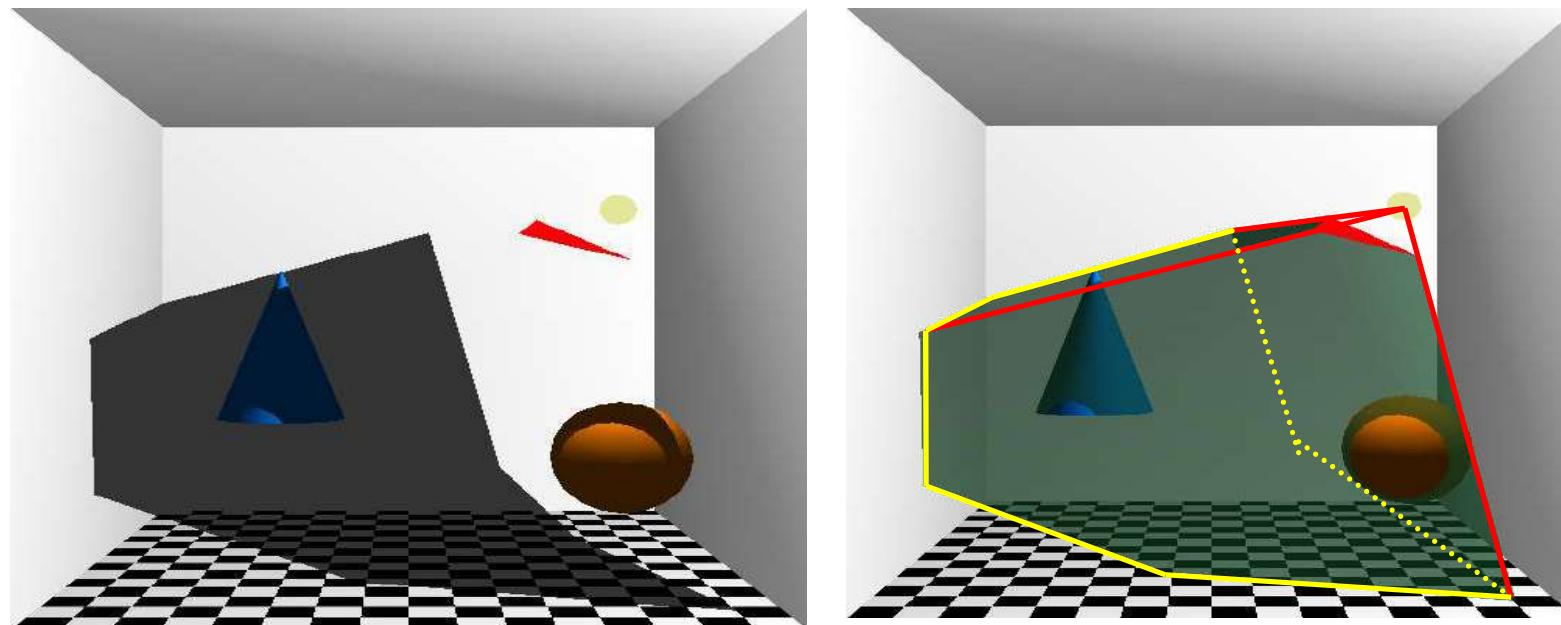
Projecció en la direcció (x,y,z)

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



$by + cz$	$-bx$	$-cx$	$-dx$
$-ay$	$ax + cz$	$-cy$	$-dy$
$-az$	$-bz$	$ax + by$	$-dz$
0	0	0	$ax + by + cz$

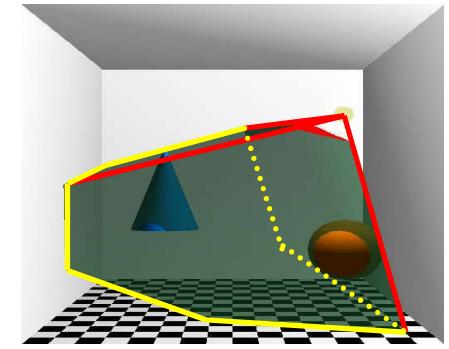
Volums d'ombra



Volums d'ombra (1/2)

// 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);  
glDepthMask(GL_FALSE);  
glStencilFunc(GL_ALWAYS, 0, 0);  
glEnable(GL_CULL_FACE);  
glStencilOp(GL_KEEP, GL_KEEP, GL_INCR);  
glCullFace(GL_BACK);  
dibuixa(volum_ombra);
```



// 3. Dibuixa al stencil les **cares posteriors del volum**

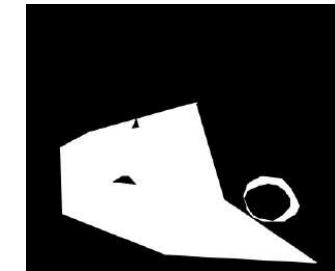
```
glStencilOp(GL_KEEP, GL_KEEP, GL_DECR);  
glCullFace(GL_FRONT);  
dibuixa(volum_ombra);
```



Volums d'ombra (2/2)

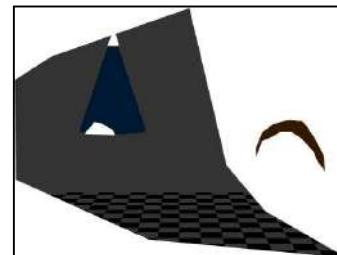
// 4. Dibuixa al color buffer la part fosca de l'escena

```
glDepthMask(GL_TRUE);  
glColorMask(GL_TRUE, ... , GL_TRUE);  
glCullFace(GL_BACK);  
glDepthFunc(GL_LEQUAL);  
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

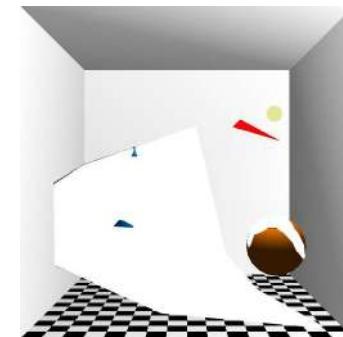
```
glStencilFunc(GL_EQUAL, 0, 1);  
glEnable(GL_LIGHTING);  
dibuixa(escena);
```



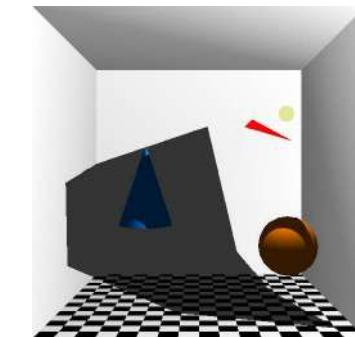
+

// 6. Restaura l'estat inicial

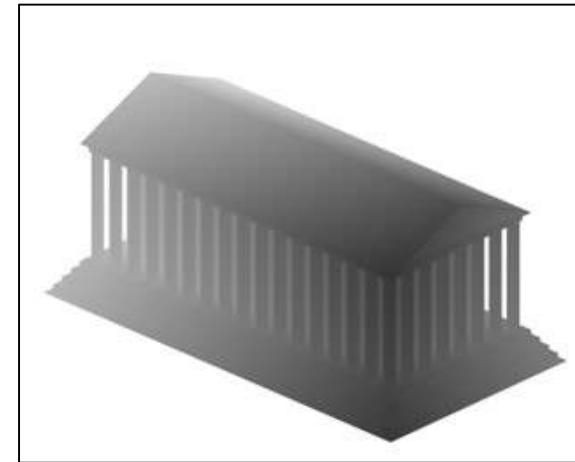
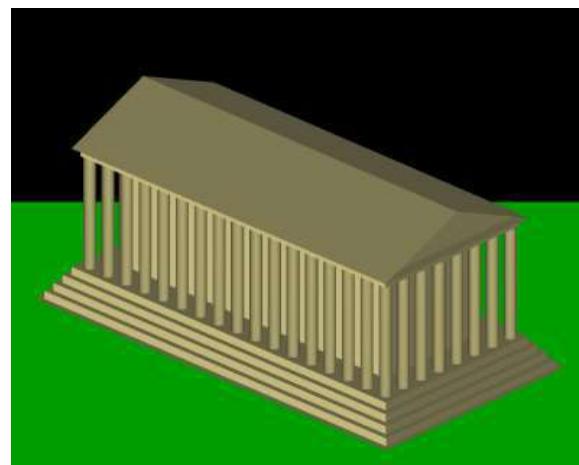
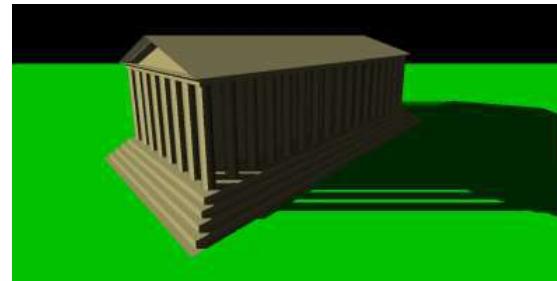
```
glDepthFunc(GL_LESS);  
glDisable(GL_STENCIL_TEST);
```

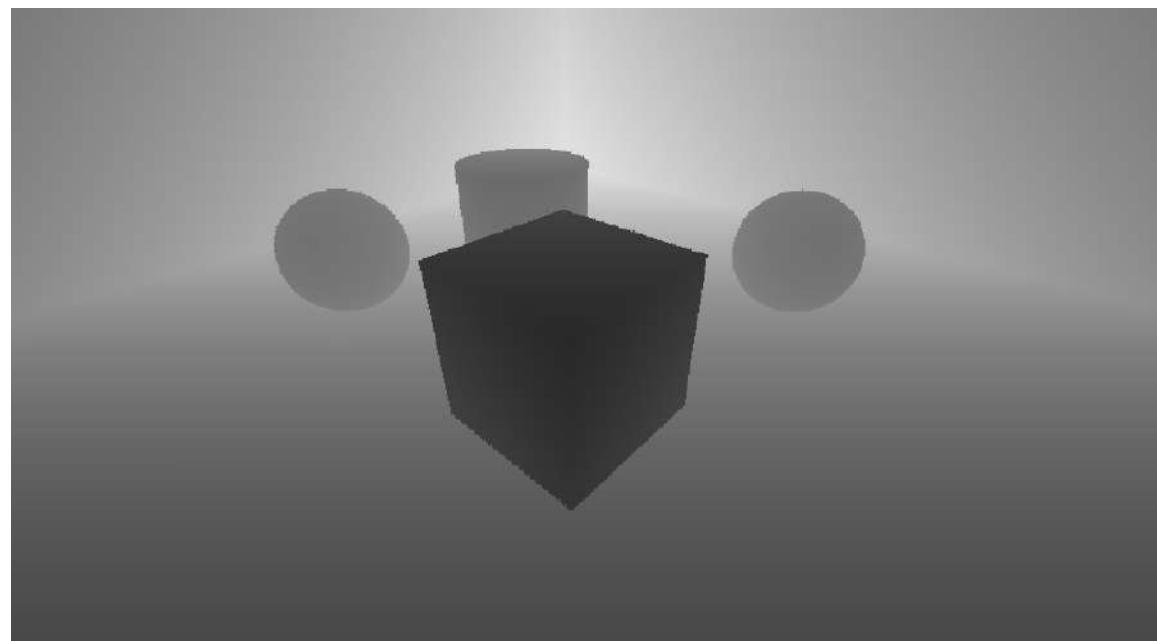
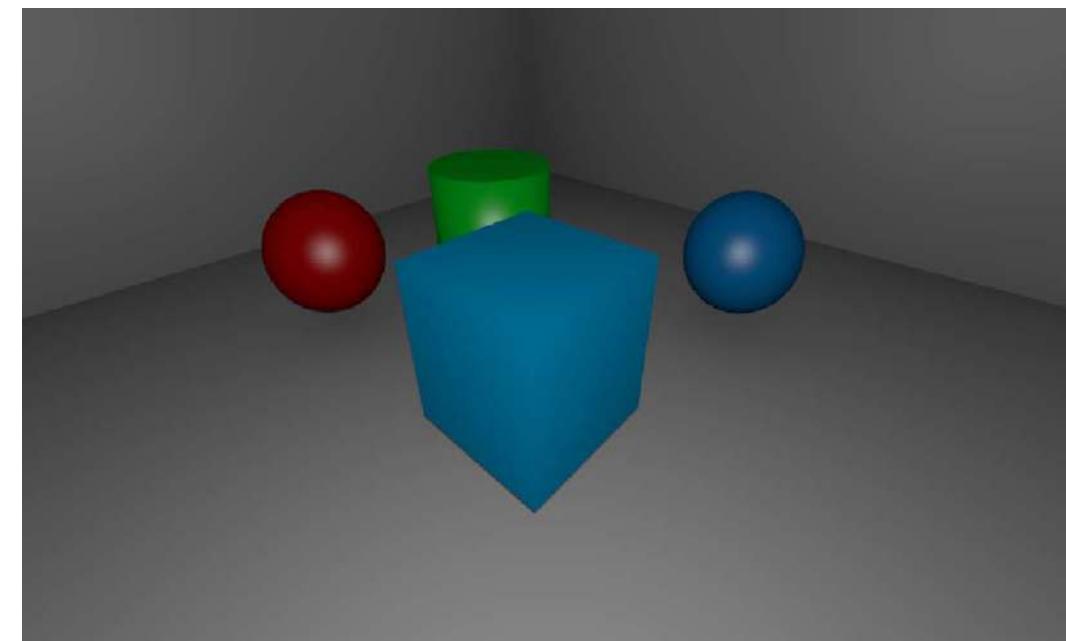
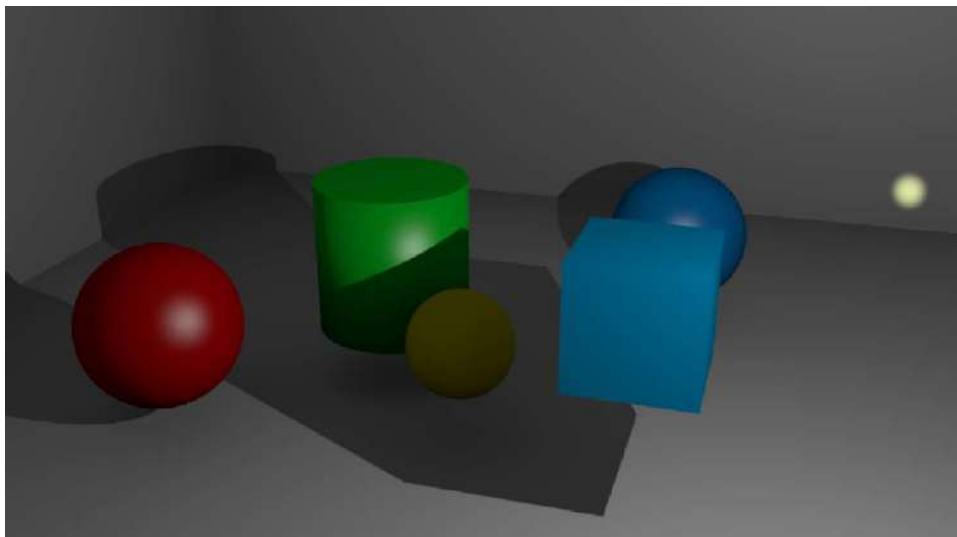


=

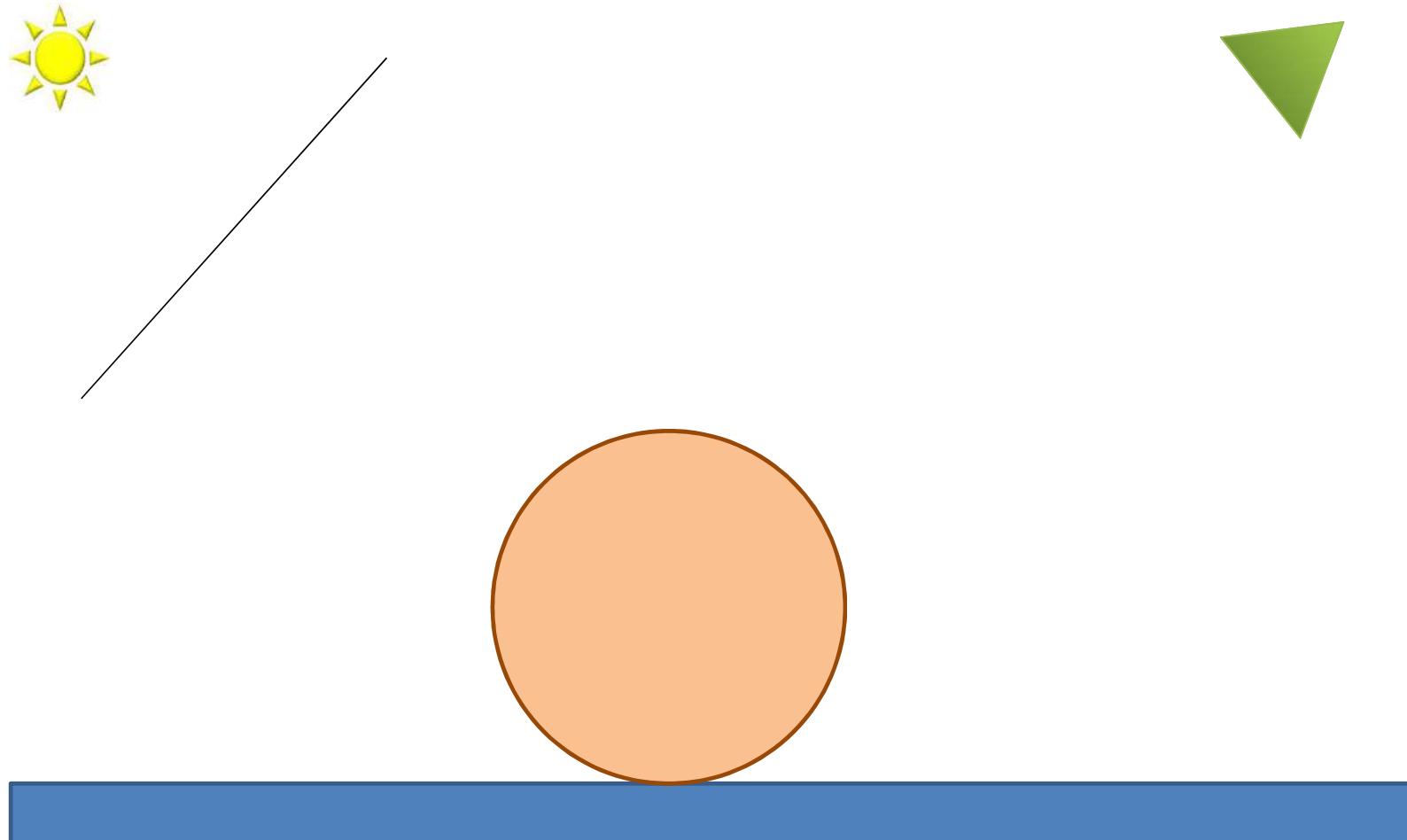


Shadow mapping





Shadow mapping



Shadow mapping - setup

```
// Setup shadow map (un cop)
glActiveTexture(GL_TEXTURE0);
 glGenTextures( 1, &textureId);
 glBindTexture(GL_TEXTURE_2D, textureId);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
 glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
 glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
 glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT32, SHADOW_MAP_WIDTH,
 SHADOW_MAP_HEIGHT, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
```

Shadow mapping – pass 1

```
// 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);

// Restaurar càmera i viewport
```

Shadow mapping – pass 2

// Generació de coords de textura pel shadow map

// La generació és similar a projective texture mapping

```
glLoadIdentity();
```

```
glTranslated( 0.5, 0.5, 0.5 );
```

```
glScaled( 0.5, 0.5, 0.5 );
```

```
gluPerspective( fov, ar, near, far);
```

```
gluLookAt( lightPos, ... lightTarget, ... up...);
```

→ La matriu resultant és la que passa les coordenades del vertex $(x,y,z,1)$ de *world space* a *homogeneous texture space* (s,t,p,q)

Shadow mapping - VS

```
// vs  
uniform mat4 lightMatrix;  
out vec4 texCoord;  
  
void main()  
{  
    ...  
texCoord = lightMatrix*vec4(vertex,1);  
    gl_Position = modelViewProjectionMatrix * vec4(vertex,1);  
}
```

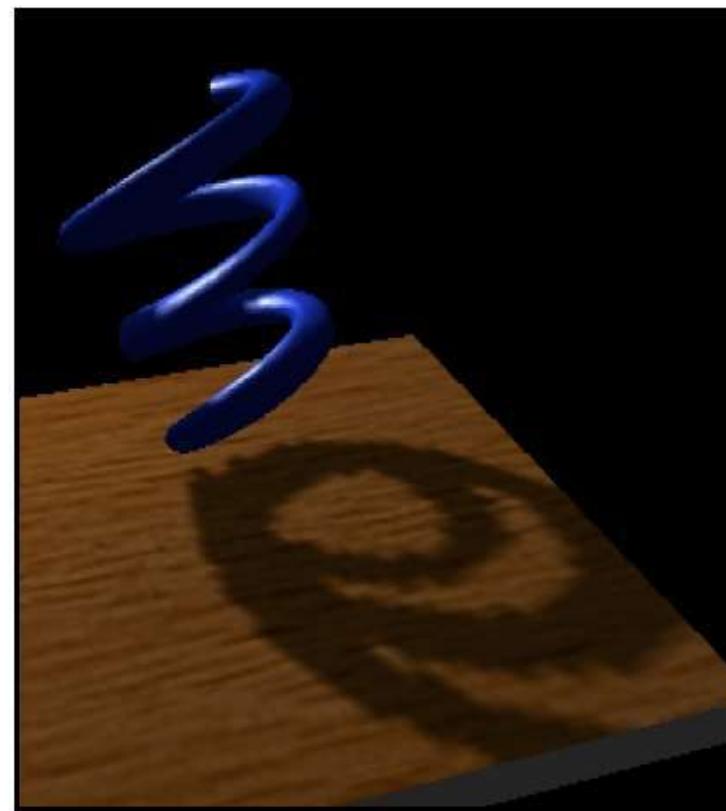
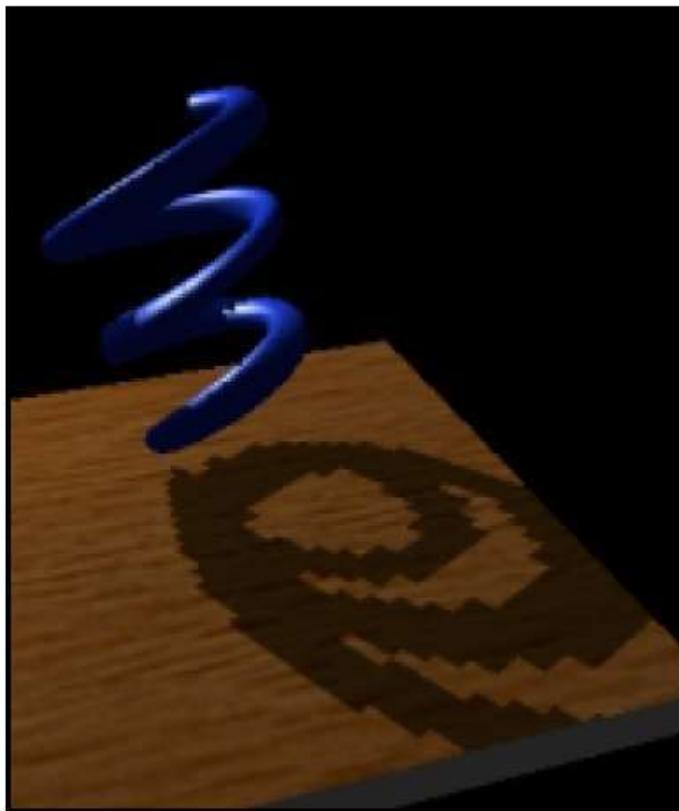
Shadow mapping - FS

// FS

...

```
vec2 st = texCoord.st / texCoord.q;  
float trueDepth = texCoord.p / texCoord.q;  
float storedDepth = texture(shadowMap, st).r;  
float bias = 0.01; // només si no hem usat glPolygonOffset  
if (trueDepth - bias <= storedDepth)  
    fragColor = ... // iluminat  
else  
    fragColor = ... // a l'ombra
```

Shadow map problems



Reflexions especulars

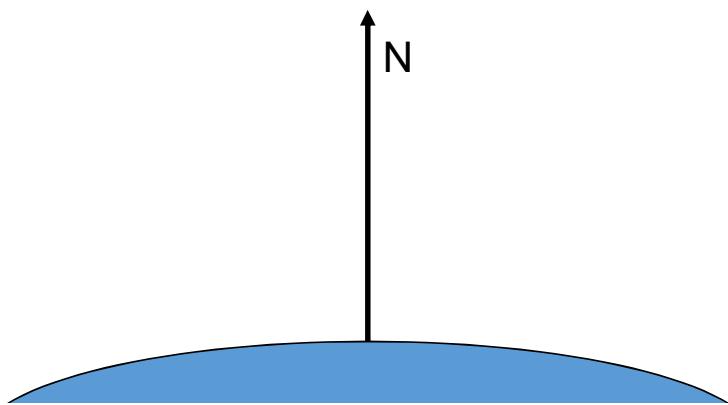
Carlos Andújar

Nov 2022

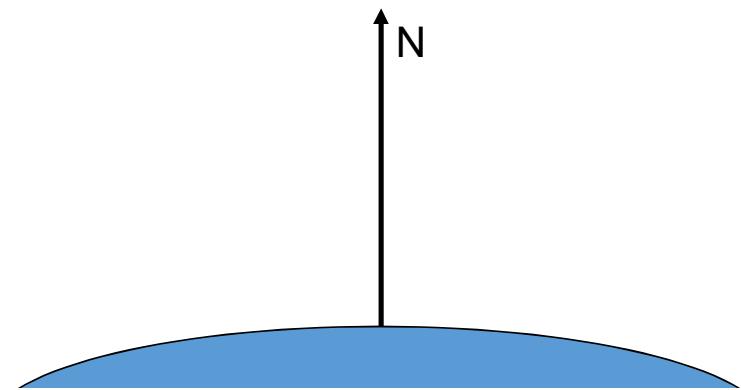
Introducció



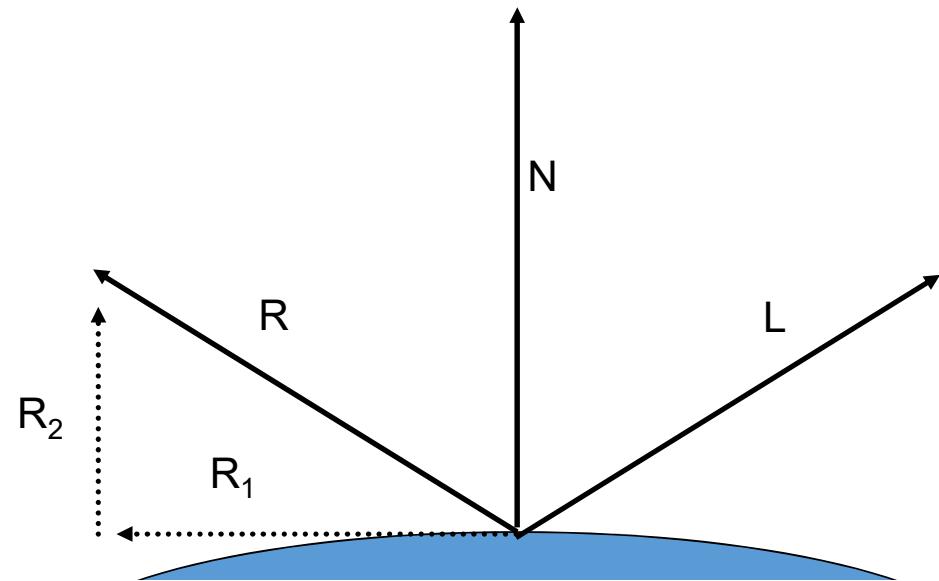
Reflexió difosa



Reflexió especular



Vector reflectit



Vector reflectit

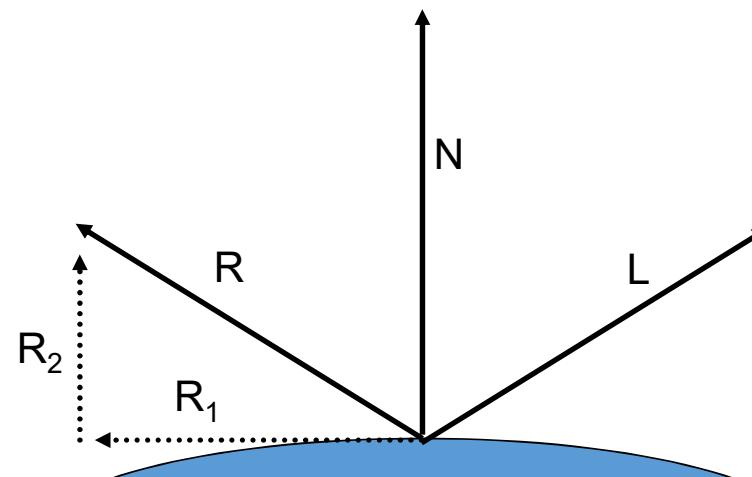
$$R = R_1 + R_2$$

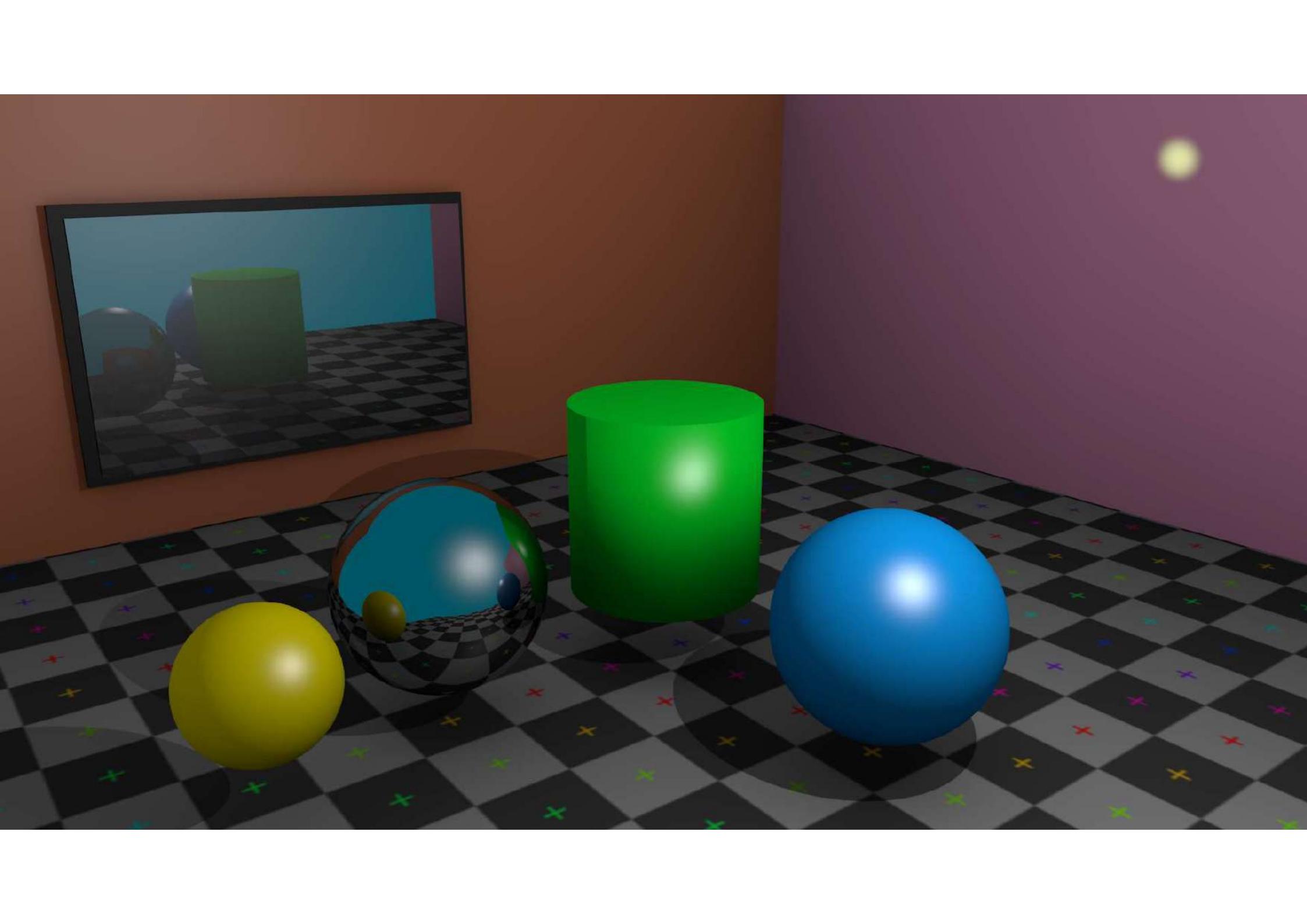
$$R_1 = -L + R_2$$

$$R = 2R_2 - L$$

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

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





Mètodes

- Ray-tracing
- Reflexions basada en **objectes virtuals**
 - Modelats
 - Reflectits (sense stencil test)
 - Reflectits (amb stencil test)
 - Textures dinàmiques
- **Environment mapping**
 - Sphere mapping
 - Cube mapping

Reflexions amb objectes virtuals

Mètodes

- Ray-tracing
- Reflexions basada en **objectes virtuals**
 - Modelats
 - Reflectits (sense stencil test)
 - Reflectits (amb stencil test)
 - Textures dinàmiques
- **Environment mapping**
 - Sphere mapping
 - Cube mapping

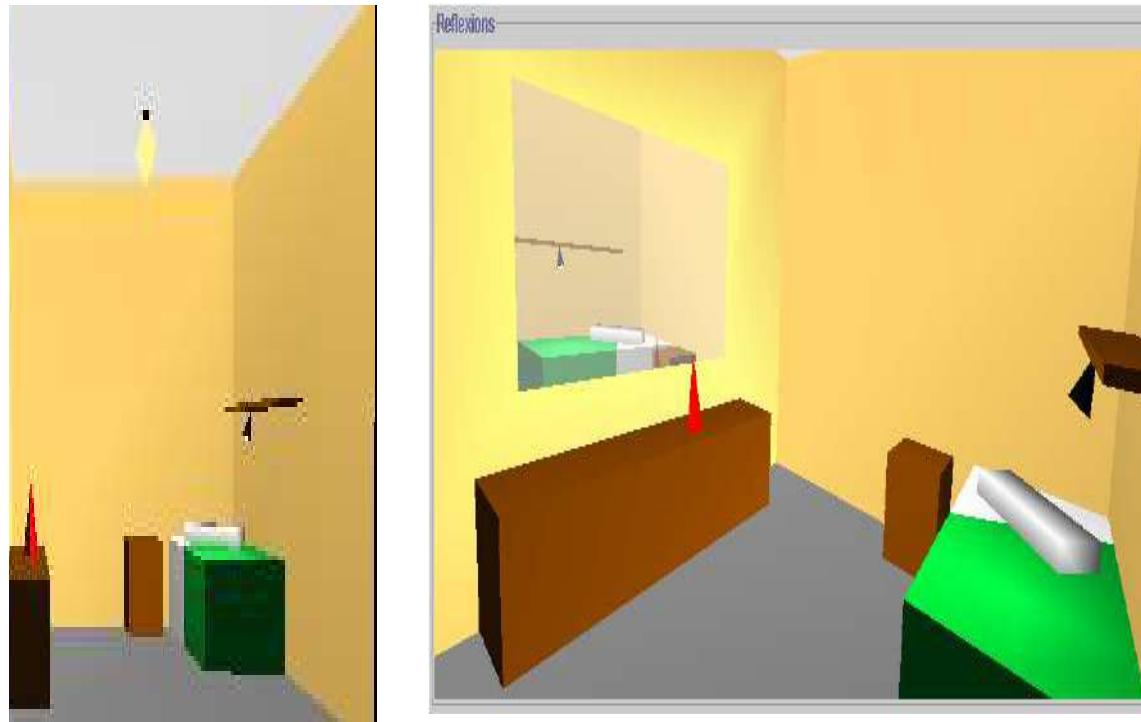
Objectes virtuals



Mètodes

- Ray-tracing
- Reflexions basada en **objectes virtuals**
 - Modelats
 - Reflectits (sense stencil test)
 - Reflectits (amb stencil test)
 - Textures dinàmiques
- **Environment mapping**
 - Sphere mapping
 - Cube mapping

Objectes virtuals



Mètodes

- Ray-tracing
- Reflexions basada en **objectes virtuals**
 - Modelats
 - Reflectits (sense stencil test)
 - Reflectits (amb stencil test)
 - Textures dinàmiques
- **Environment mapping**
 - Sphere mapping
 - Cube mapping

Algorisme (versió 1)

// 1. Dibuixar els objectes en posició virtual

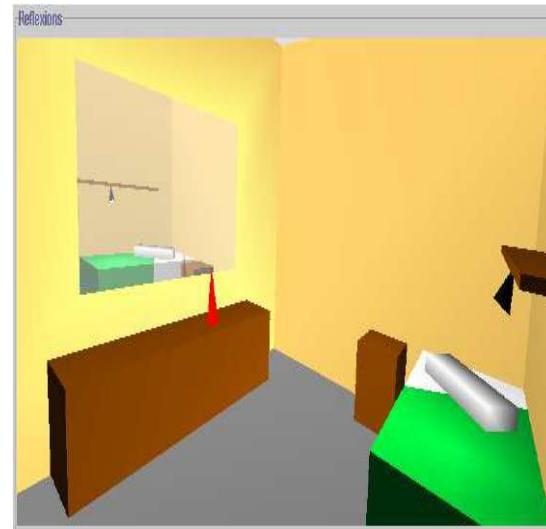
```
glPushMatrix();
glMultMatrix(matriu_simetria)
glLightfv(GL_LIGHT0, GL_POSITION, pos);
glCullFace(GL_FRONT);
dibuixar(escena);
glPopMatrix();
```

// 2. Dibuixar el mirall semi-transparent

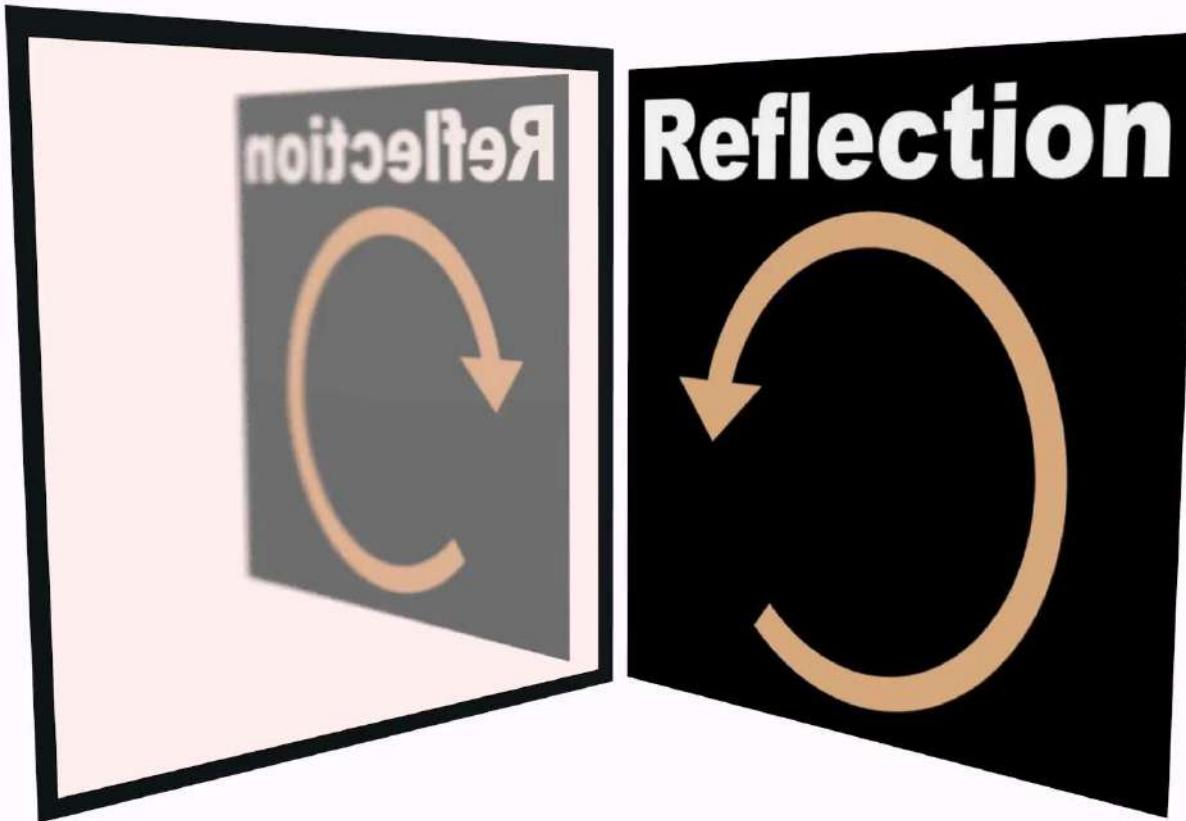
```
glLightfv(GL_LIGHT0, GL_POSITION, pos);
glCullFace(GL_BACK);
dibuixar(mirall);
```

// 3. Dibuixar els objects en posició real

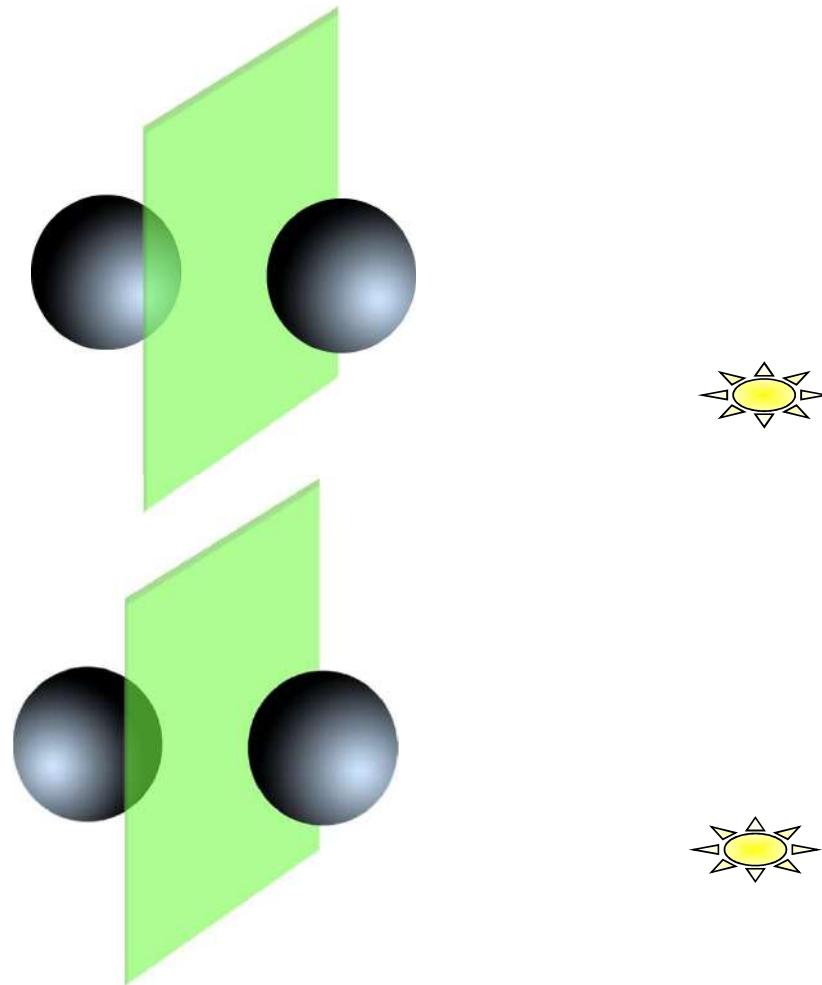
```
dibuixar(escena);
```



Inversió ordre

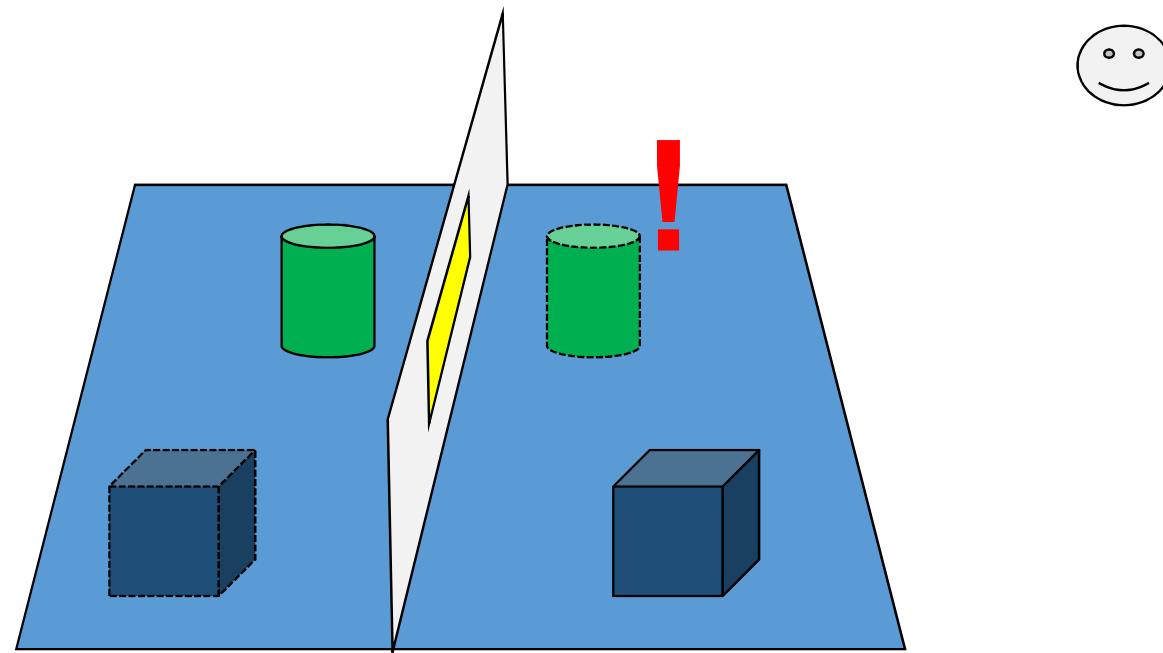


Reflexió de les fonts de llum



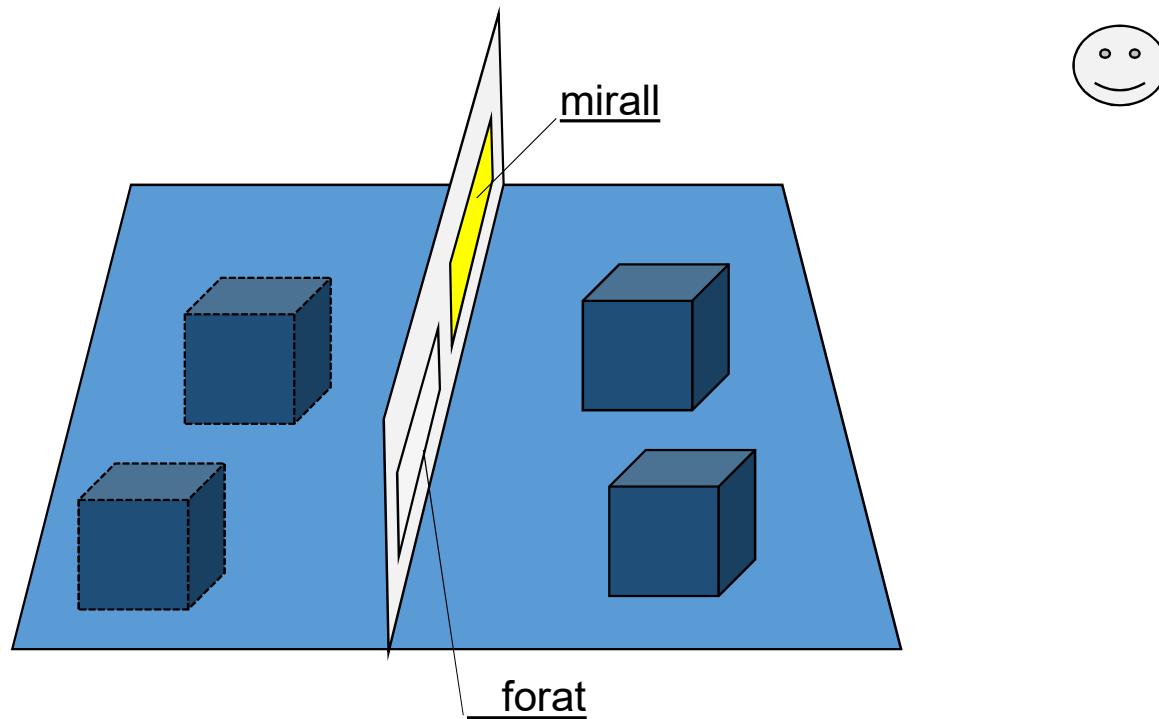
Limitacions

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



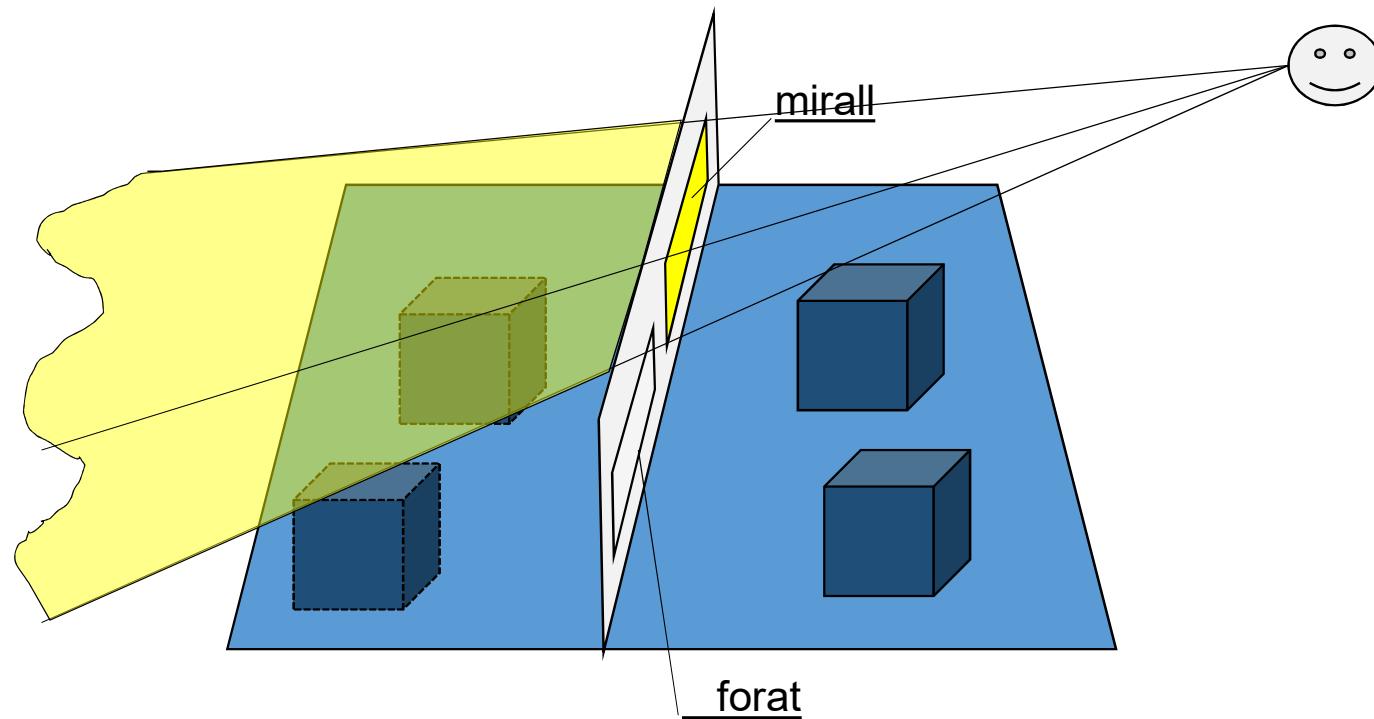
Limitacions

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



Solució 1

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



Solució 2

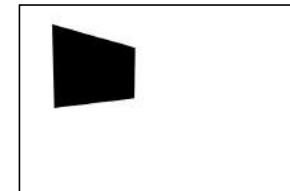
Usar stencil per limitar els objectes virtuals a la regió ocupada pel mirall.

Mètodes

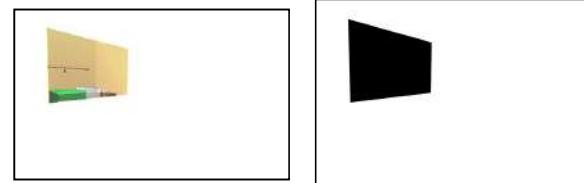
- Ray-tracing
- Reflexions basada en **objectes virtuals**
 - Modelats
 - Reflectits (sense stencil test)
 - Reflectits (amb stencil test)
 - Textures dinàmiques
- **Environment mapping**
 - Sphere mapping
 - Cube mapping

Algorisme (versió 2)

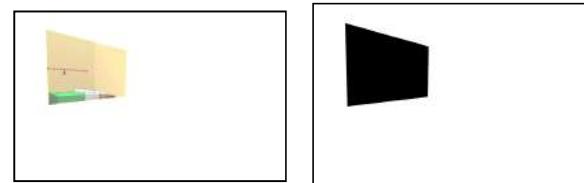
Pas 1. Dibuixar mirall al stencil buffer



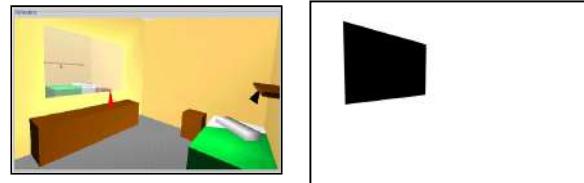
Pas 2. Dibuixar objectes en pos virtual



Pas 3. Dibuixar mirall semi-transparent



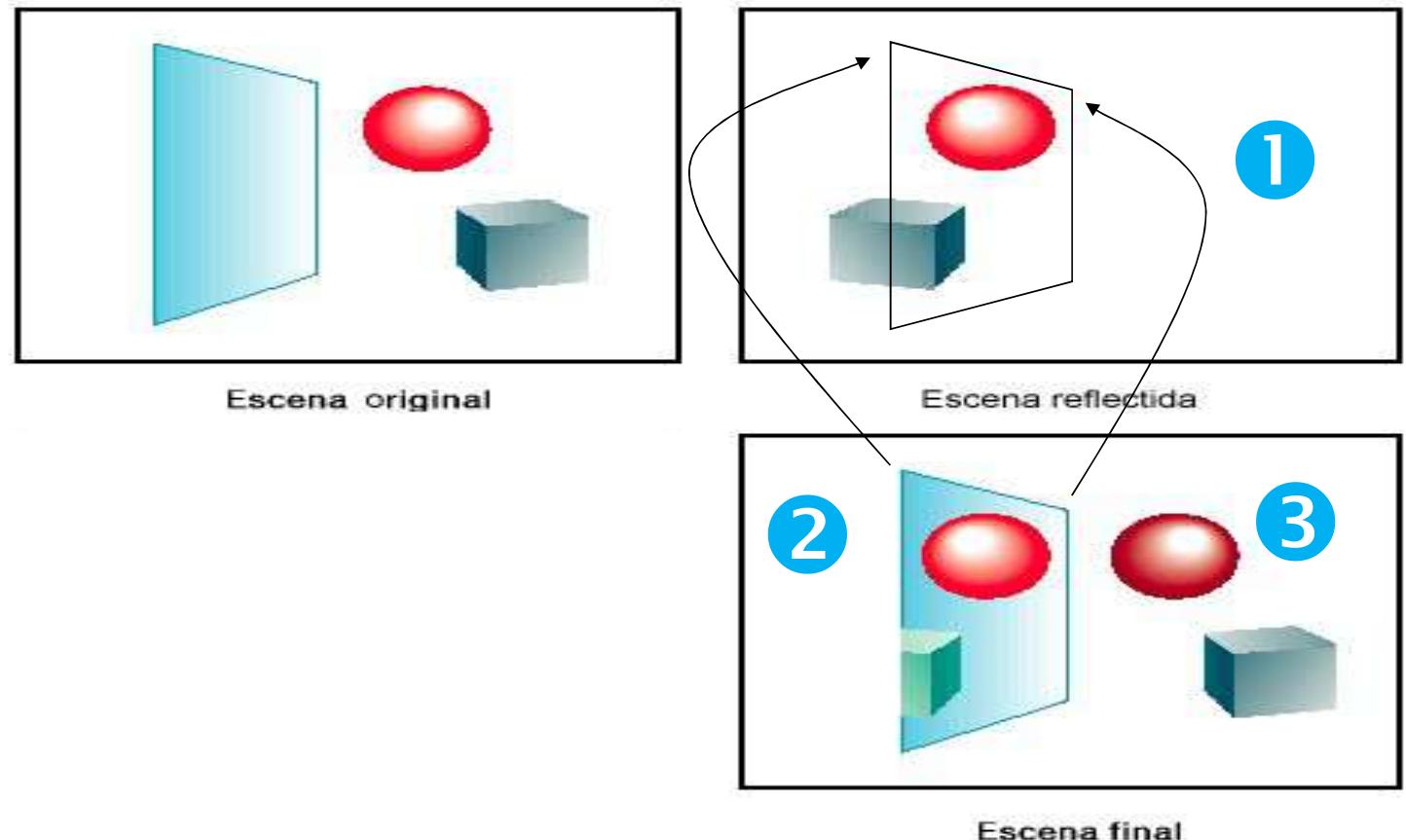
Pas 4. Dibuixar objectes en pos real



Mètodes

- Ray-tracing
- Reflexions basada en **objectes virtuals**
 - Modelats
 - Reflectits (sense stencil test)
 - Reflectits (amb stencil test)
 - Textures dinàmiques
- **Environment mapping**
 - Sphere mapping
 - Cube mapping

Textures dinàmiques



Matriu de reflexió

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

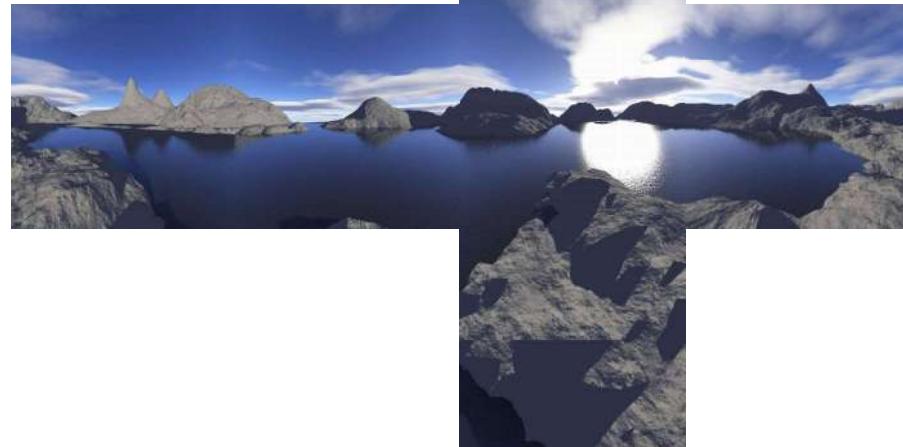
Environment map

Donat una direcció arbitrària R, ens retorna el color de l'entorn en direcció R

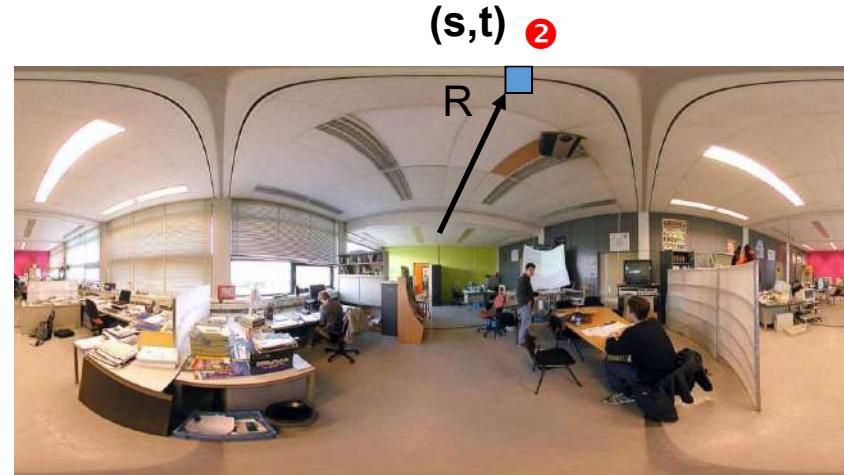
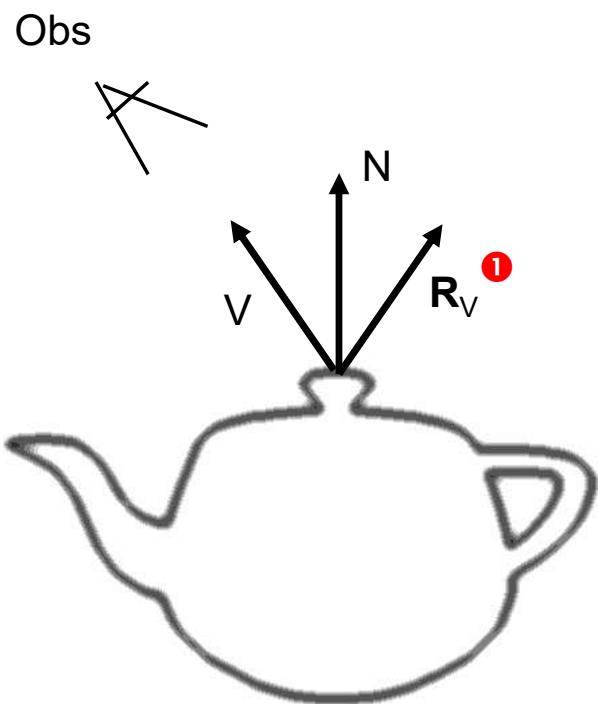
$$\text{color} = \text{environmentMap}(R)$$



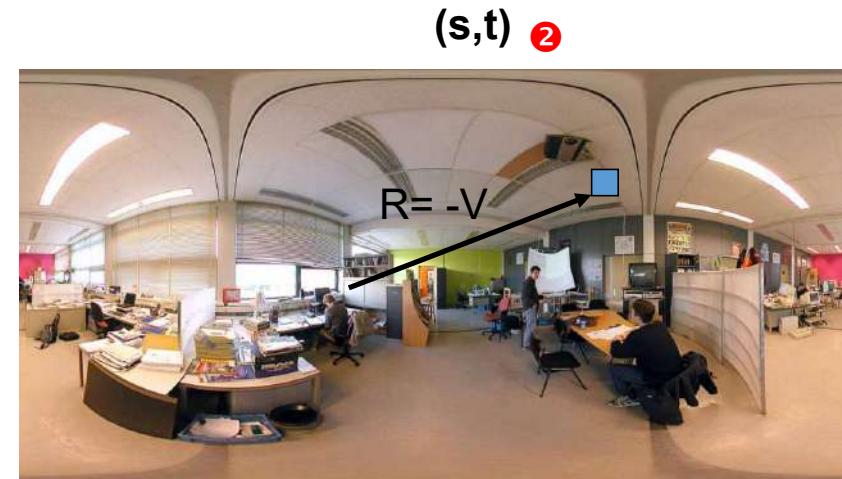
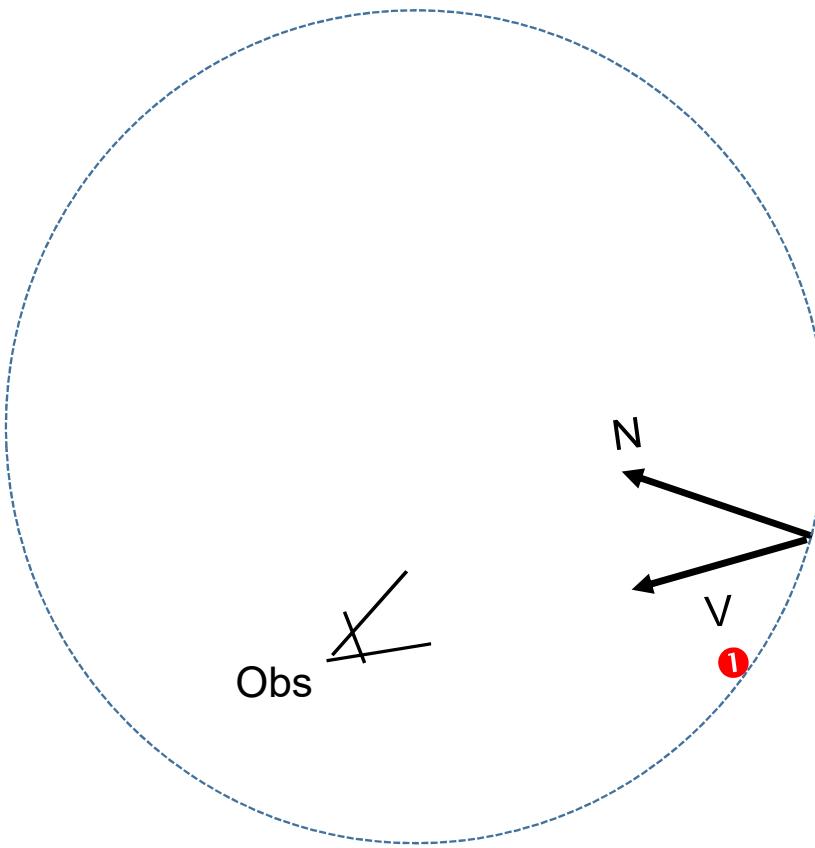
Representació com a textura



Ús per reflexions especulars



Ús com a entorn (background)



Sphere mapping

Sphere map

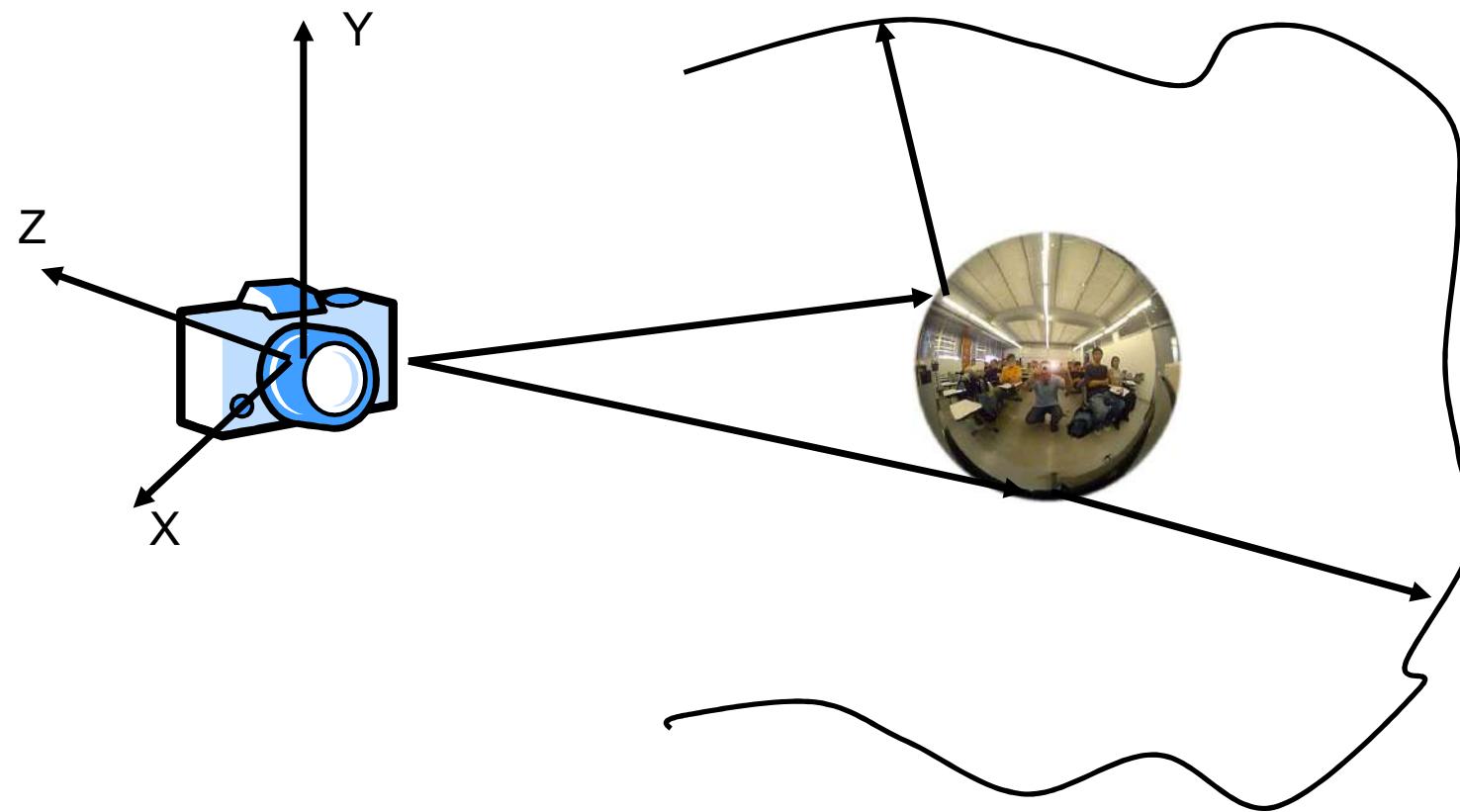


Sphere map

Hand with Reflecting Sphere by M. C. Escher.
Lithograph, 1935.
Official M.C. Escher website.

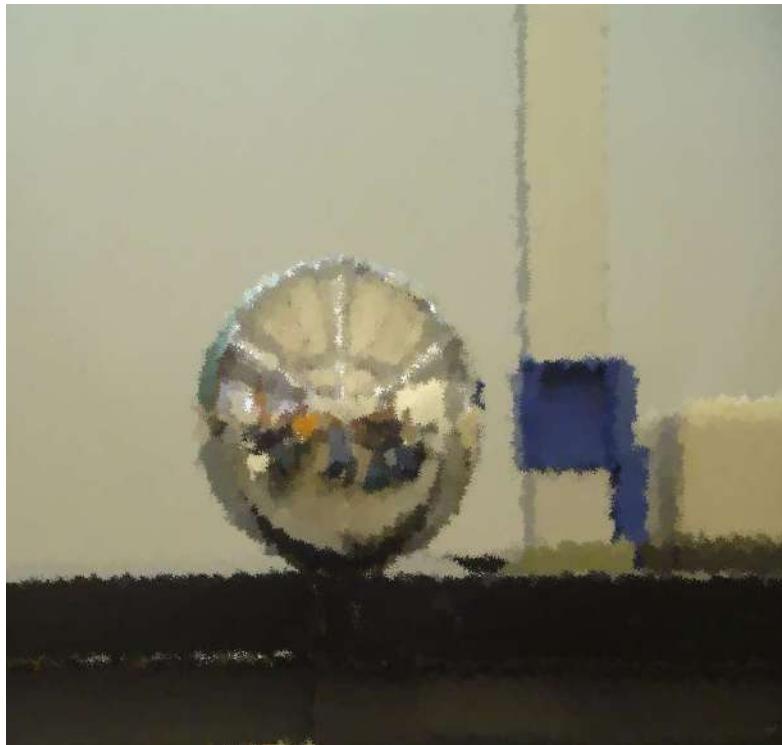


Construcció de sphere maps



Exemple sphere map

Sense retallar

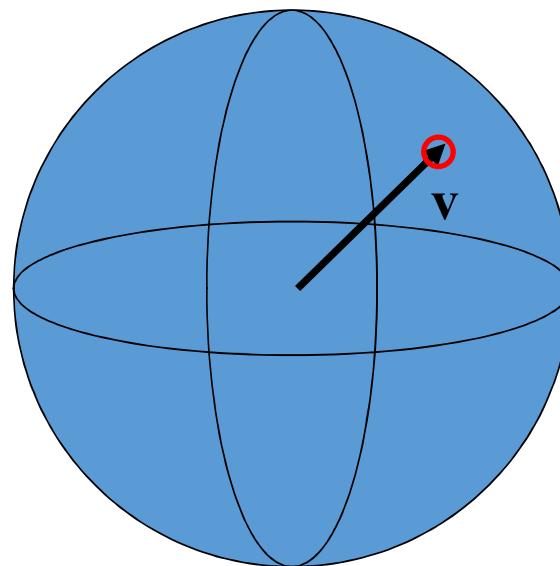


Retallat



Propietats

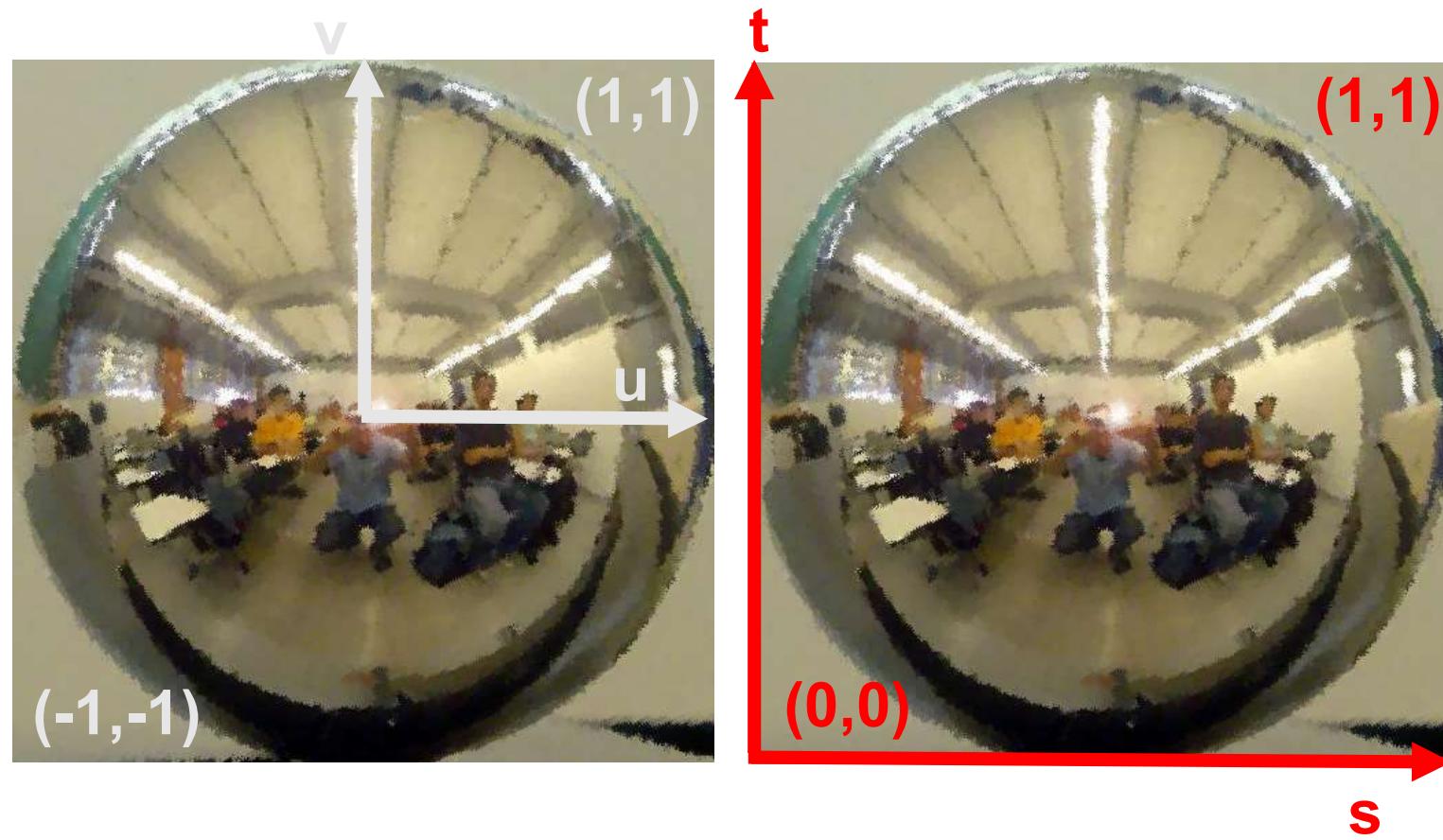
- De la textura només se n'aprofita el cercle inscrit
- Conté informació de aproximadament tot l'entorn (totes direccions)
- Distorsió considerable a prop de la vora del cercle



Exemple



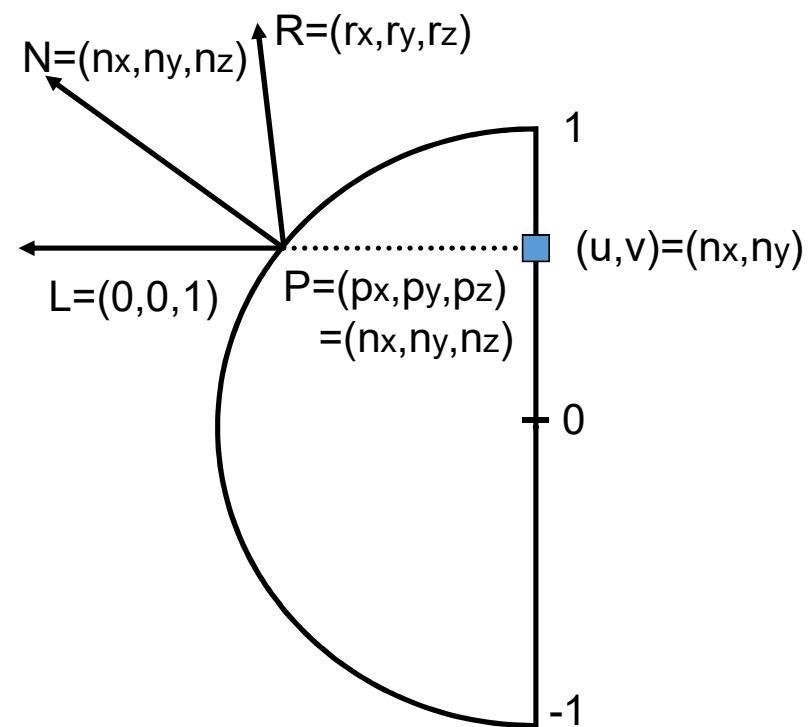
Coordenades $(u,v) \leftrightarrow (s,t)$



Relació entre vector R i (u,v)

Relació entre vector R i (u,v)

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



Càlcul 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 texture(sampler, st);
}
```

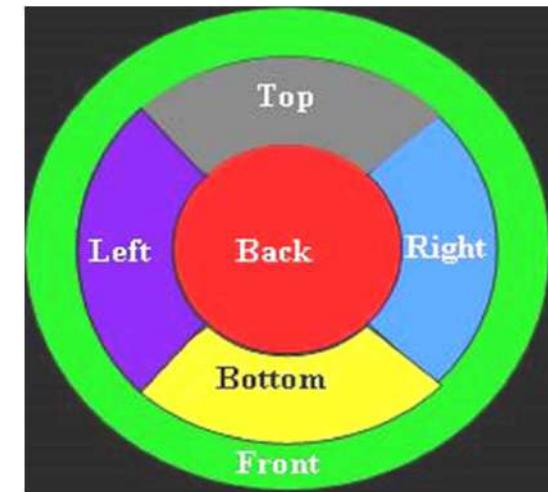
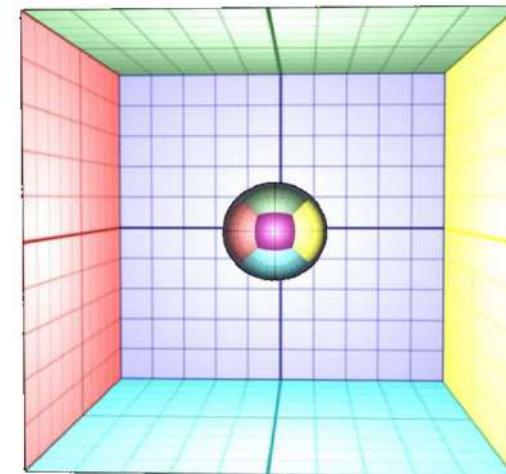
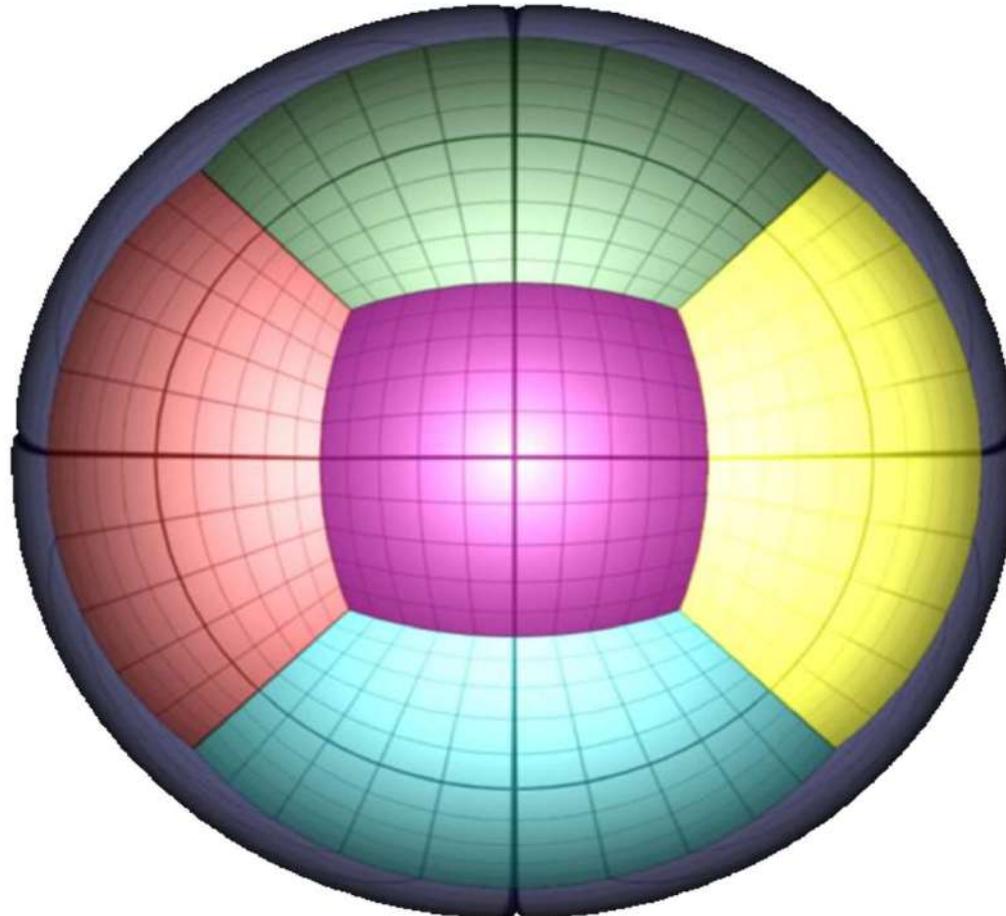
Eye/world coordinates



Eye/world coordinates



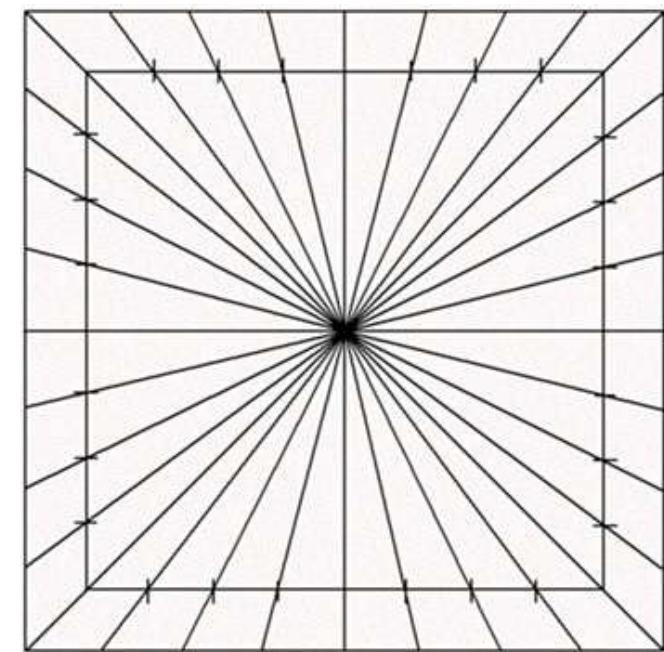
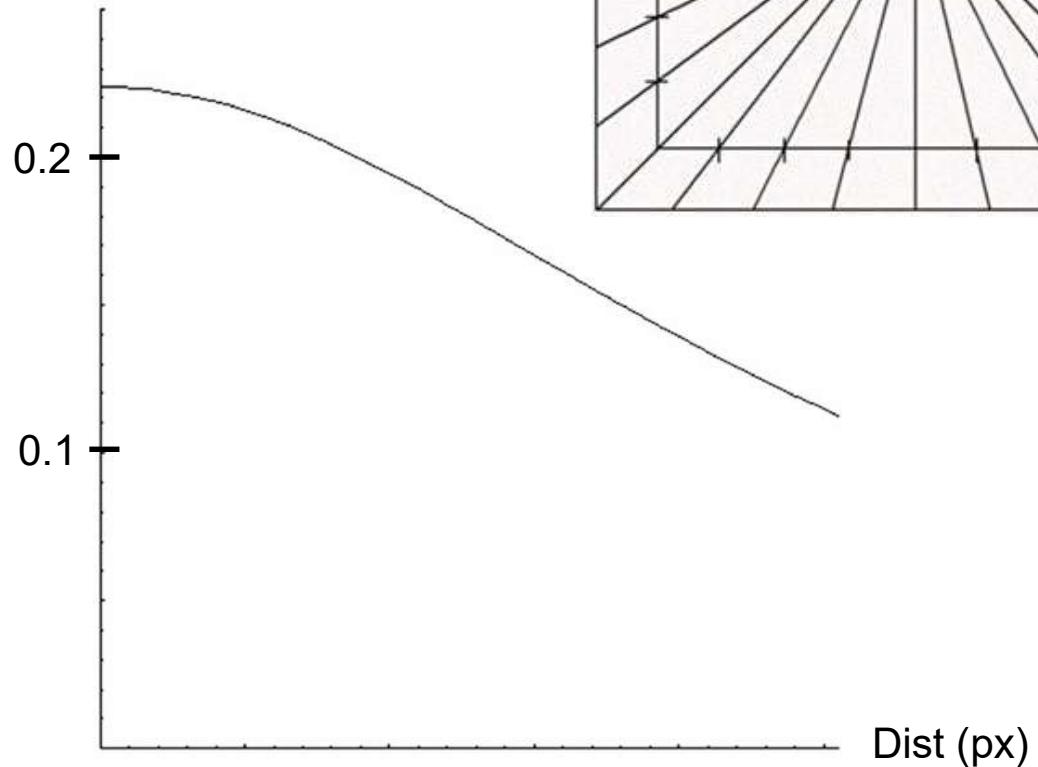
Sphere mapping: distorsió



CUBE MAPPING



Angle (deg)



54

Cube mapping: exemple



Cube mapping: exemple

// 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, ...);
```

Cube mapping: GLSL

```
uniform sampler2D sampler;  
...  
fragColor = texture(sampler, vtexCoord);
```

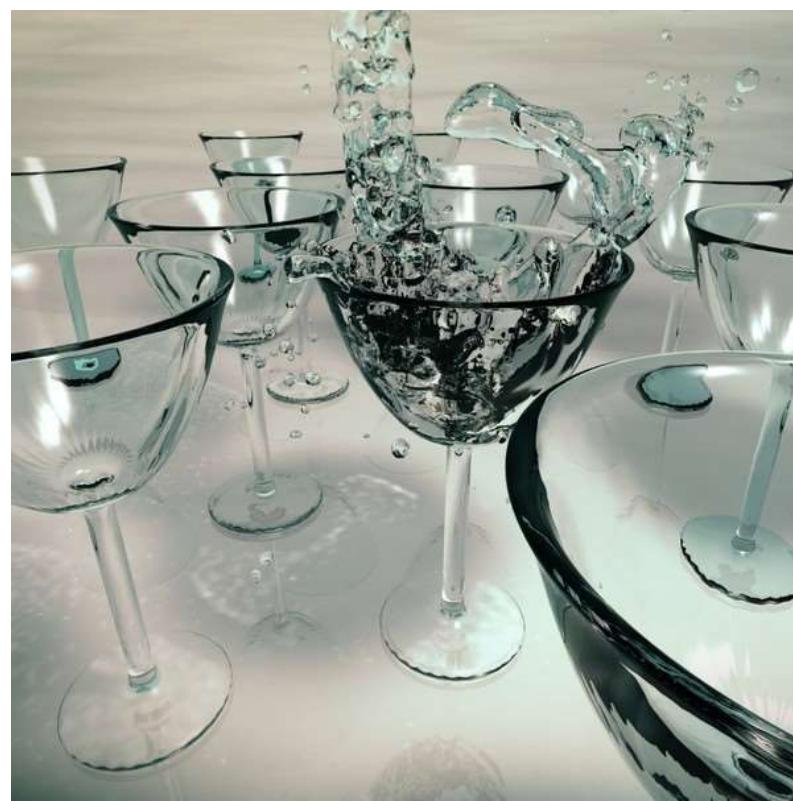
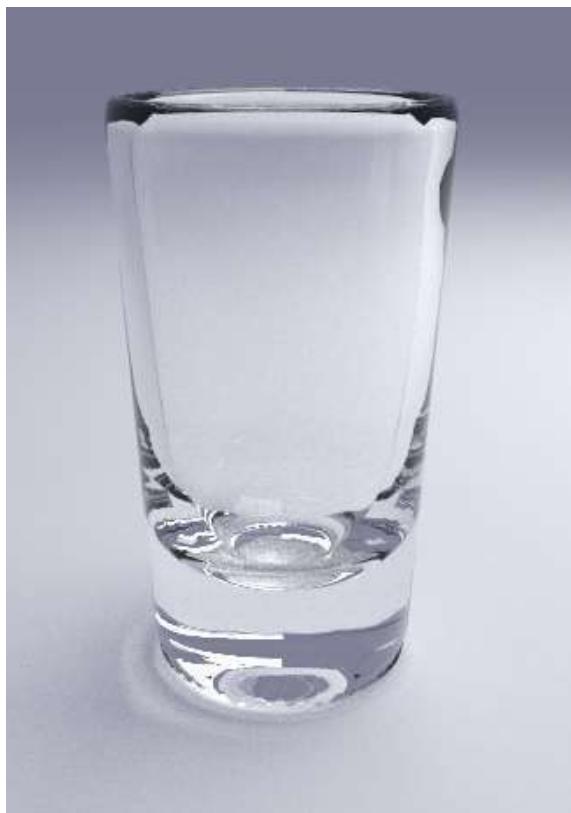
```
uniform samplerCube samplerC;  
...  
vec3 R;  
...  
fragColor = textureCube(samplerC, R);
```

Simulació d'objectes translúcids

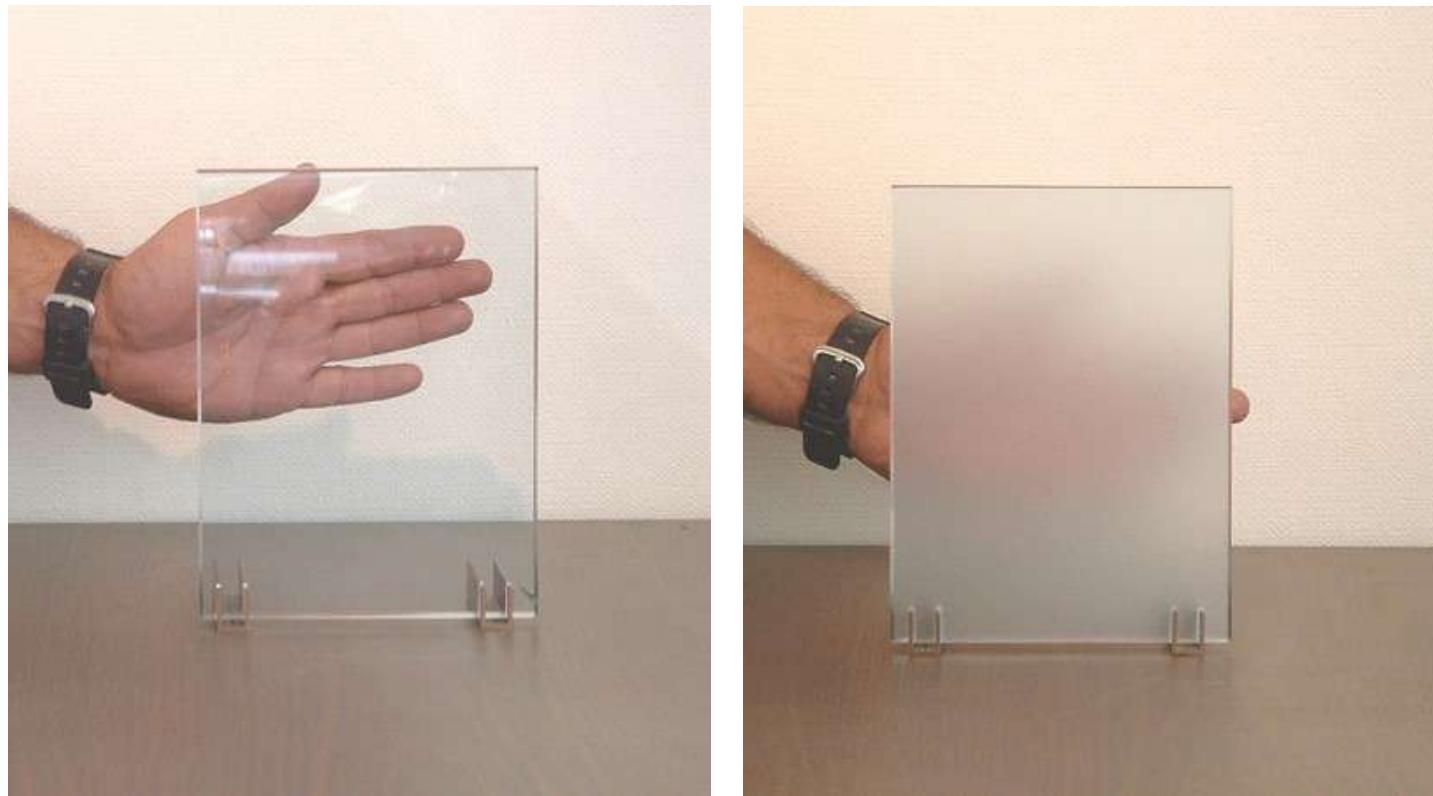
Carlos Andújar

Abril 2022

Introducció



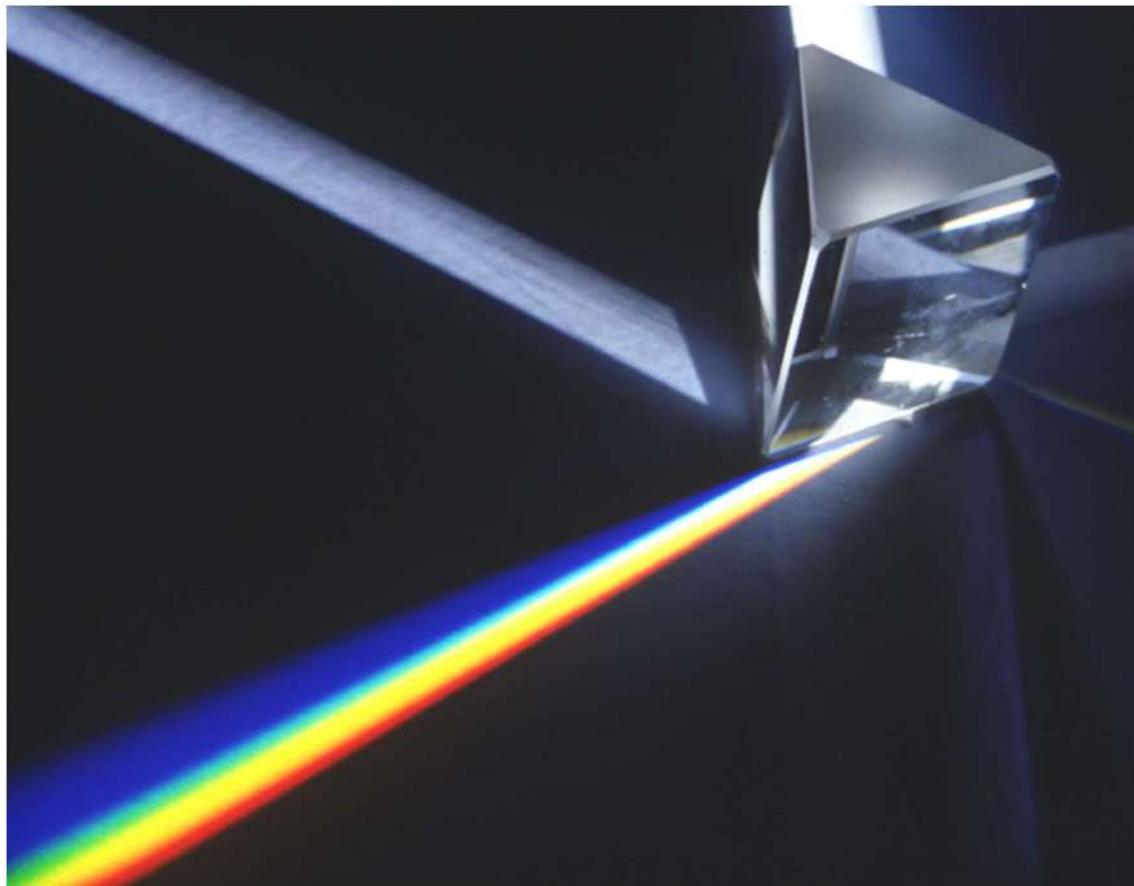
Dispersió de la llum transmesa



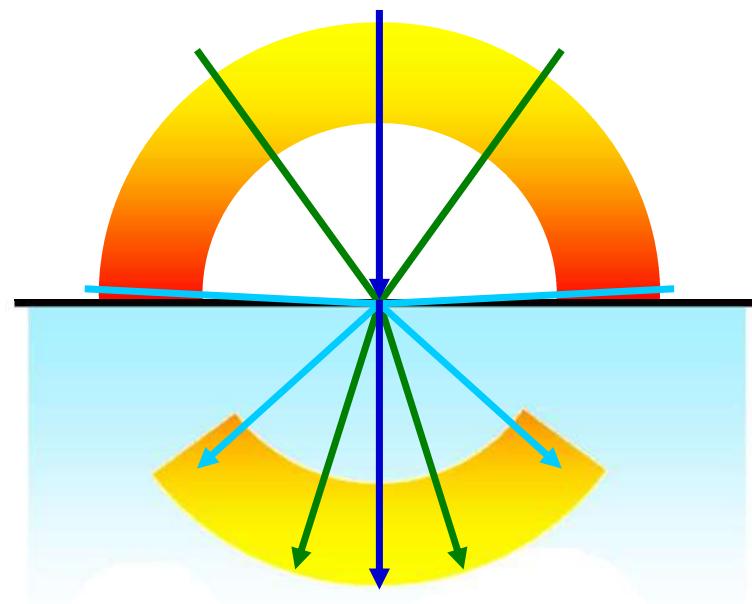
Refracció: índexs de refracció

Buit	1.0
Aire	1.0003
Gel	1.31
Aigua a 20º C	1.33
Alcohol	1.36
Cristall	1.52
Safir	1.77
Diamant	2.417

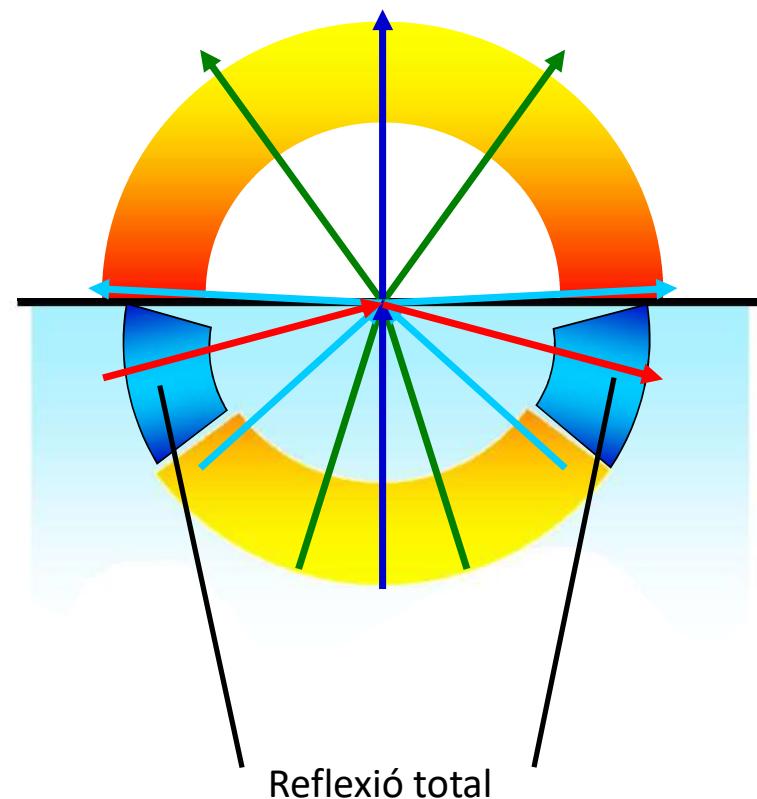
Refracció i longitud d'ona



Refracció aire→aigua



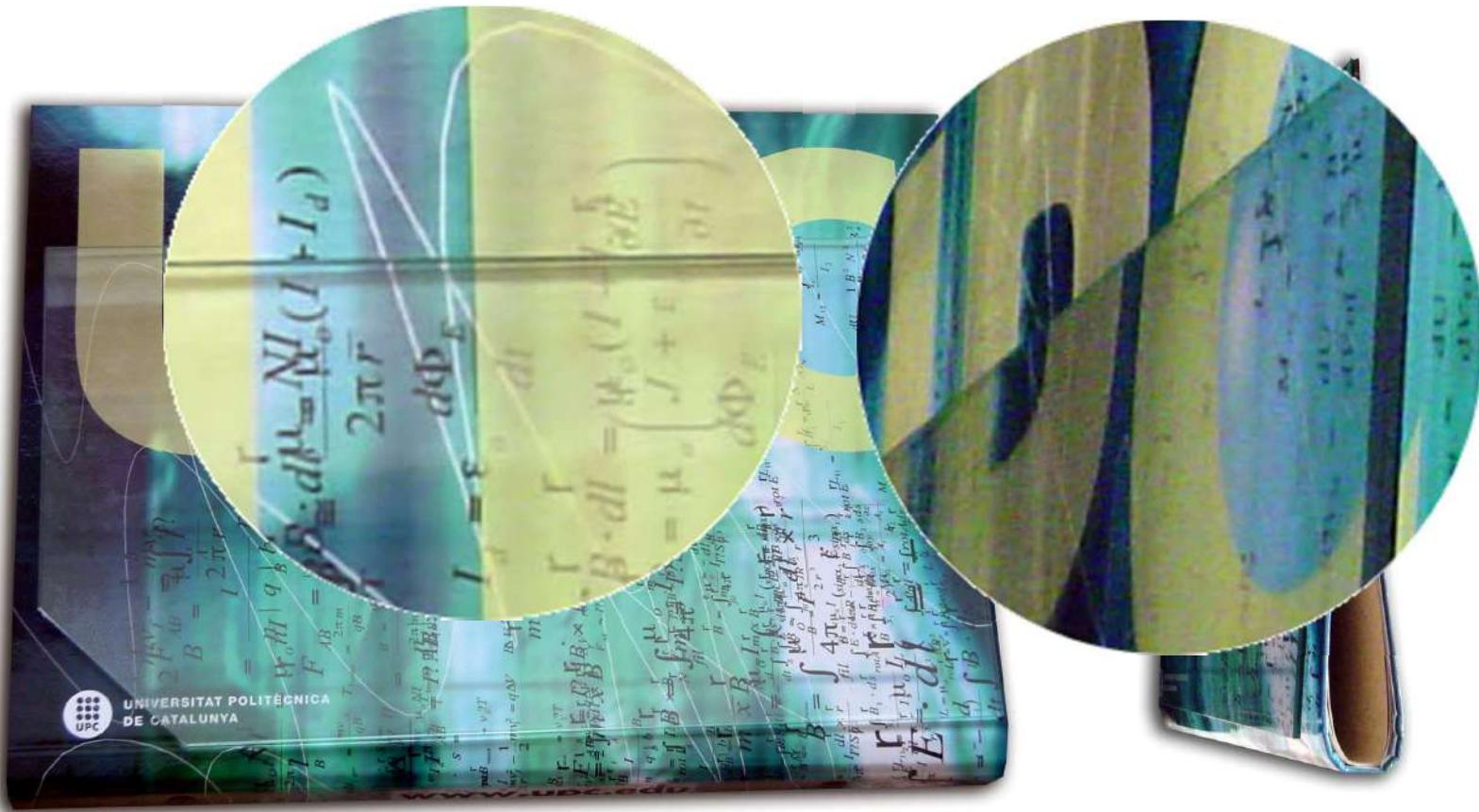
Refracció aigua→aire



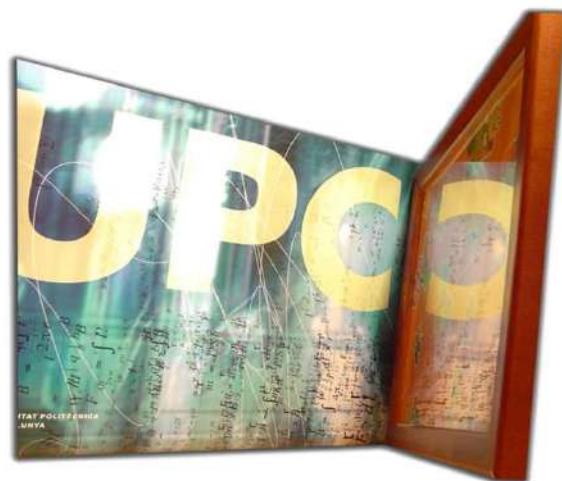
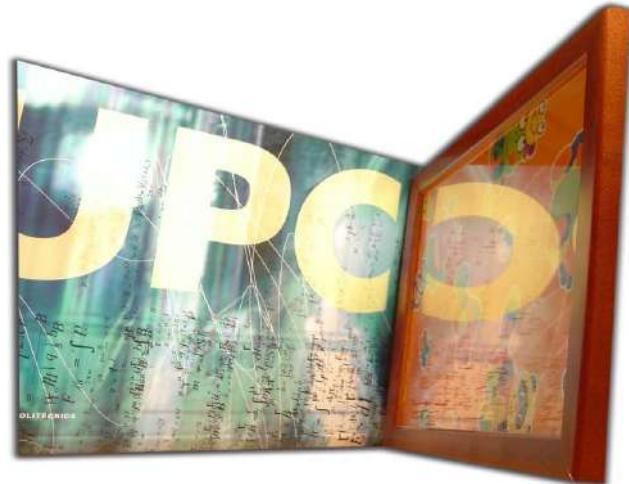
Angle crític



Refracció: superfícies paral·leles



Equaciones de Fresnel

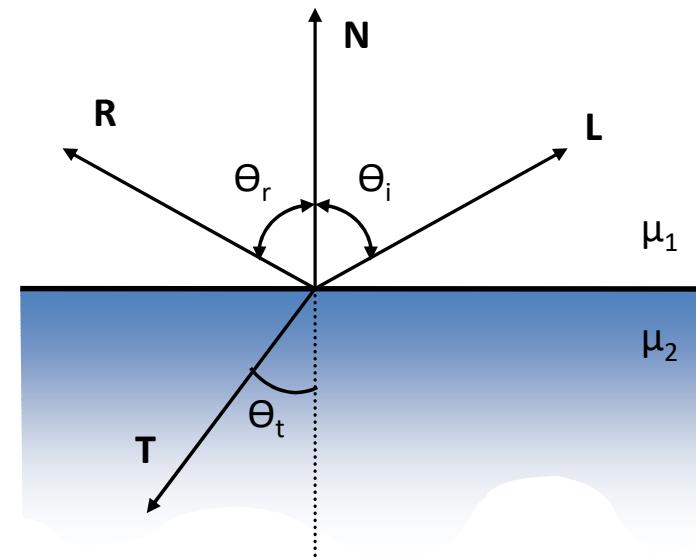


Equacions de Fresnel

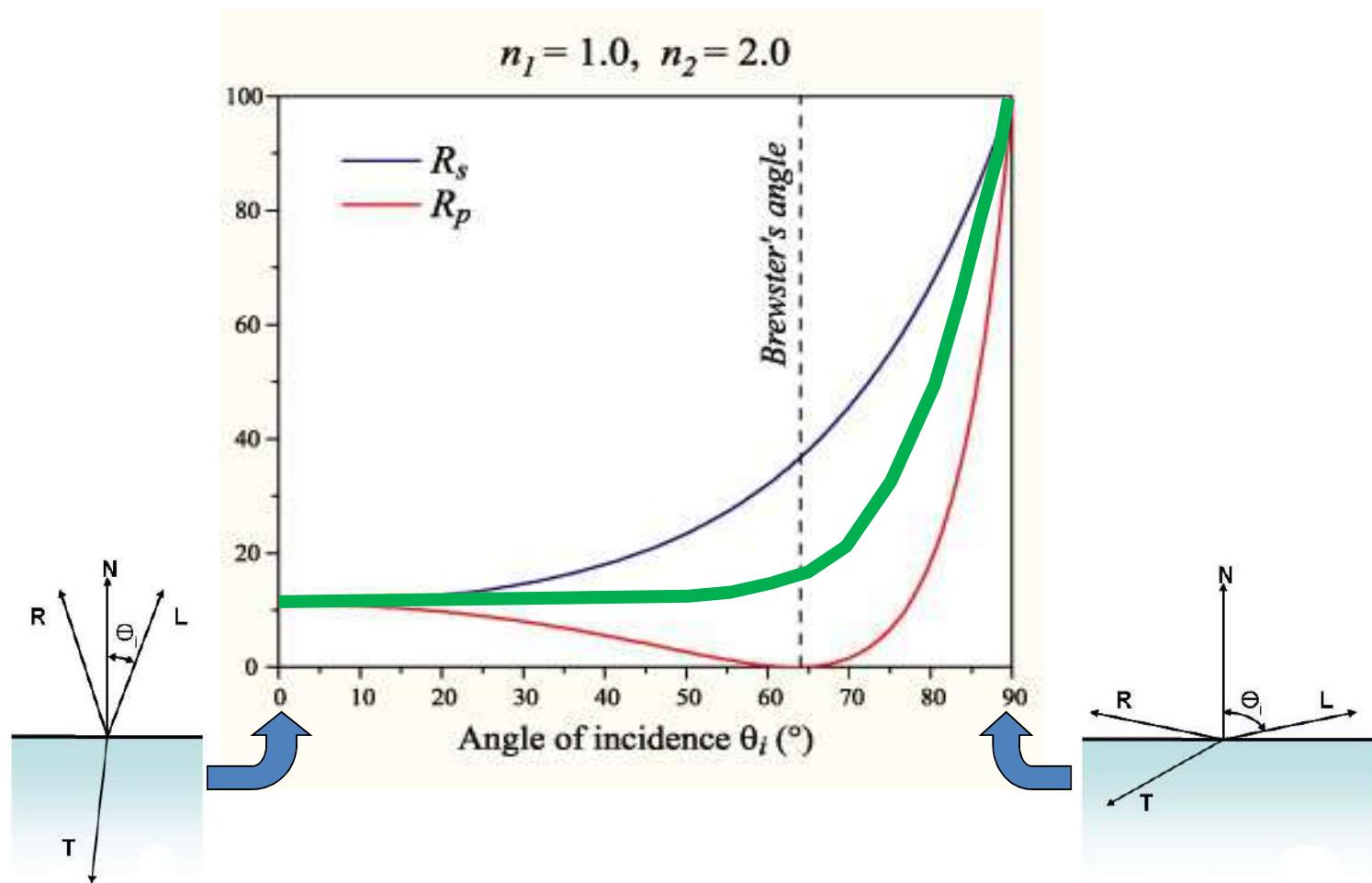
$$R = \frac{R_s + R_p}{2}$$

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

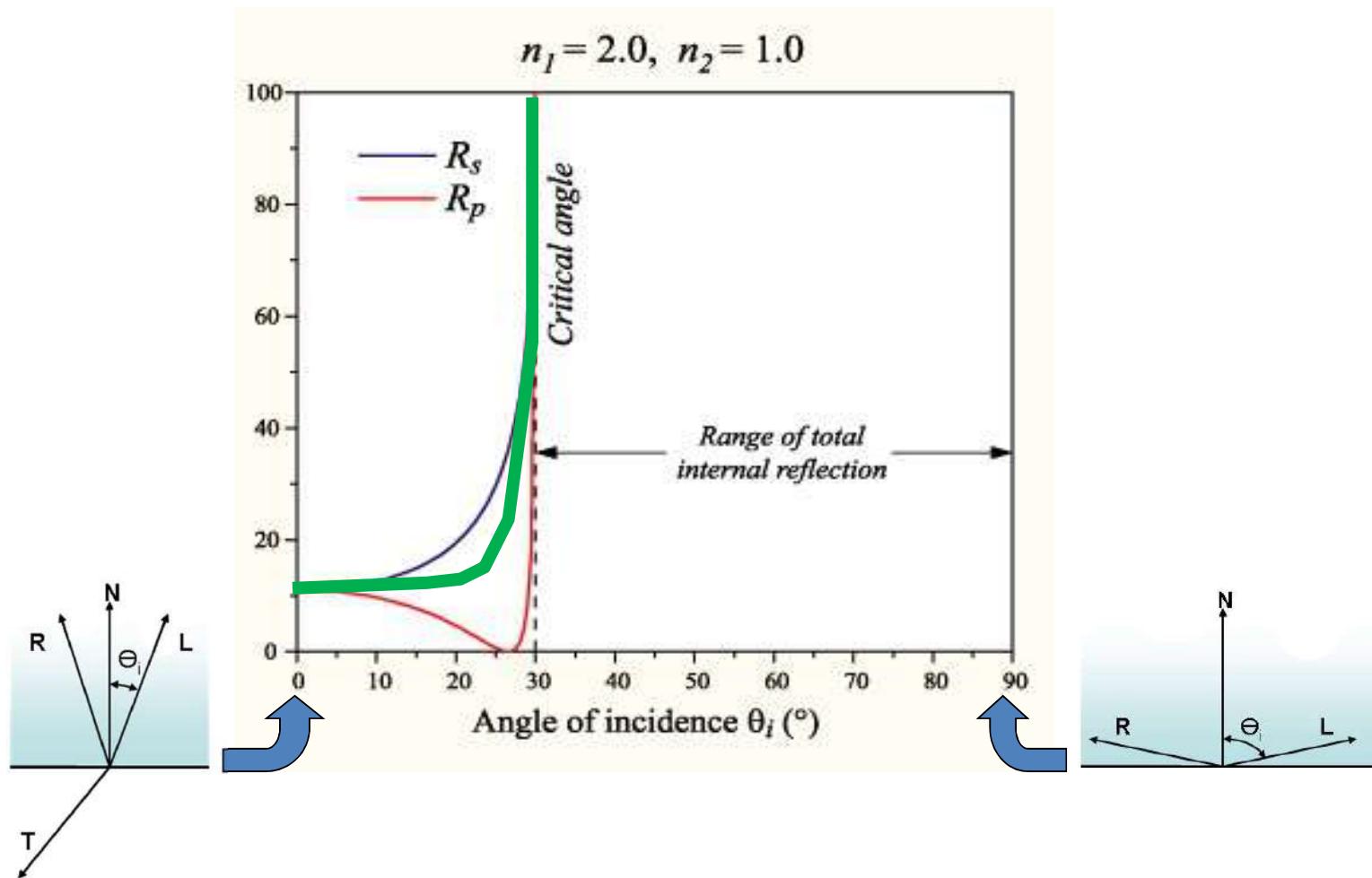
$$R_p = \left(\frac{\tan(\theta_t - \theta_i)}{\tan(\theta_t + \theta_i)} \right)^2$$



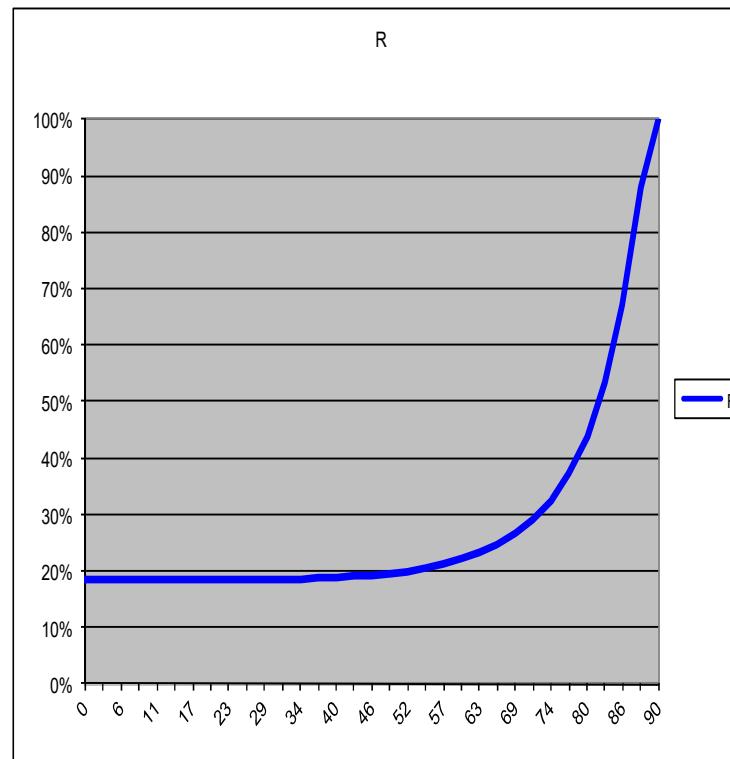
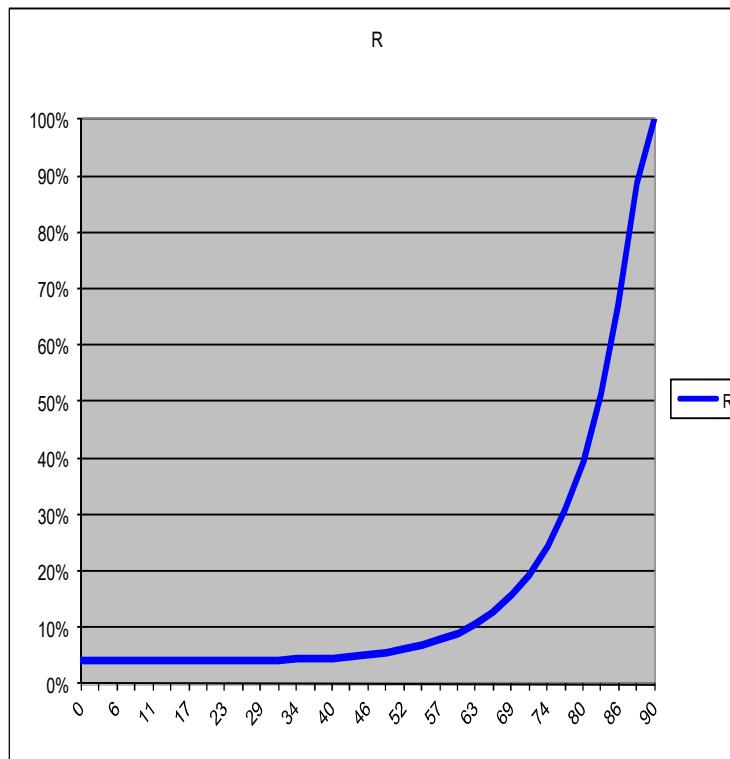
Buit → Medi dens



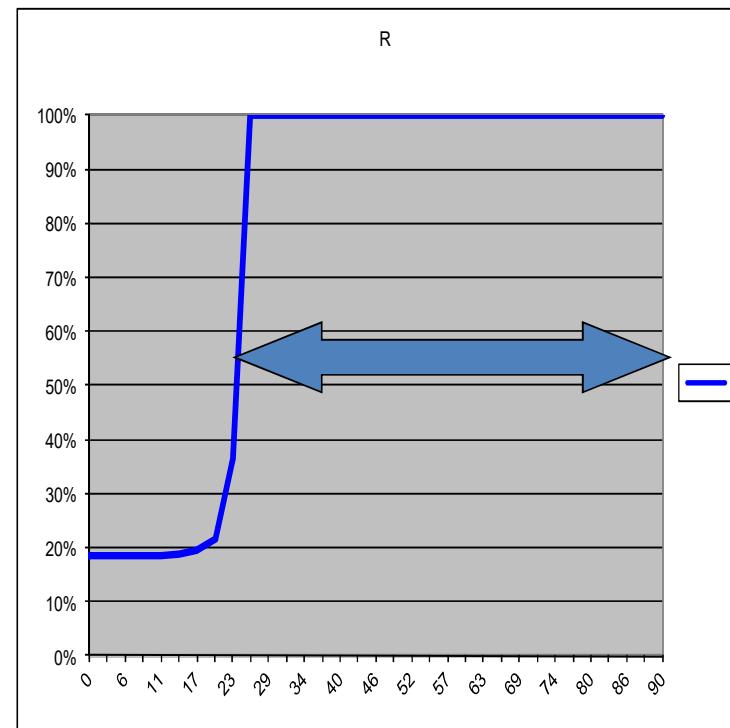
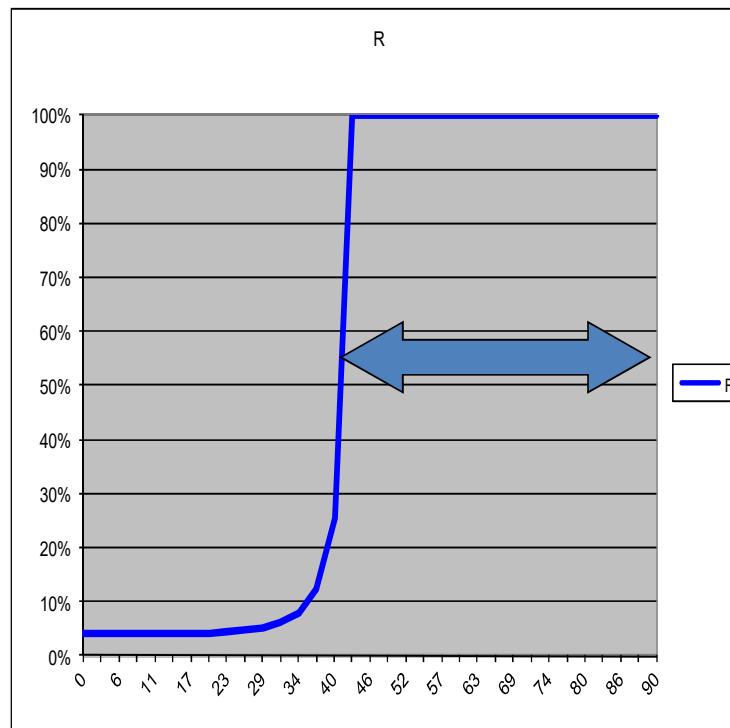
Medi dens → Buit



Aire→Cristall Aire→Diamant

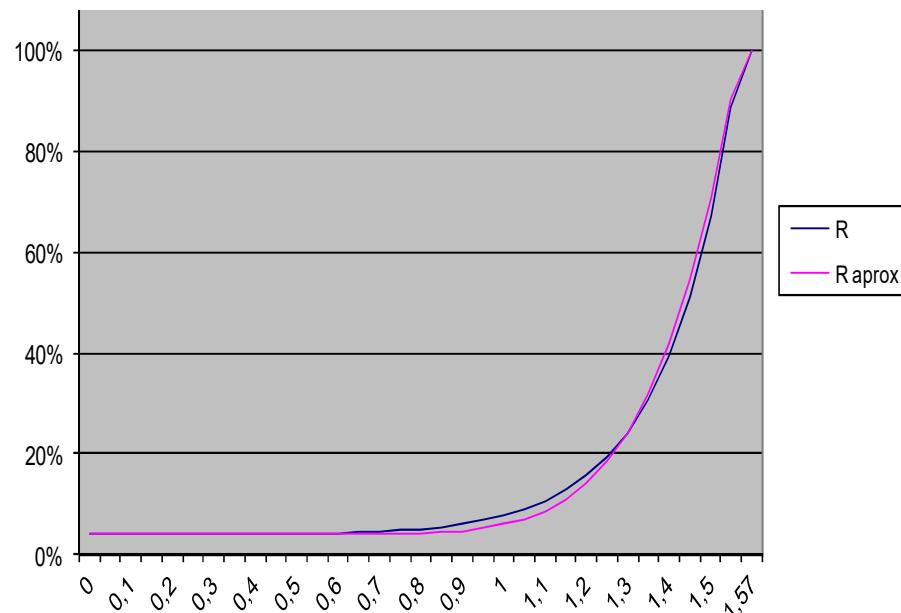


Cristall→Aire Diamant→Aire



Aproximació de Schlick

Evita l'ús de funcions trigonomètriques:



$$R = f + (1 - f)(1 - L \cdot N)^5$$

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

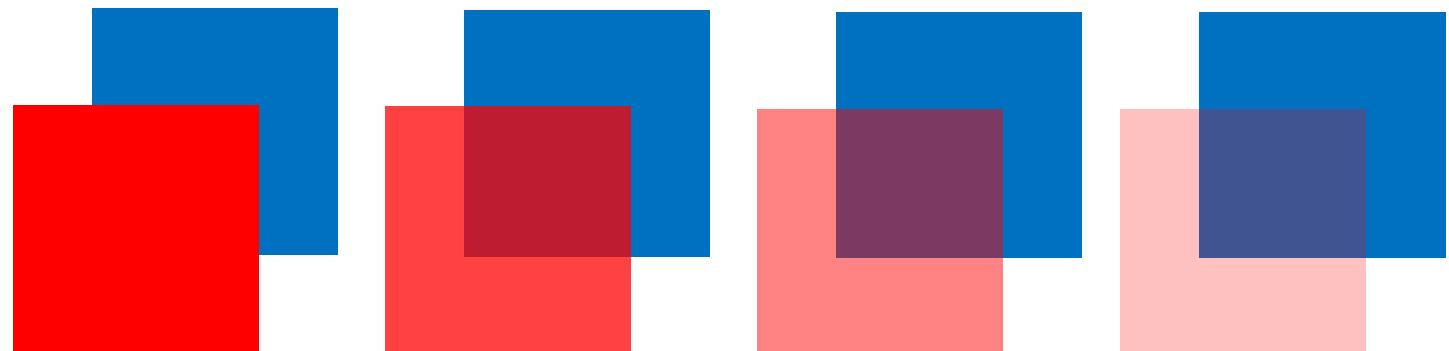
Alpha blending

Maig 2021

Introducció



Introducció

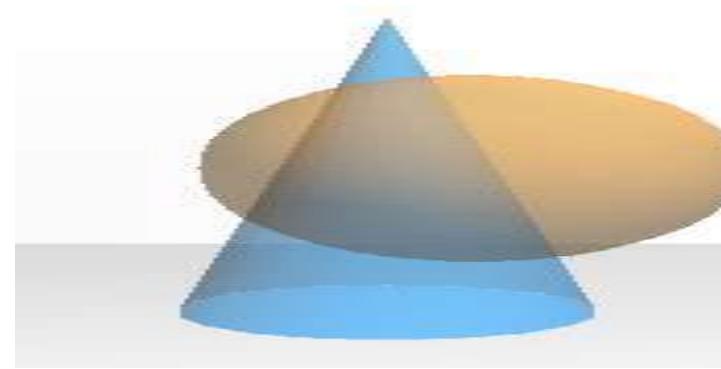
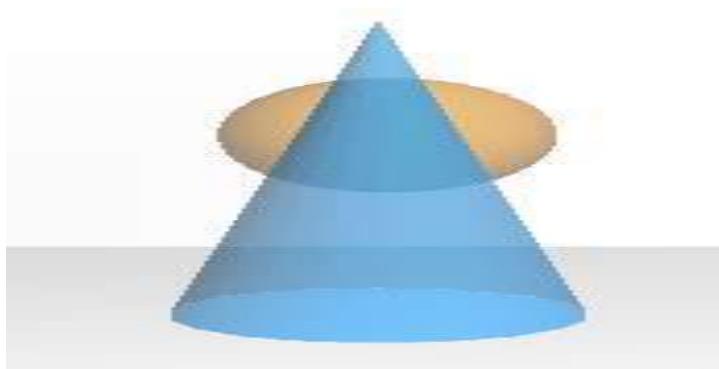


```
glEnable(GL_BLEND);  
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
```

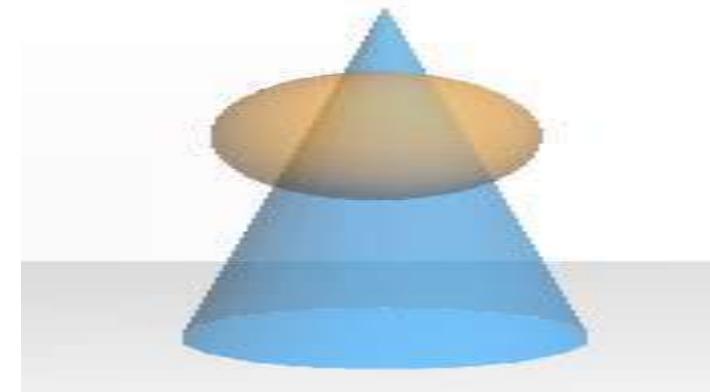
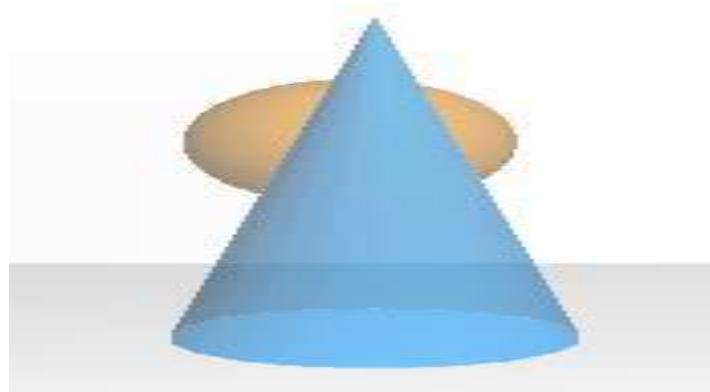
```
glEnable(GL_BLEND);  
glBlendFunc(GL_SRC_ALPHA, GL_ONE)
```

Ordre de pintat dels polígons

Ordenats (esfera-con, con-esfera): resultat “correcte”

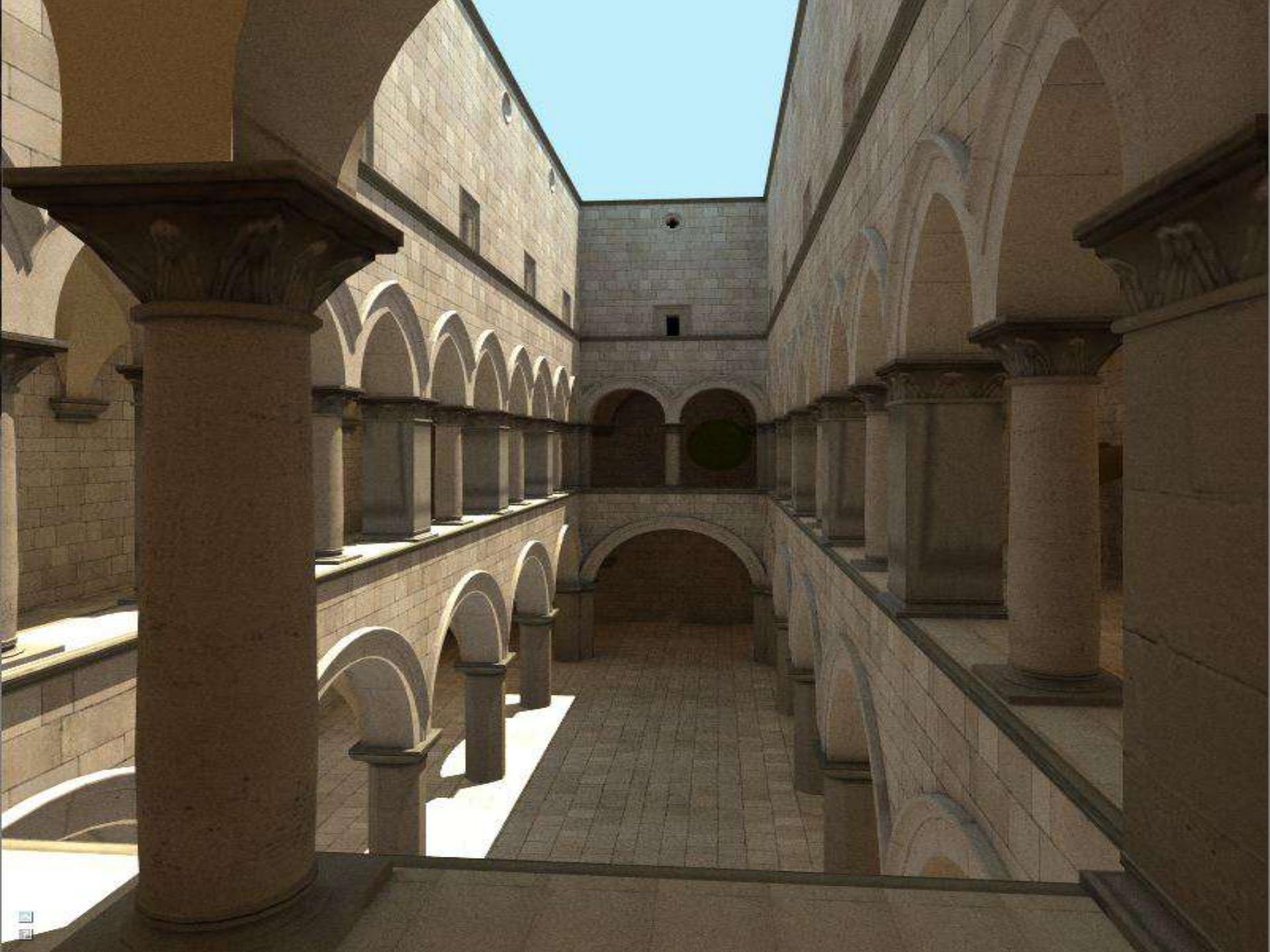


No ordenats (con-esfera): amb/sense Z-buffer

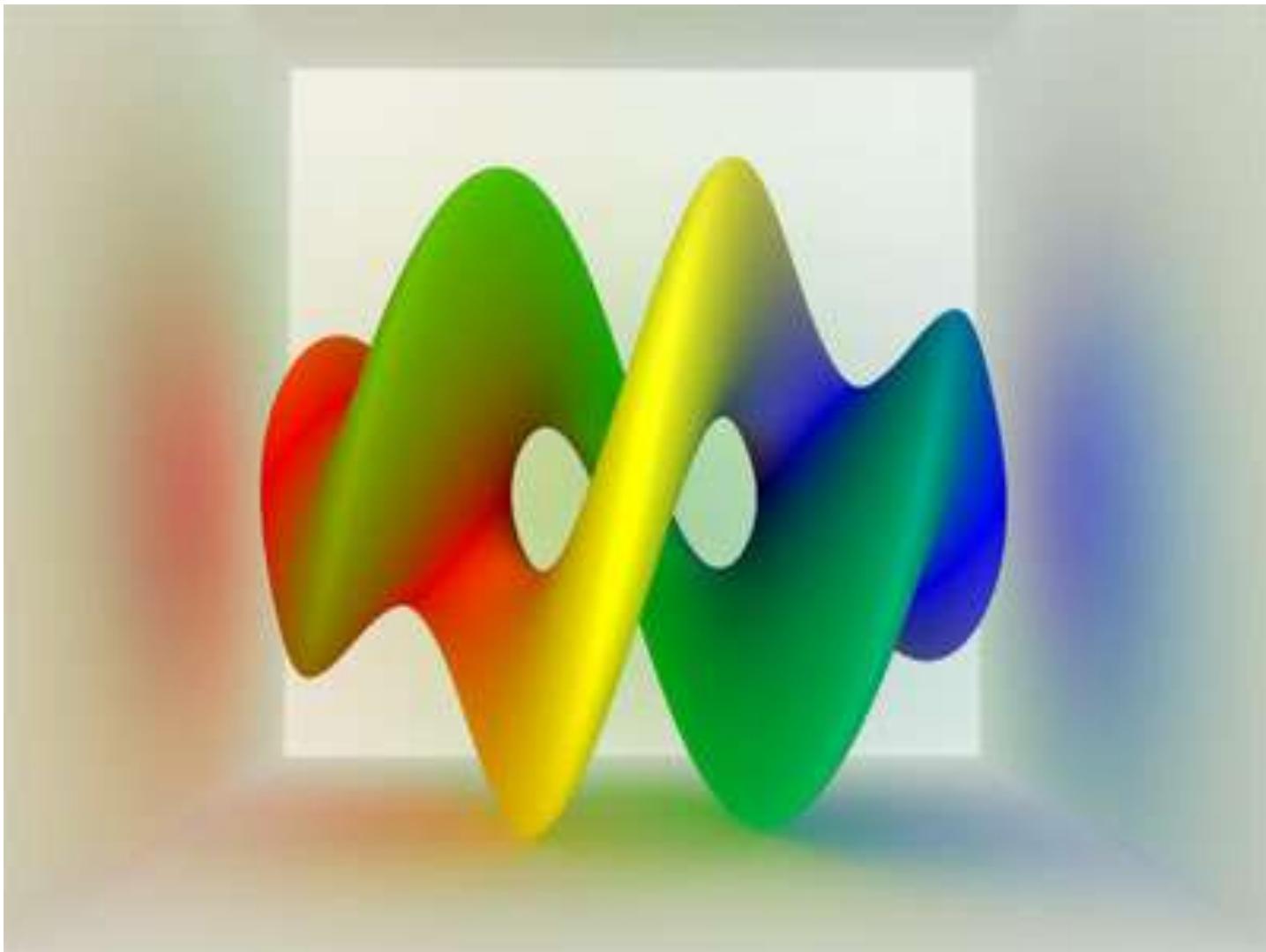


IL·LUMINACIÓ GLOBAL

amb materials de Carlos Andújar i Mario Costa Sousa

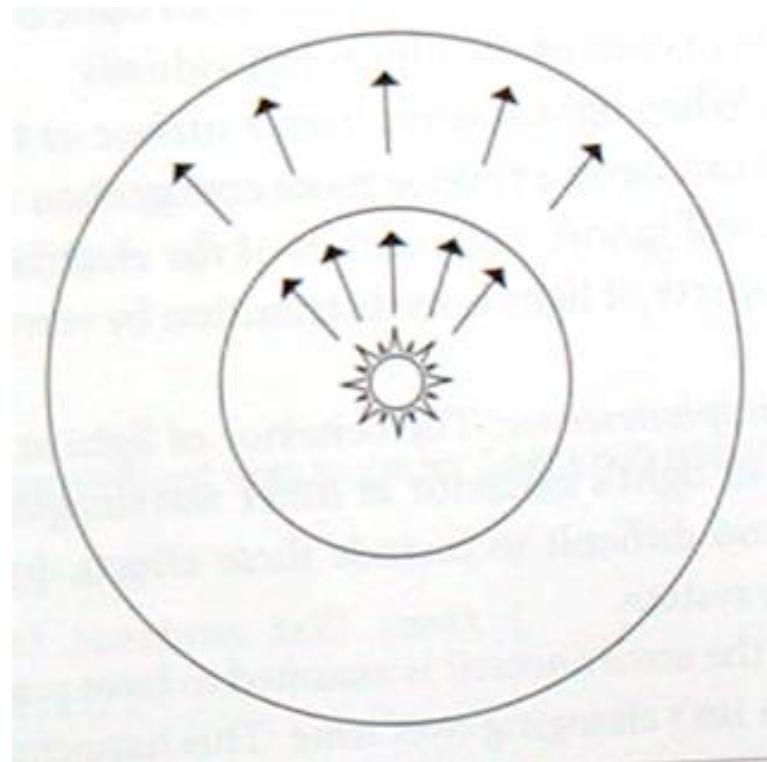


Color bleeding

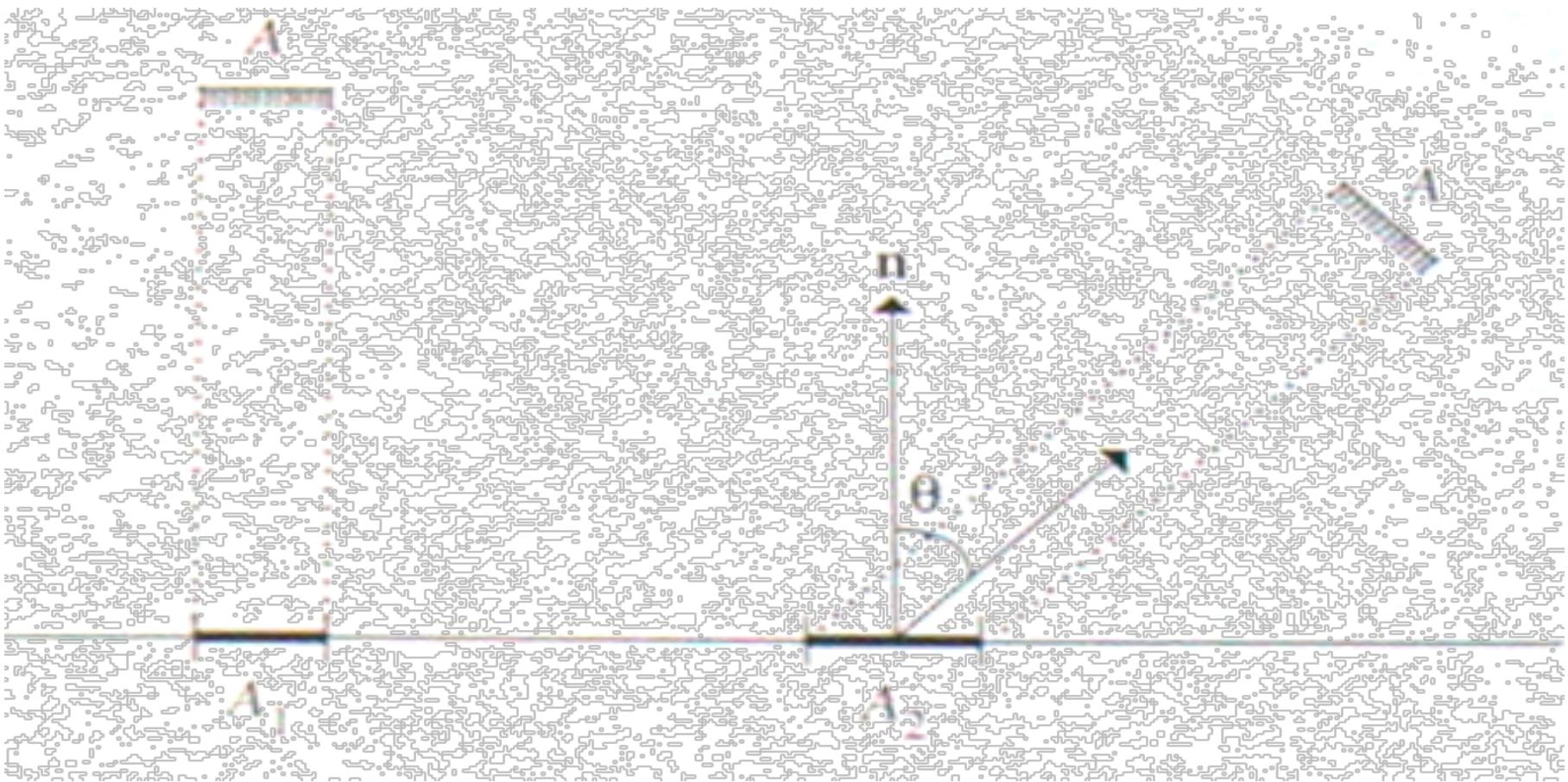


UNITATS RADIOMETRIA

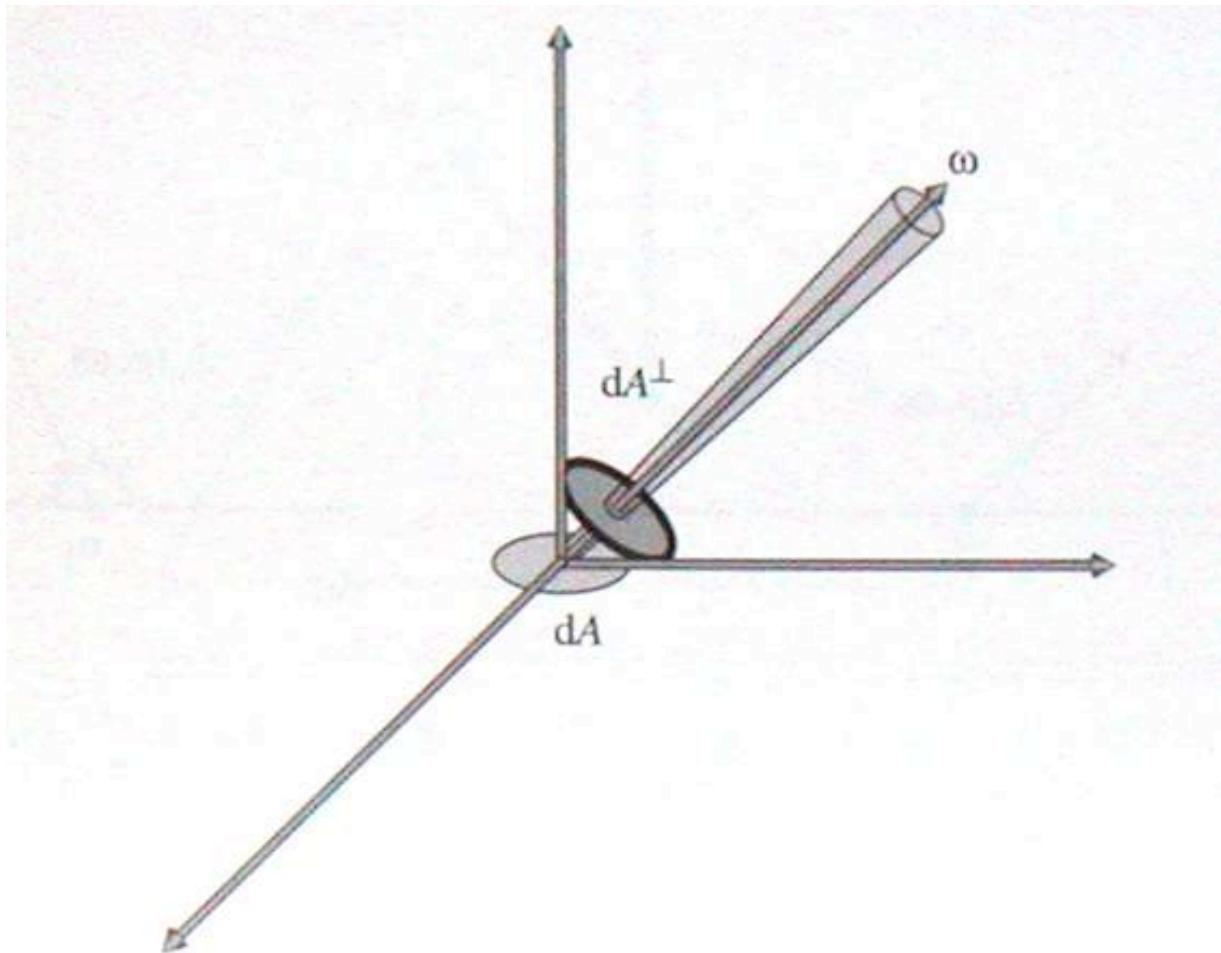
Flux radiant Φ



Irradiància E i Llei de Lambert



Radiància $L(p, \omega)$

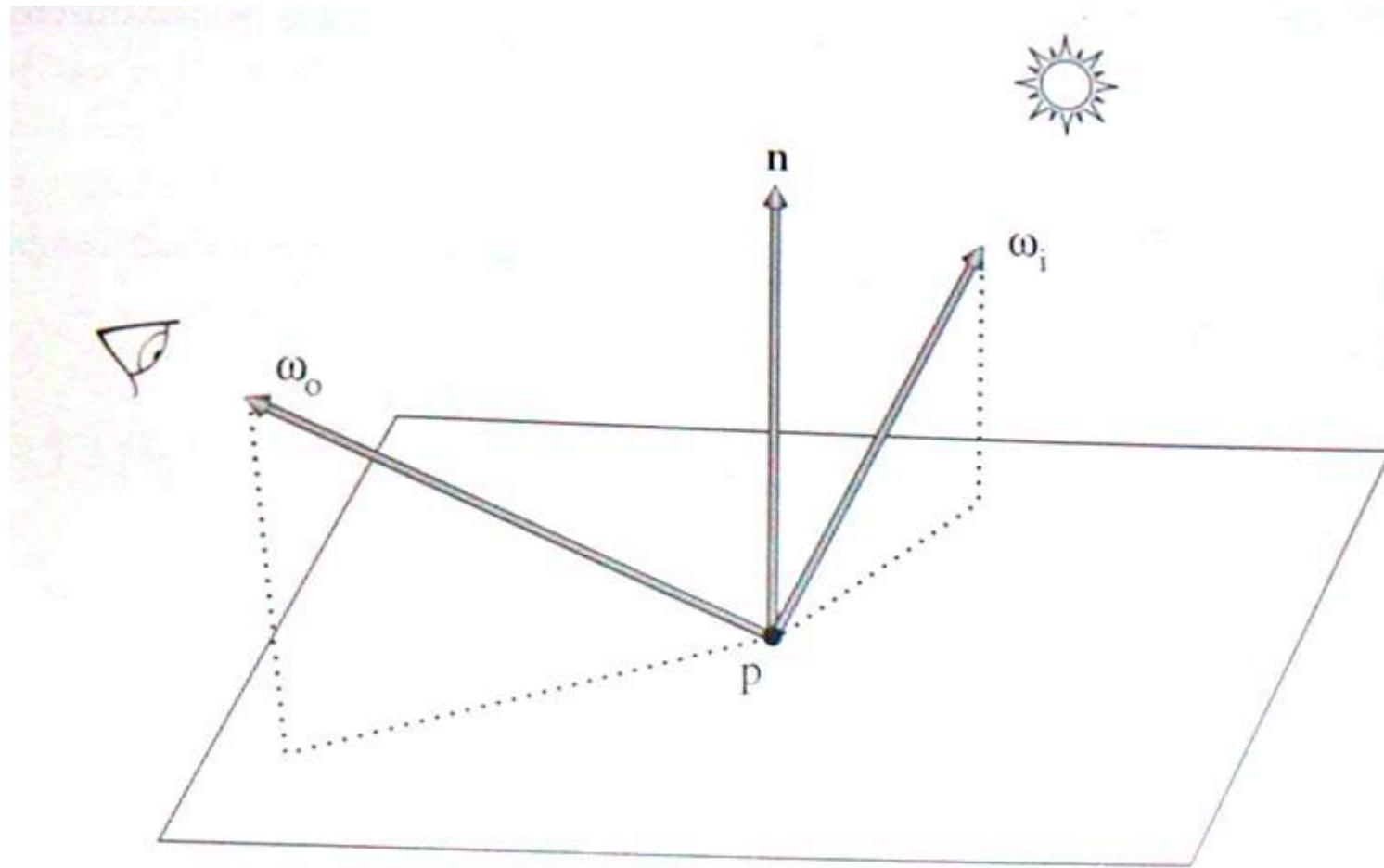


Resum fotometria

Sím.	Radiomet.	Fotometria	Definició	Ús
Φ	Fluxe (W)	Fluxe (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)	Fluxe per unitat d'àrea	Llum que incideix en un punt, des de qualsevol direcció
I	Intensitat (W/sr)	Intensitat (cd=lm/sr)	Fluxe per unitat d'angle sòlid	Distribució direccional d'una llum puntual
L	Radiància W/(sr·m ²)	Luminància (cd/m ²)	Fluxe per unitat d'àrea i unitat d'angle sòlid	Energia que travessa un punt en una determinada direcció

BRDF

$$\text{BRDF } f(p, \omega_o, \omega_i)$$

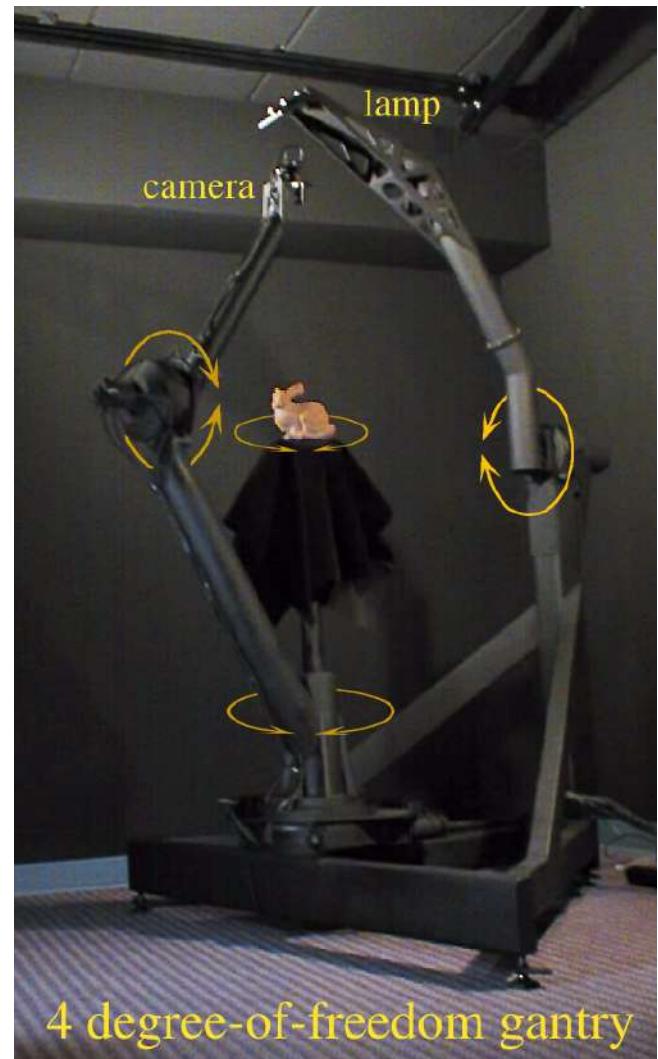


BRDF

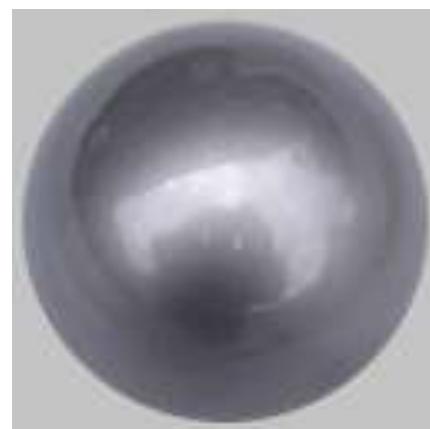


MERL BRDF database

- www.merl.com/brdf/



BRDF de pinturas



Phong vs BRDF mesurat

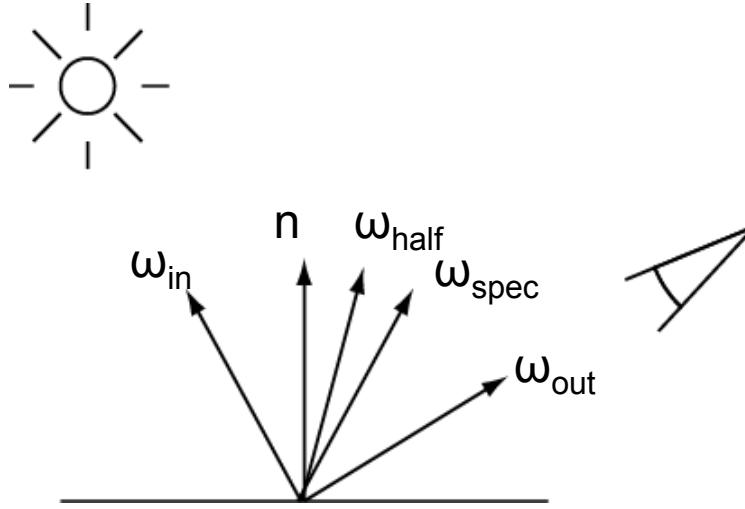


$$L_{\text{out}}(x, \omega_{\text{out}}) = L_{\text{in}}(x, \omega_{\text{in}})^* f(\omega_{\text{in}}, \omega_{\text{out}})^*(\omega_{\text{in}} \cdot n)$$

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{\Omega_x} L(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) \cos(n_x, \Psi) d\omega \Psi$$

- Reciprocitat
 - $f(x, \omega_{\text{in}}, \omega_{\text{out}}) = f(x, \omega_{\text{out}}, \omega_{\text{in}})$
- Conservació de l'energia
 - $\int f(\omega_{\text{in}}, \omega_{\text{out}})^*(\omega_{\text{in}} \cdot n) d\omega_{\text{in}} \leq 1$

BRDFs Analítiques



Lambert:

$$f(\omega_{out}, \omega_{in}) = k_d$$

Blinn:

$$f(\omega_{out}, \omega_{in}) = k_d + k_s (n \cdot \omega_{half})^n$$

Phong

$$f(\omega_{out}, \omega_{in}) = k_d + k_s (\omega_{out} \cdot \omega_{spec})^n$$

Hi ha moltes altres BRDFs analítiques que hom ha proposat, com Cook Torrance-BRDF, Ward-BRDF, ...

Radiosity

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{\Omega_x} L(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) \cos(n_x, \Psi) d\omega \Psi$$

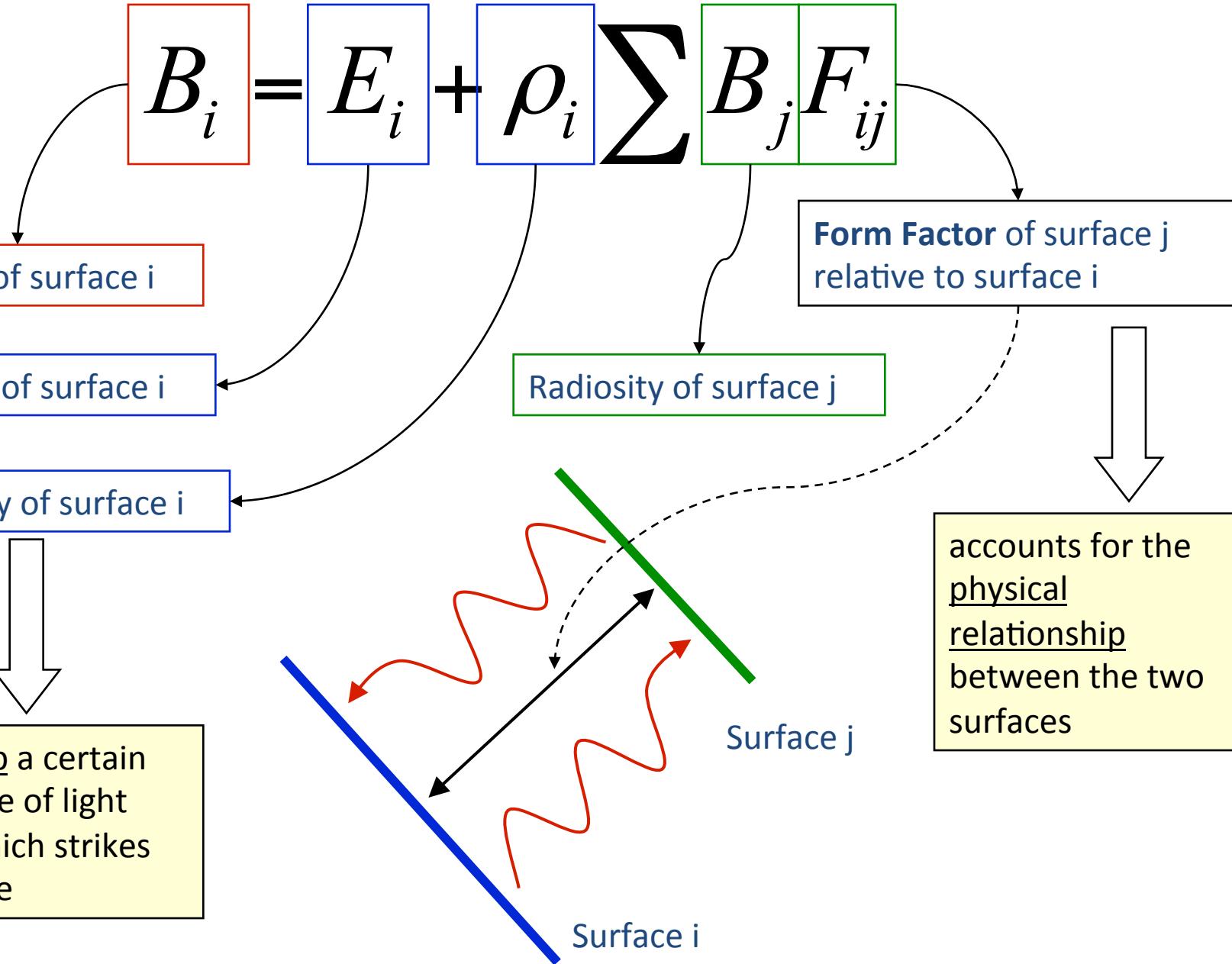
Suposem que totes les superfícies i fonts de llum són difuses, i tenen un comportament uniforme sobre tota la superfície.

Subdividim en pedaços per a mantenir l'uniformitat

Obtenim una versió discretitzada de l'equació de rendering

$$B_i = E_i + \rho_i \sum B_j F_{ij}$$

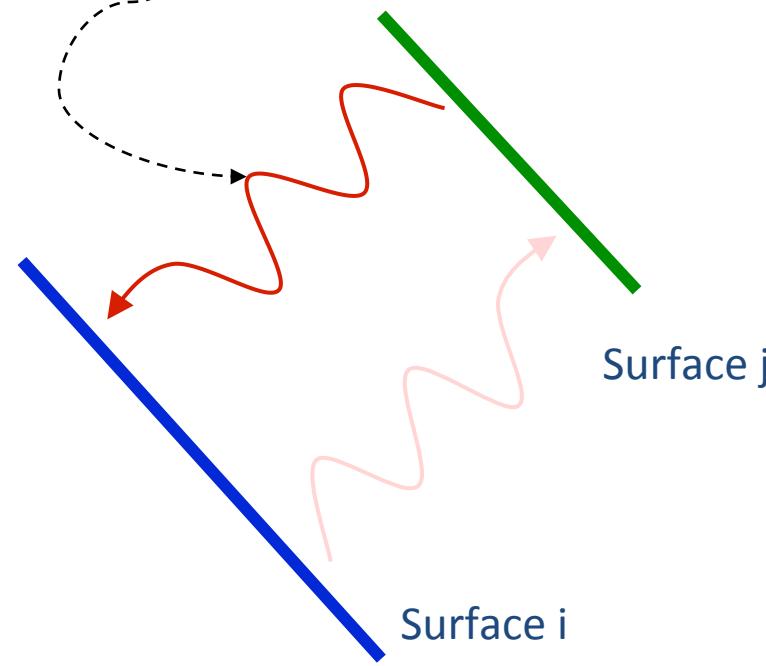
The Radiosity Equation



The Radiosity Equation

$$B_i = E_i + \rho_i \sum B_j F_{ij}$$

Energy reaching surface i from other surfaces



The Radiosity Equation

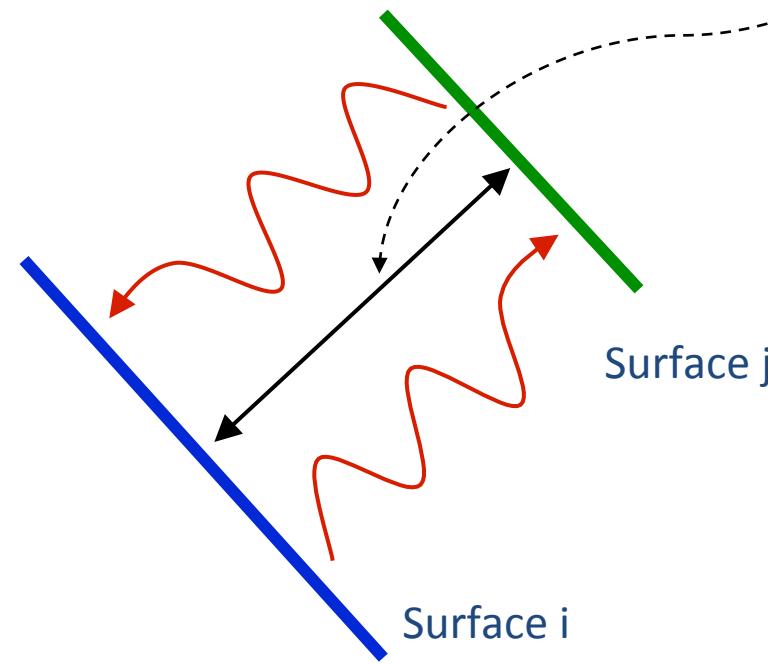
$$B_i = E_i + \rho_i \sum B_j F_{ij}$$

Energy reaching
surface i from other
surfaces

Form Factor of surface j
relative to surface i

Radiosity of surface j

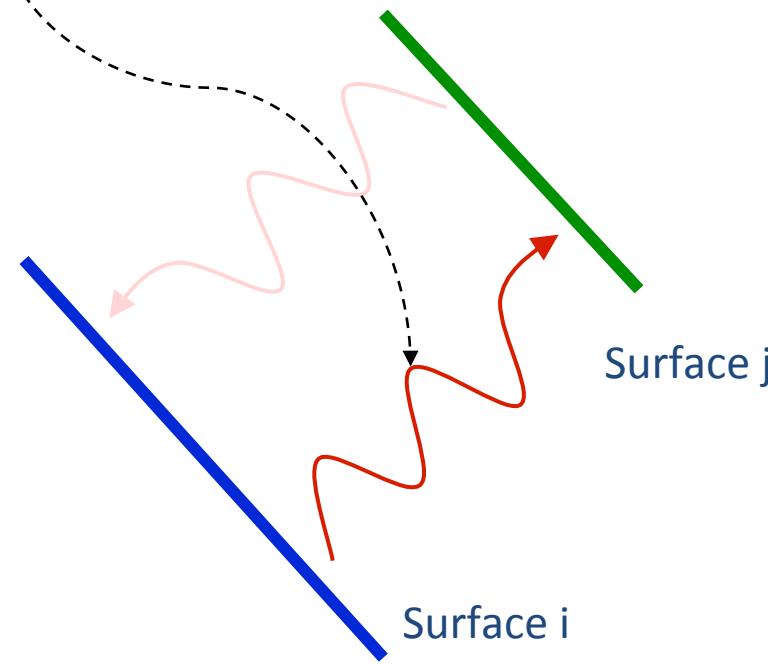
accounts for the
physical
relationship
between the two
surfaces



The Radiosity Equation

$$B_i = E_i + \rho_i \sum B_j F_{ij}$$

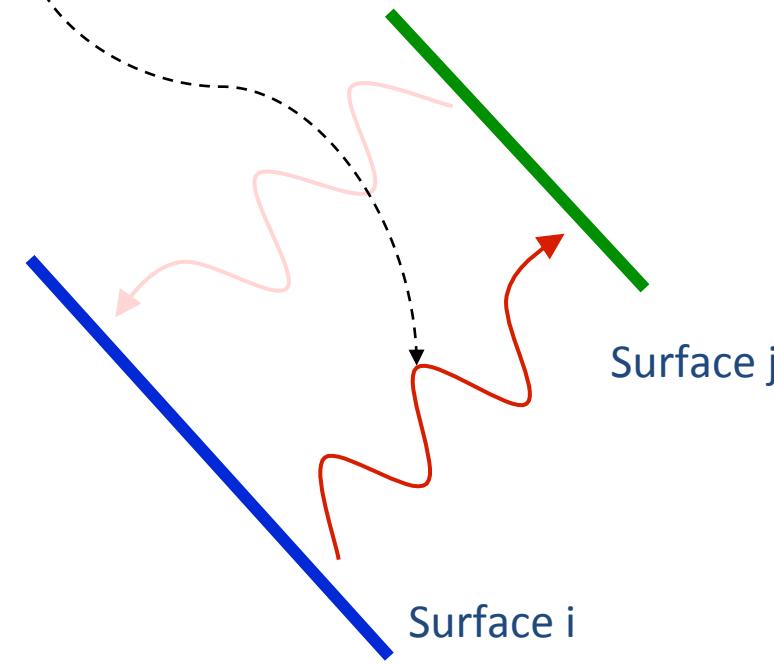
Energy emitted by surface i



The Radiosity Equation

$$B_i = E_i + \rho_i \sum B_j F_{ij}$$

Energy reflected by surface i



The Radiosity Equation

Energy reflected by surface i

$$B_i = E_i + \rho_i \sum B_j F_{ij}$$

Reflectivity of surface i

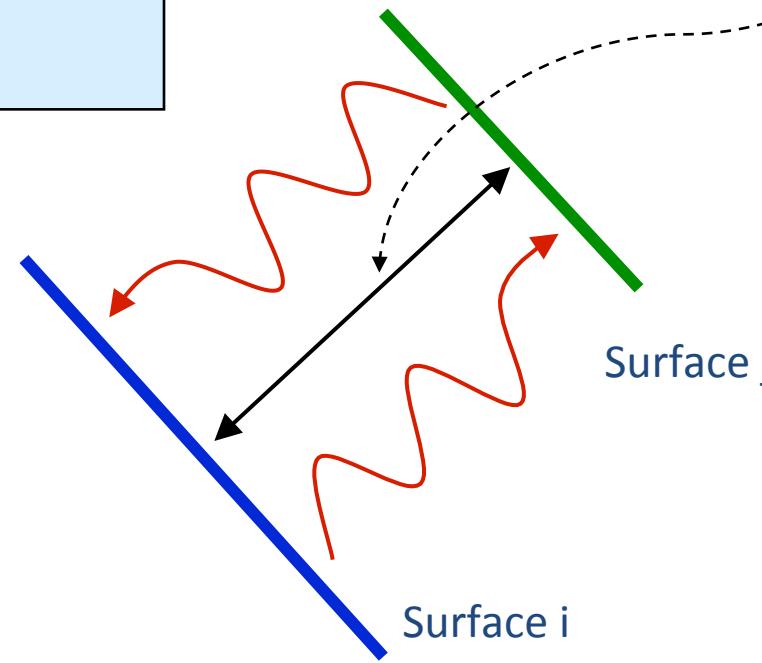
Form Factor of surface j
relative to surface i

Energy reflected by surface i =
Reflectivity of surface i *
Energy reaching surface i
from other surfaces

Radiosity of surface j

Reflectivity
will absorb a certain
percentage of light
energy which strikes
the surface

Form Factor
accounts for the
physical
relationship
between the two
surfaces

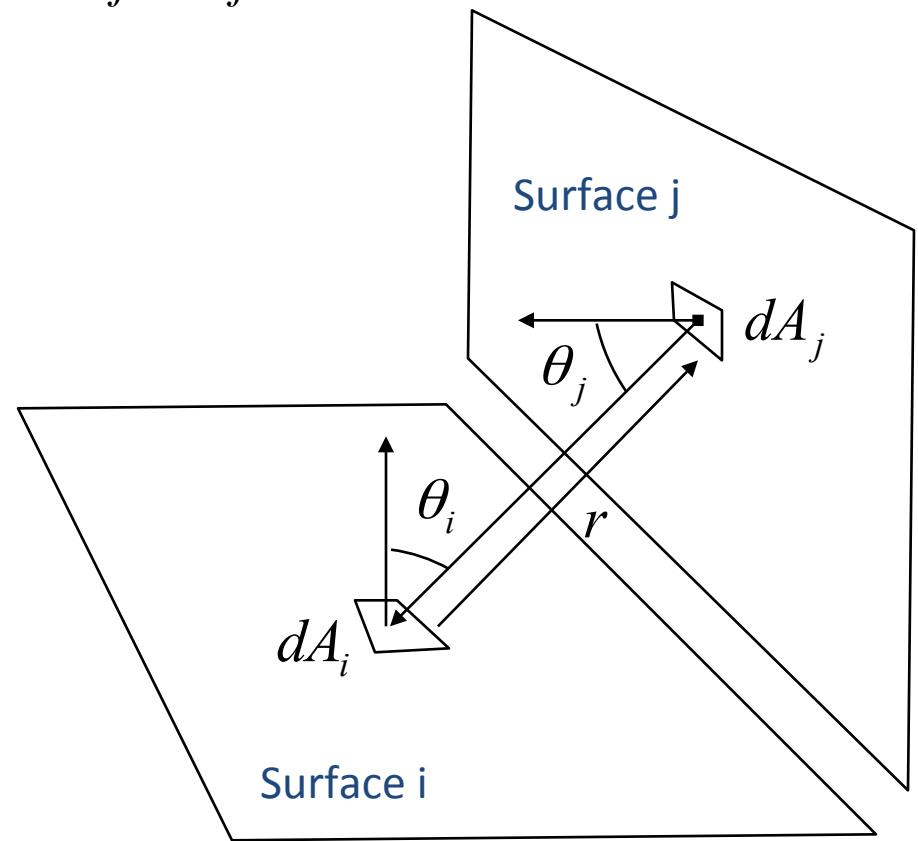


Càcul dels factors de forma

$$F_{ij} = \frac{1}{A_i} \iint_{A_i} \iint_{A_j} \frac{\cos \theta_i \cos \theta_j}{\pi r^2} V_{ij} dA_j dA_i$$

Area integral
over surface *i*

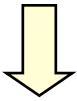
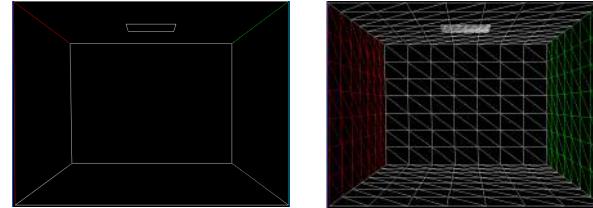
Area integral
over surface *j*



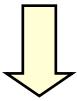
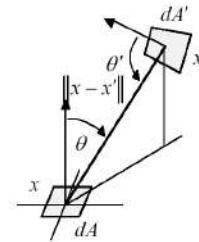
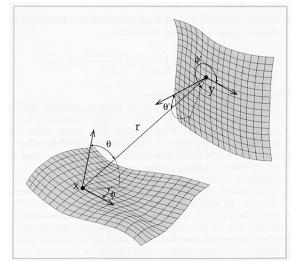
Classic Radiosity Algorithm



Mesh Surfaces into Elements

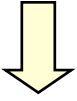


Compute Form Factors
Between Elements

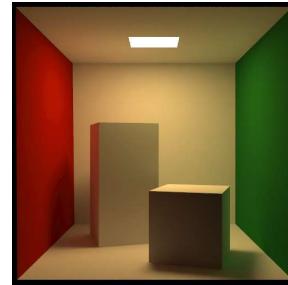


Solve Linear System
for Radiosities

$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_n \end{bmatrix} + \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_n \end{bmatrix}$$



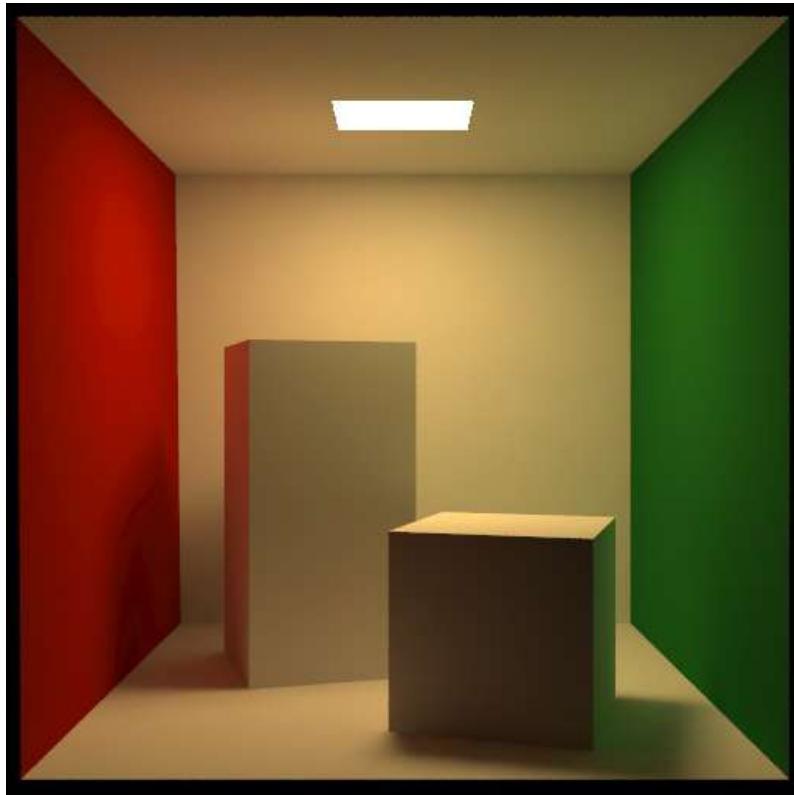
Reconstruct and Display
Solution



Radiosity com a sistema d'equacions

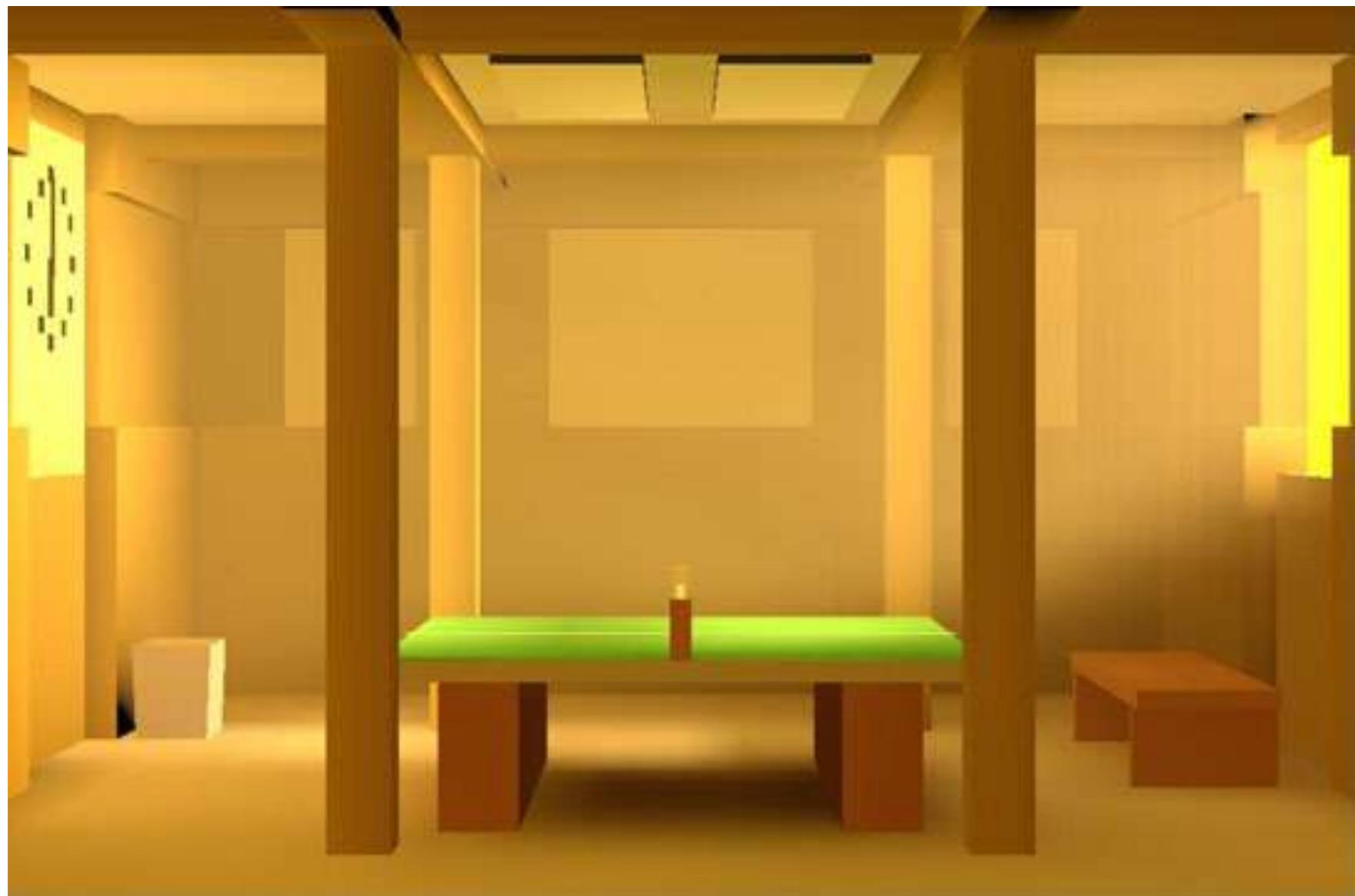
$$\begin{bmatrix} 1 - \rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 - \rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1 - \rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

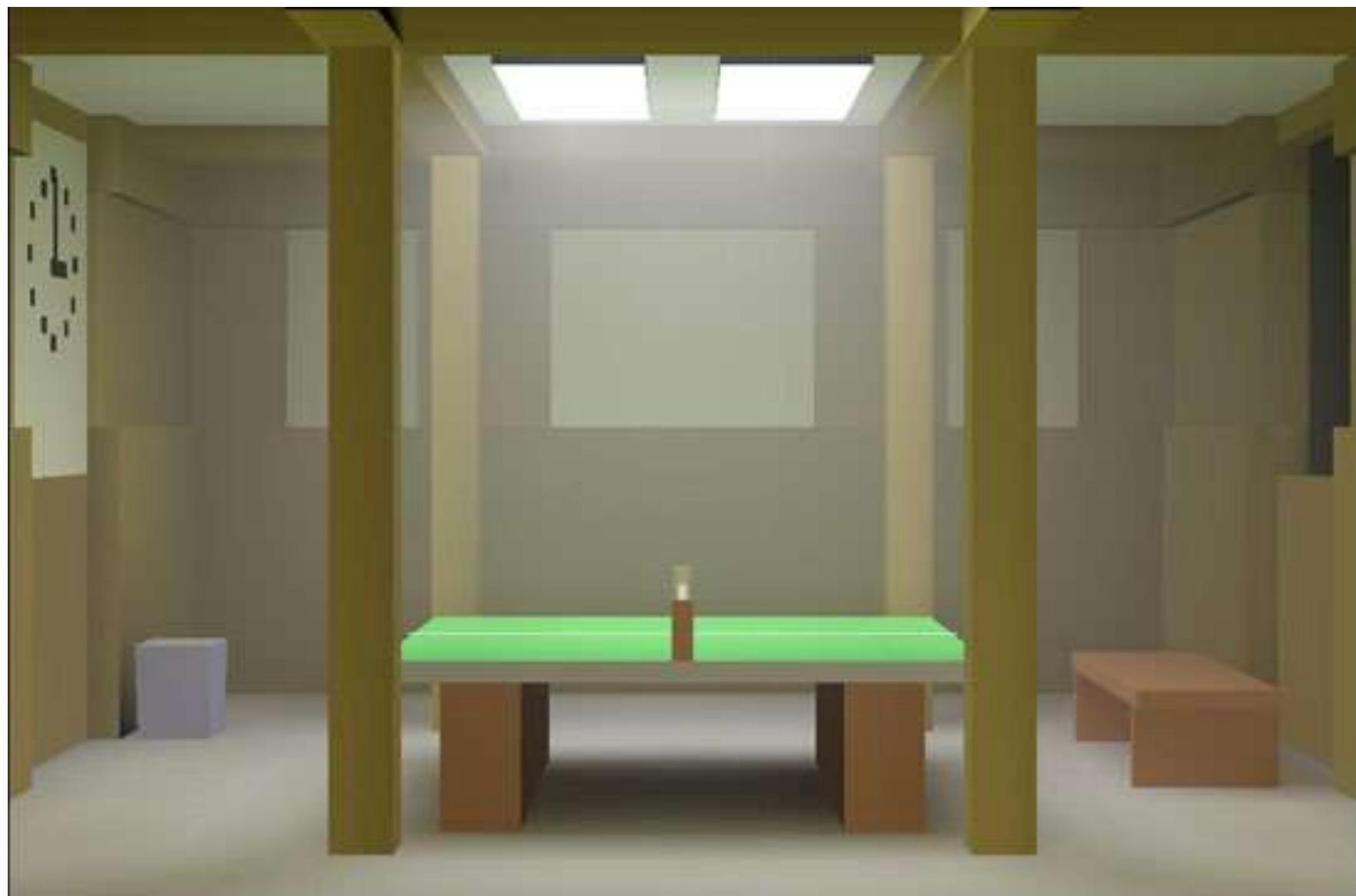
Resultats







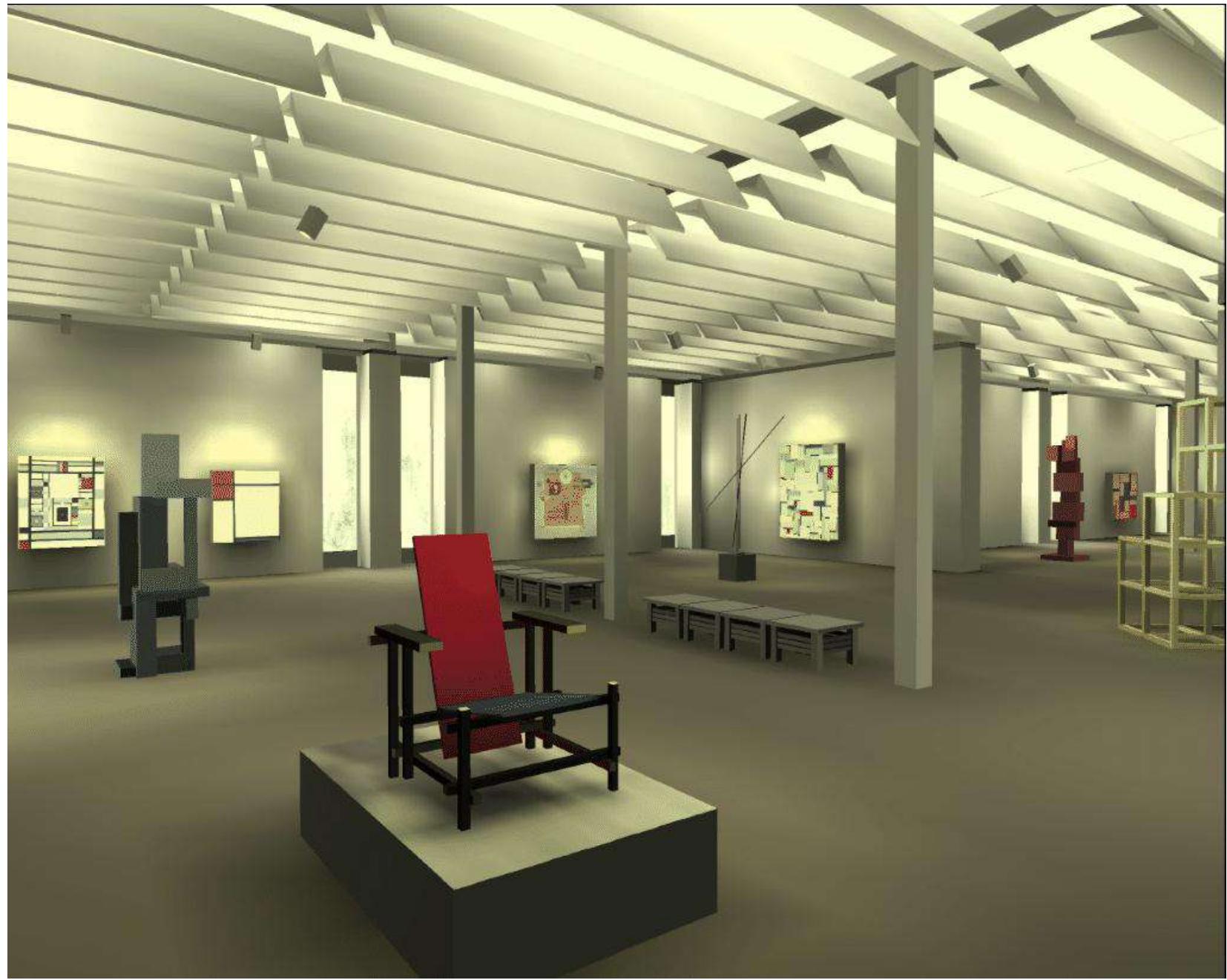


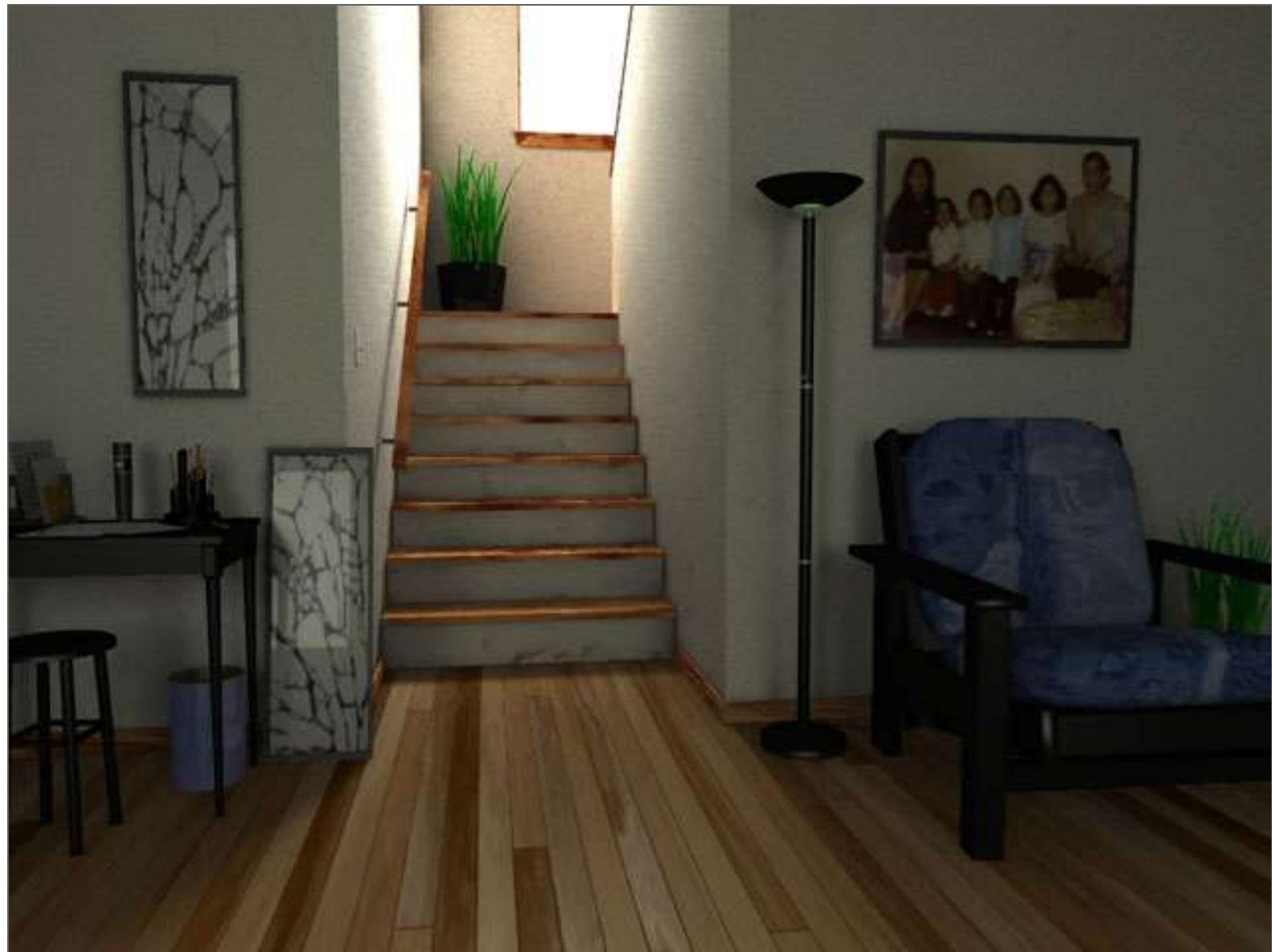














Oclusió ambient

Pràctica de ray-tracing

© Professors VA

MOVING Group
Universitat Politècnica
de Catalunya



Índex

- Concepte
- Càcul
- Resultats

Concepte

- L'occlusió ambient (*ambient occlusion*) és una forma relativament poc costosa de calcular una il·luminació realista
- La idea principal és modular la il·luminació que arriba a un punt de l'escena com a part de la il·luminació ambient
- Existeixen formes d'accelerar els càculs utilitzant GPUs o amb preprocés

Concepte

- Idea:
 - Assumim que tenim un entorn amb molta il·luminació de l'ambient (p. ex. una escena a cel obert)
 - Volem *disminuir* la il·luminació que arriba a un cert punt en funció dels objectes que *tapen* la “font de llum”
 - Cal mesurar la porció d'entorn no *tapada*

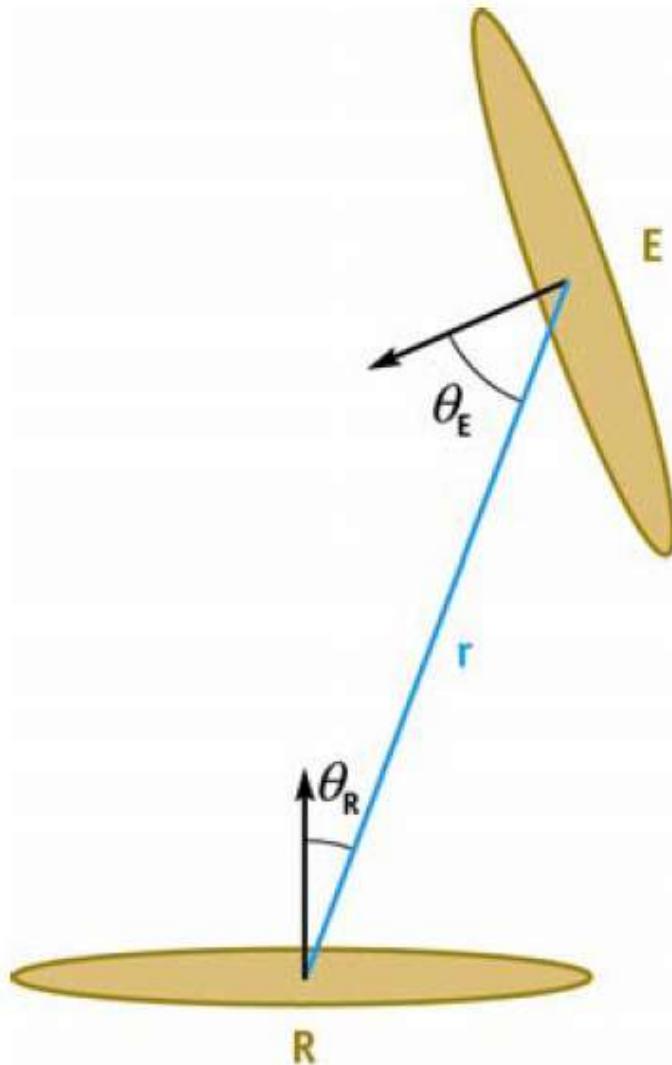
Concepte

- Idea:
 - Mesura de la porció d'entorn no *tapada*:
 - Cal buscar *quins* objectes estan a l'entorn del punt que volem calcular
 - Mesurar el que ocupen
 - Modular la il·luminació en funció de l'espai ocupat
 - El procés és lent però es pot aproximar

- Concepte
- Càcul
- Resultats

Concepte

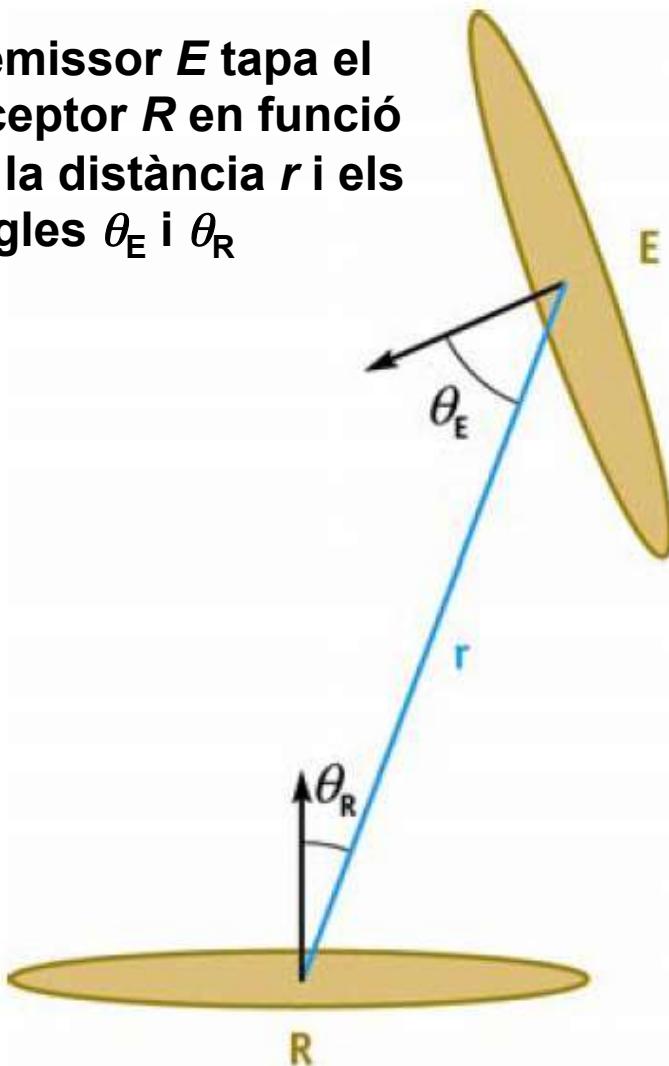
L'emissor E tapa el receptor R en funció de la distància r i els angles θ_E i θ_R



Concepte

- El percentatge de l'hemisferi *tapat* per sobre d'un punt es pot mesurar com l'angle sòlid
- Equivalent al factor de forma de radiositat amb visibilitat 100%

L'emissor E tapa el receptor R en funció de la distància r i els angles θ_E i θ_R



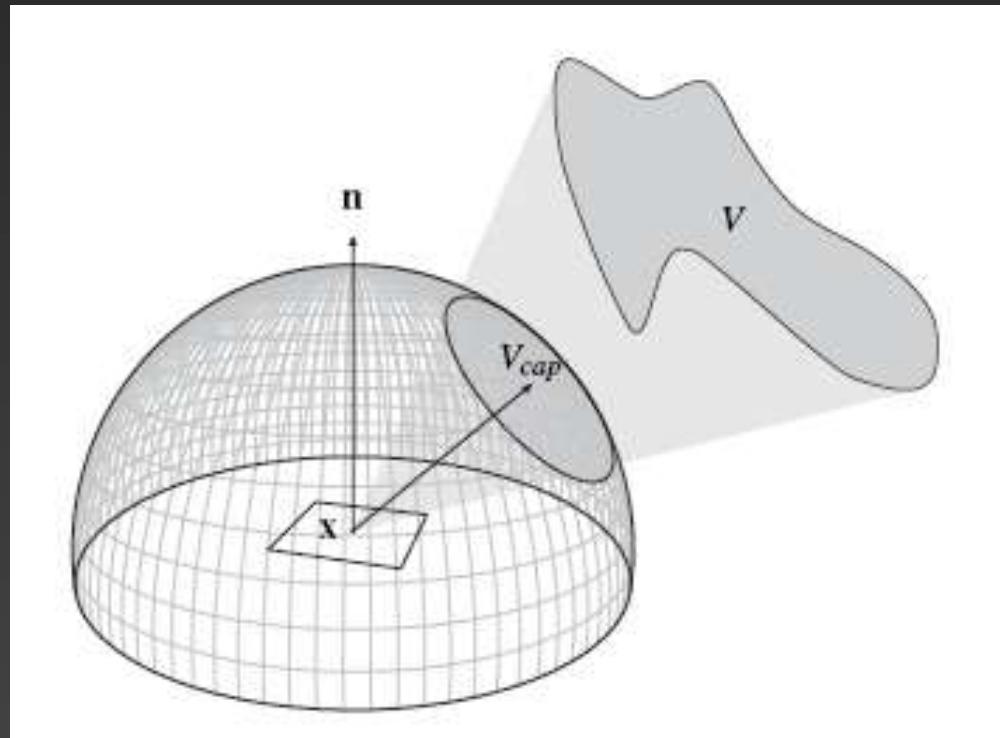
Concepte

- Formalment, el càlcul s'ha de fer per tot l'hemisferi:

Funció de visibilitat

$$\tilde{A}(x, n) = \frac{1}{\pi} \int_{\omega \in \Omega} V_{cap}(x, \omega) [\omega \cdot n] d\omega$$

hemisferi



Càcul

- Es pot calcular l'occlusió en un punt sumant factors de forma
- Algoritme

occlusió = 0

per a cada element E

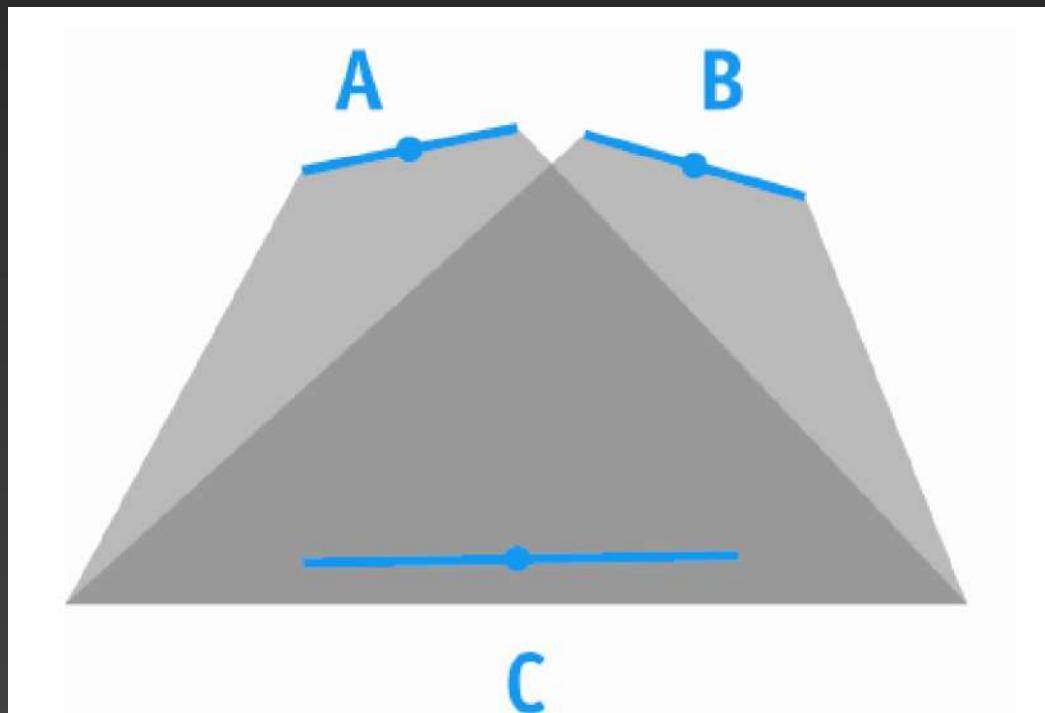
occlusió += factorDeForma (E)

iluminació -= occlusió

Càlcul

- Concepte
- Càlcul
- Resultats

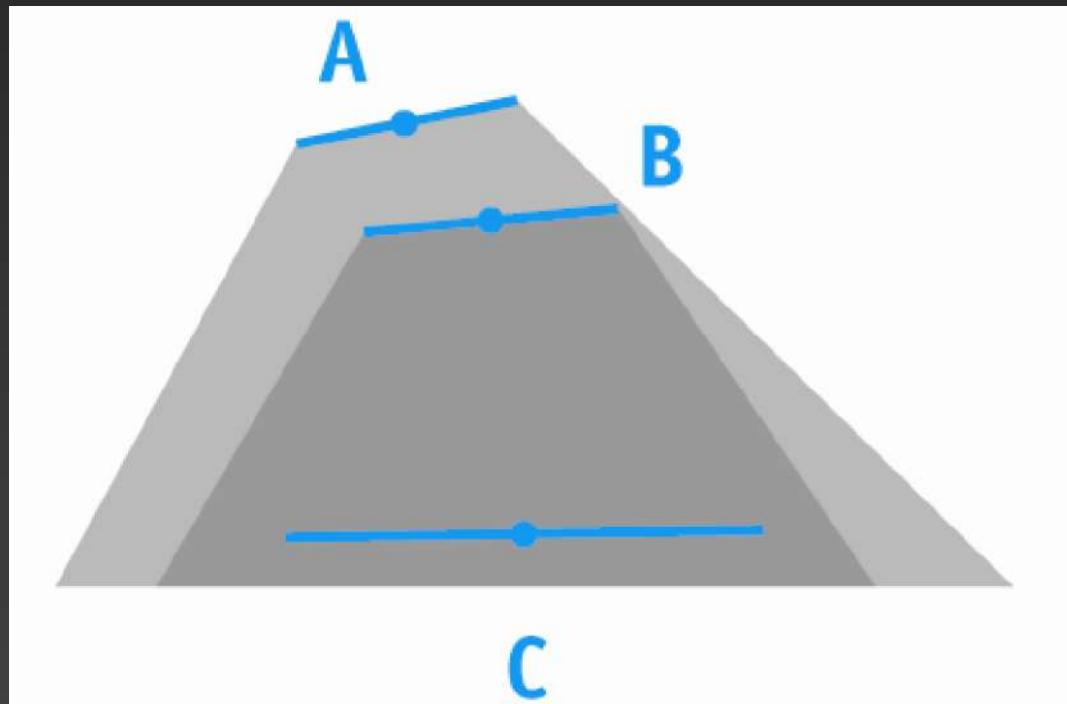
- Cas general:
 - A i B oclusors
 - C en ombra
 - OK



Càlcul

- Concepte
- Càlcul
- Resultats

- Cas problemàtic:
 - A i B oclusors
 - C en ombra dues vegades
 - NO OK



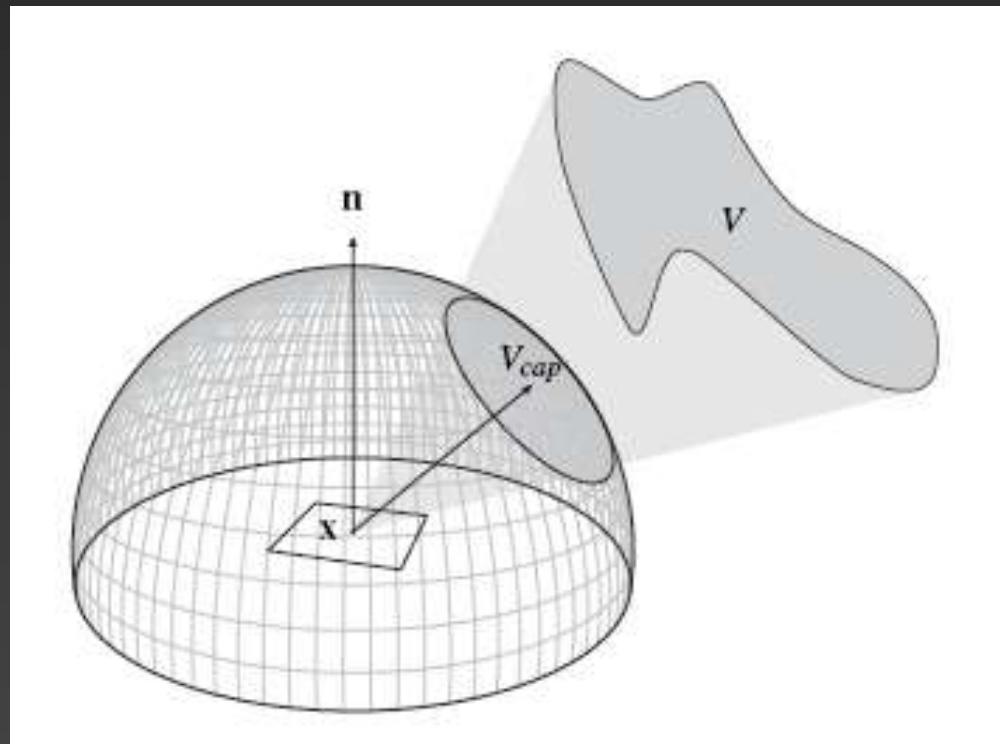
Càcul

- Formalment, el càlcul s'ha de fer per tot l'hemisferi:

Funció de visibilitat

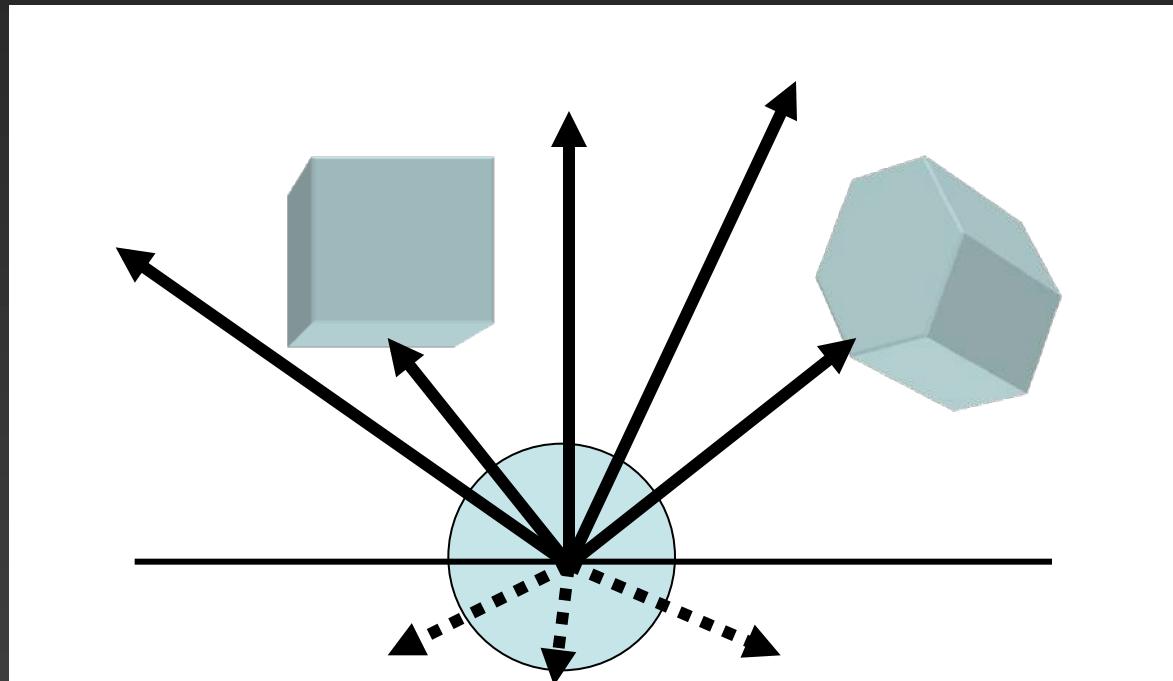
$$\tilde{A}(x, n) = \frac{1}{\pi} \int_{\omega \in \Omega} V_{cap}(x, \omega) \lfloor \omega \cdot n \rfloor d\omega$$

hemisferi



Càcul

- Formalment, el càlcul s'ha de fer per tot l'hemisferi
 - Però podem aproximar per un subconjunt de rajos:
 - Idealment, esbiaixats cap a la normal



Càcul

- El càlcul pot ser:

occlusió = 0

per a cada raig R

occlusió +=

factor(*distIntersecció, angleRaig*)

iluminació -= occlusió

Càcul

- El *factor* depèn de:
 - La distància a la intersecció:
 - Com més distant menys oclusió
 - L'angle amb la normal del punt:
 - Com més proper a la normal, més oclusió

```
oclosió = 0  
per a cada raig R  
    oclosió +=  
        factor(dist,  
                angleRaig)  
    iluminació -= oclosió
```

- Concepte
- Càlcul
- Resultats

Resultats

- Diferents tècniques:





1



Oclusió ambient

Pràctica de ray-tracing

© Professors VA

MOVING Group
Universitat Politècnica
de Catalunya



Ray Tracing

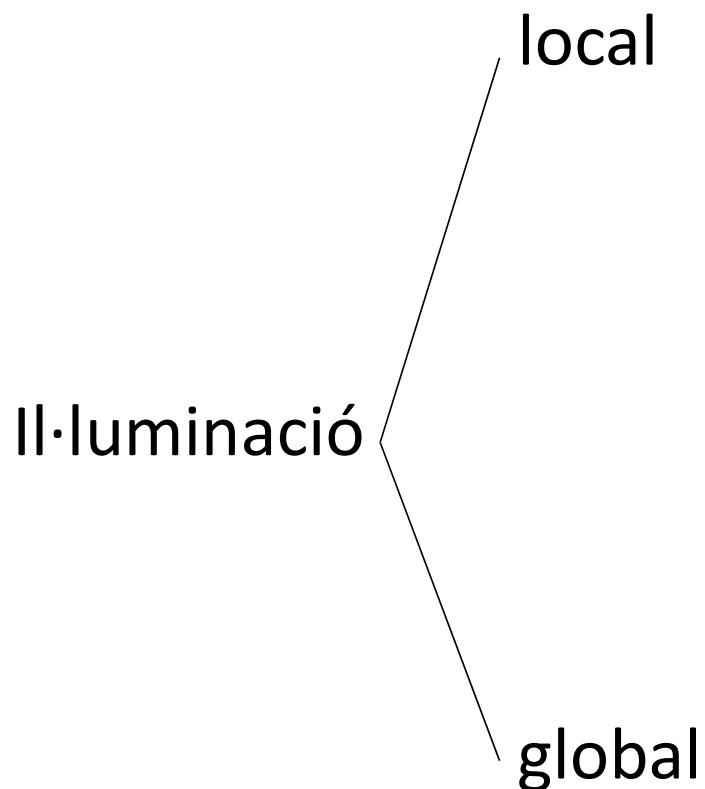
Carlos Andujar

Des 2022



Il·luminació global

Il·luminació local / global





Pabelló de Mies van der Rohe (foto: Ashley Pomeroy)

Llum directa



Pabelló de Mies van der Rohe (model: eMirage, Claudio Andres, Hamza Cheggour)

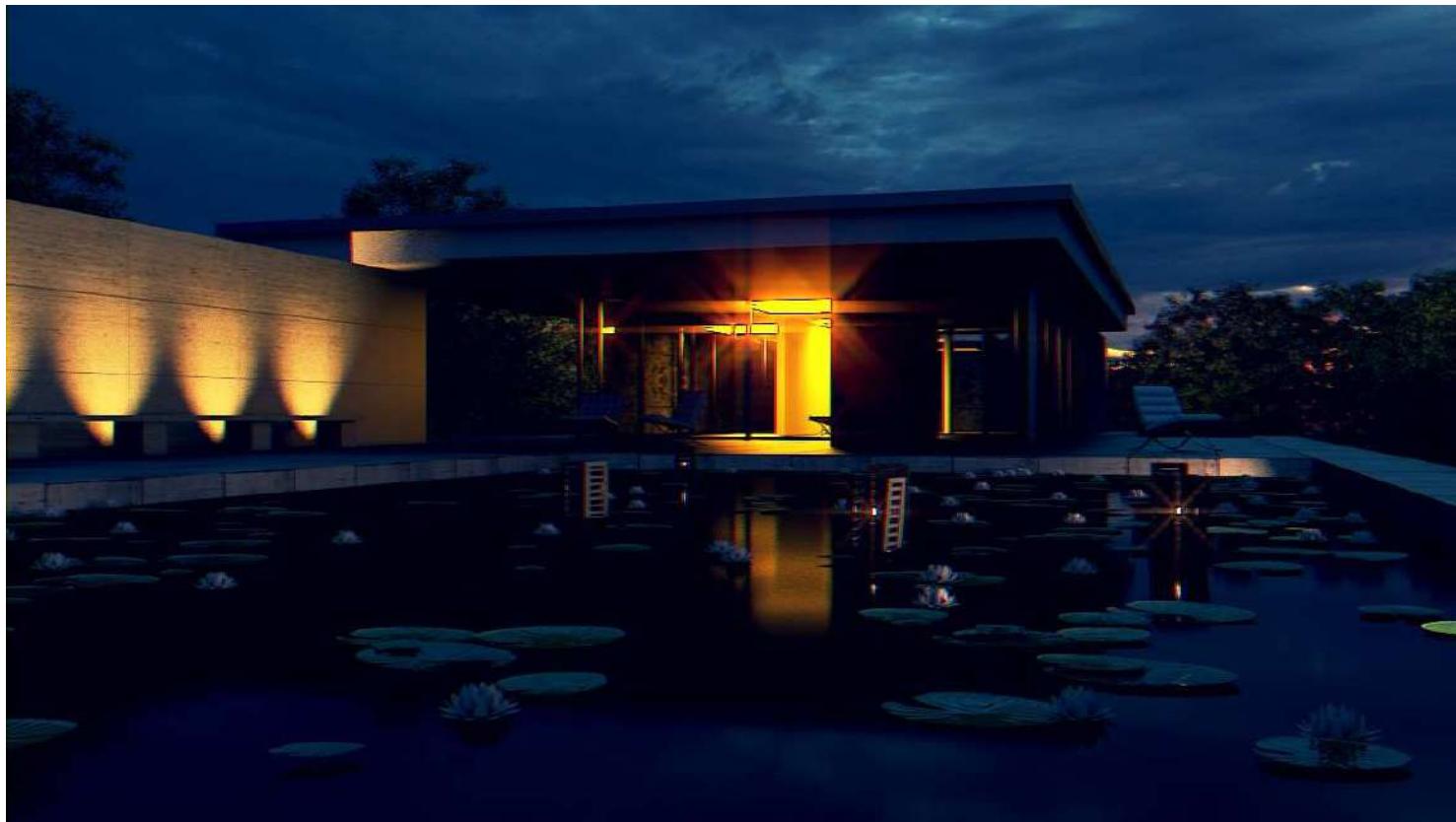
Llum directa + indirecta



Pabelló de Mies van der Rohe (model: eMirage, Claudio Andres, Hamza Cheggour)



Llum directa



Pabelló de Mies van der Rohe (model: eMirage, Claudio Andres, Hamza Cheggour)

Llum directa + indirecta



Pabelló de Mies van der Rohe (model: eMirage, Claudio Andres, Hamza Cheggour)

Llum directa



Pabelló de Mies van der Rohe (model: eMirage, Claudio Andres, Hamza Cheggour)

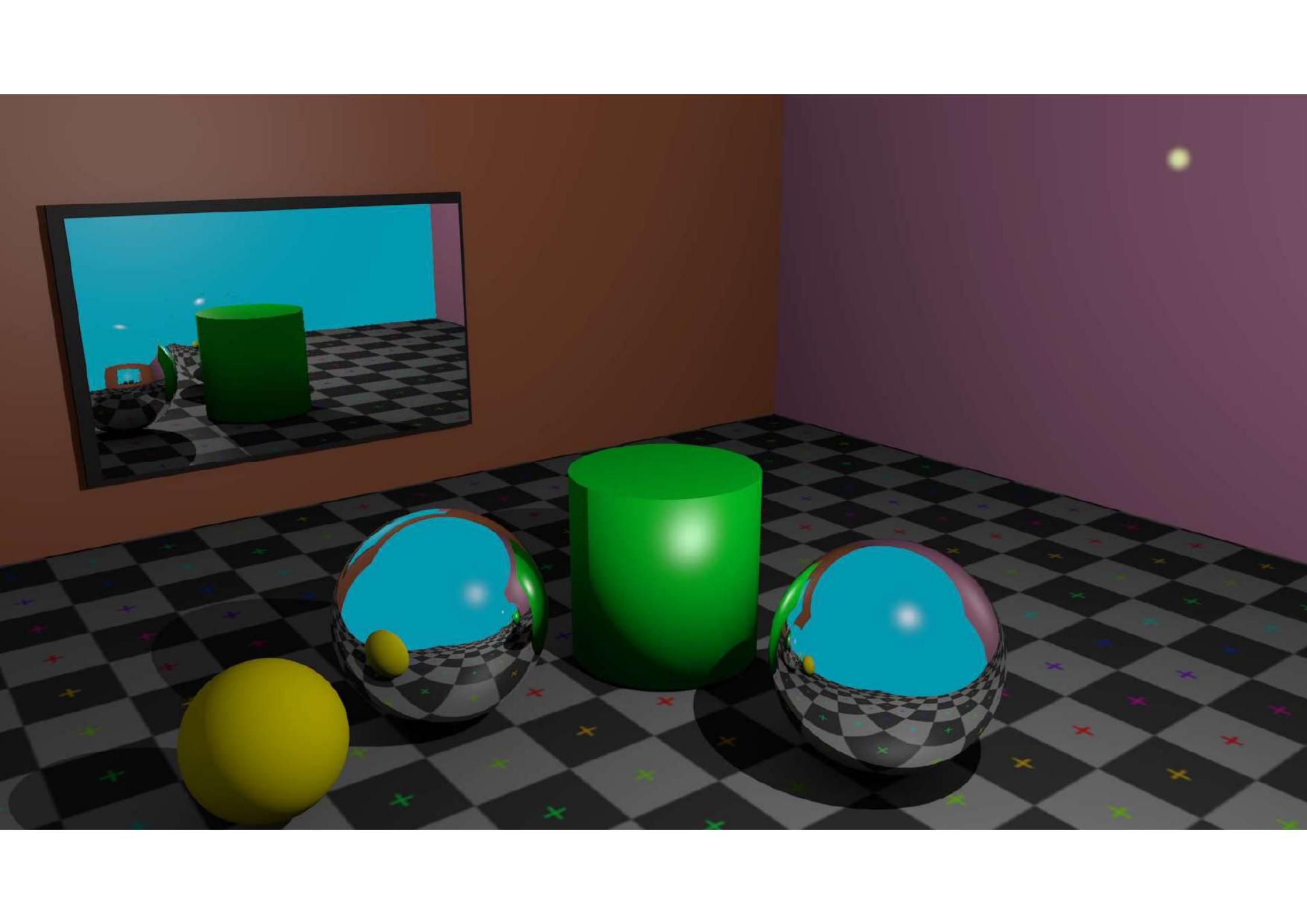
Llum directa + indirecta

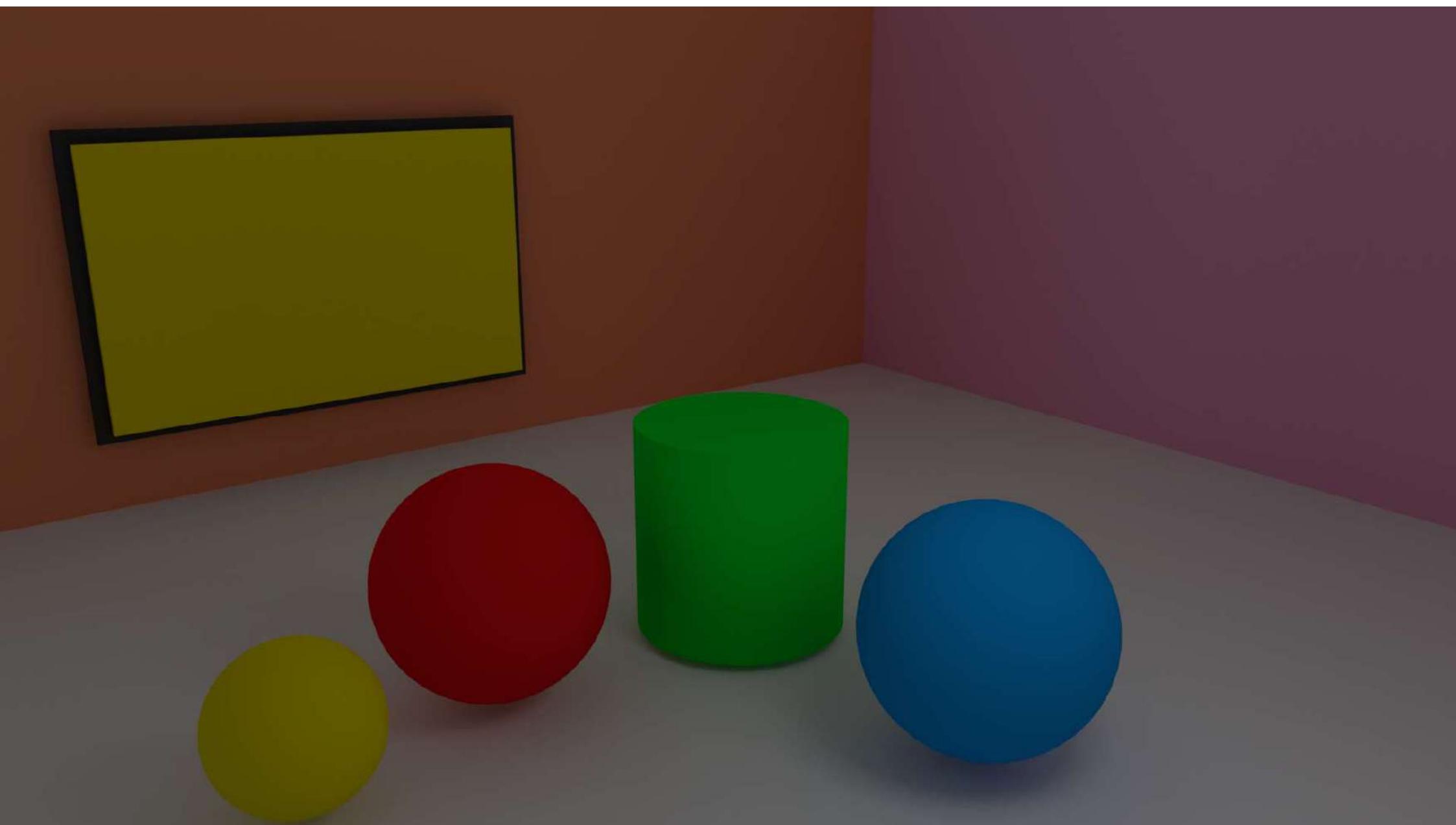


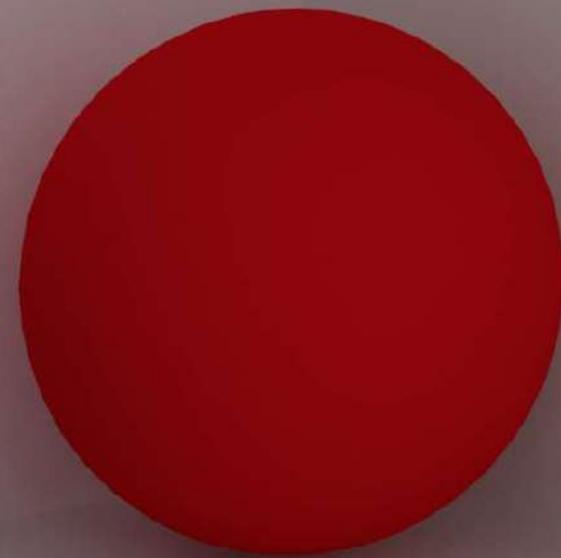
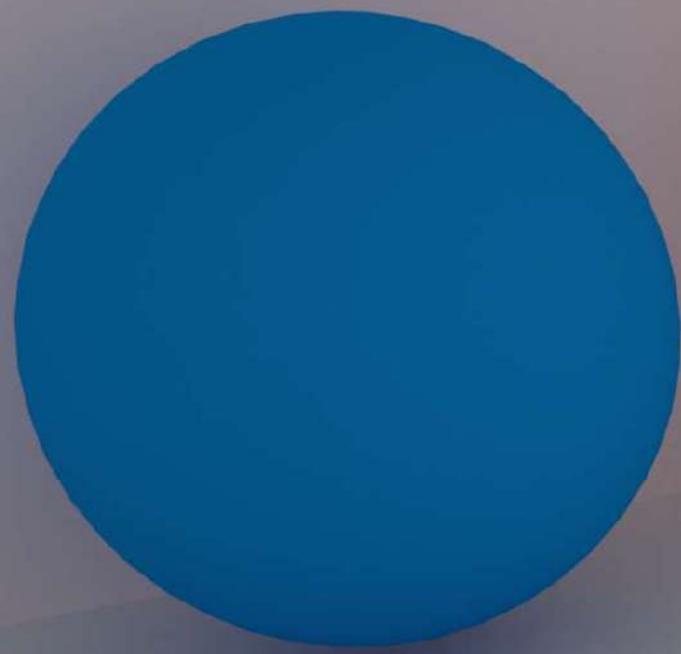
Pabelló de Mies van der Rohe (model: eMirage, Claudio Andres, Hamza Cheggour)

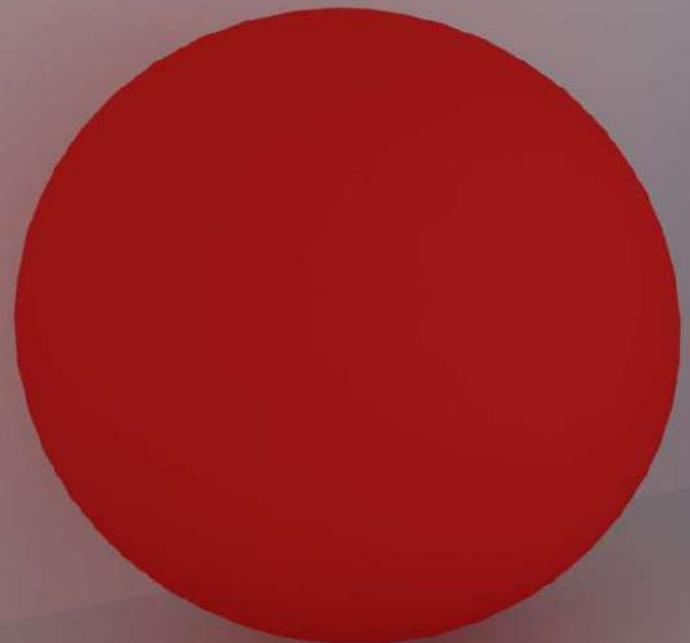


Sponza Atrium by Crytek









RADIOMETRIA



Mesurar el color de la llum

```
vec4 lightDiffuse = vec4(0.8, 0.8, 0.8, 1.0);  
vec4 lightSpecular = vec4(1.0, 1.0, 1.0, 1.0);
```

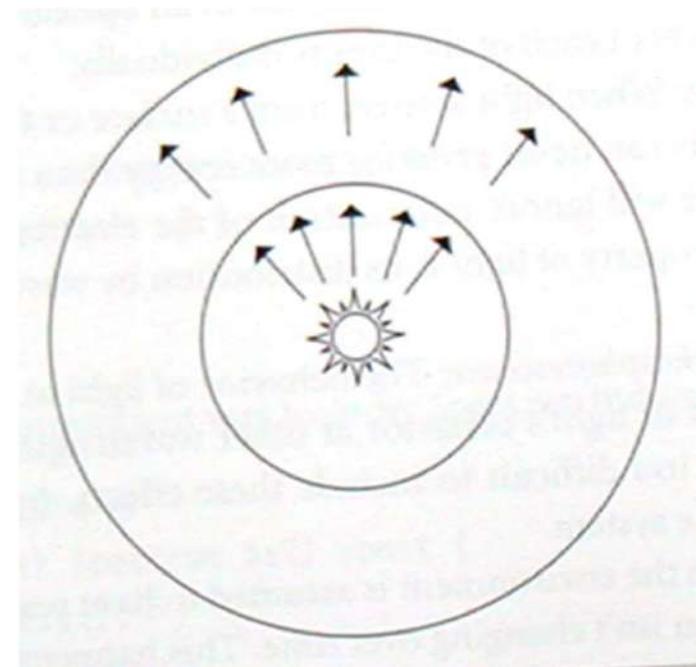


Resum fotometria

Sím.	Radiomet.	Fotometria	Definició	Ús
Φ	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 puntuall
L	Radiància W/(sr·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 Φ

- Definició:
- Unitats:
- Exemple:
- Ús:



Irradiància E

• Definició:

• Unitats:

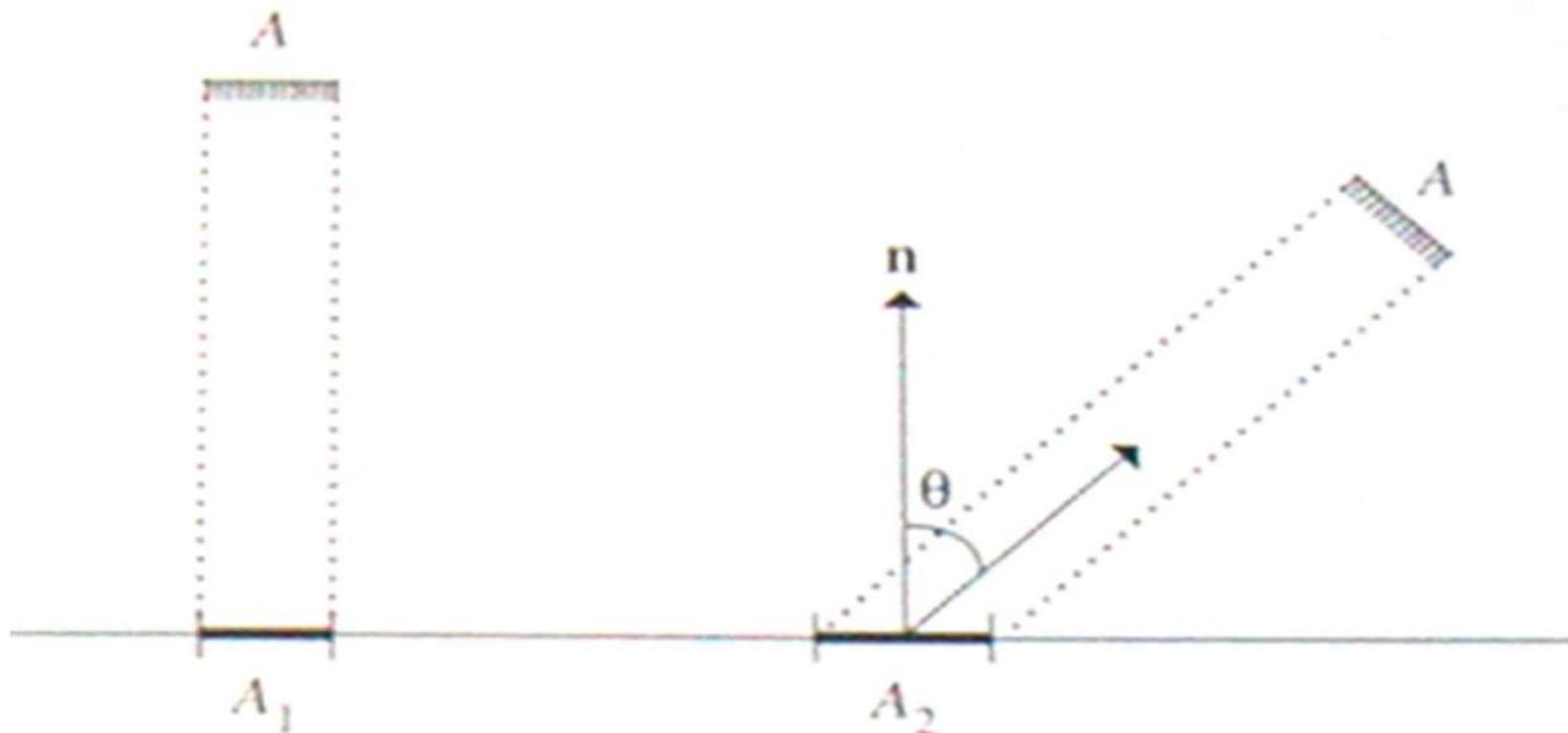
• Exemple:

• Ús:

Irradiància E



Irradiància E i Lley de Lambert

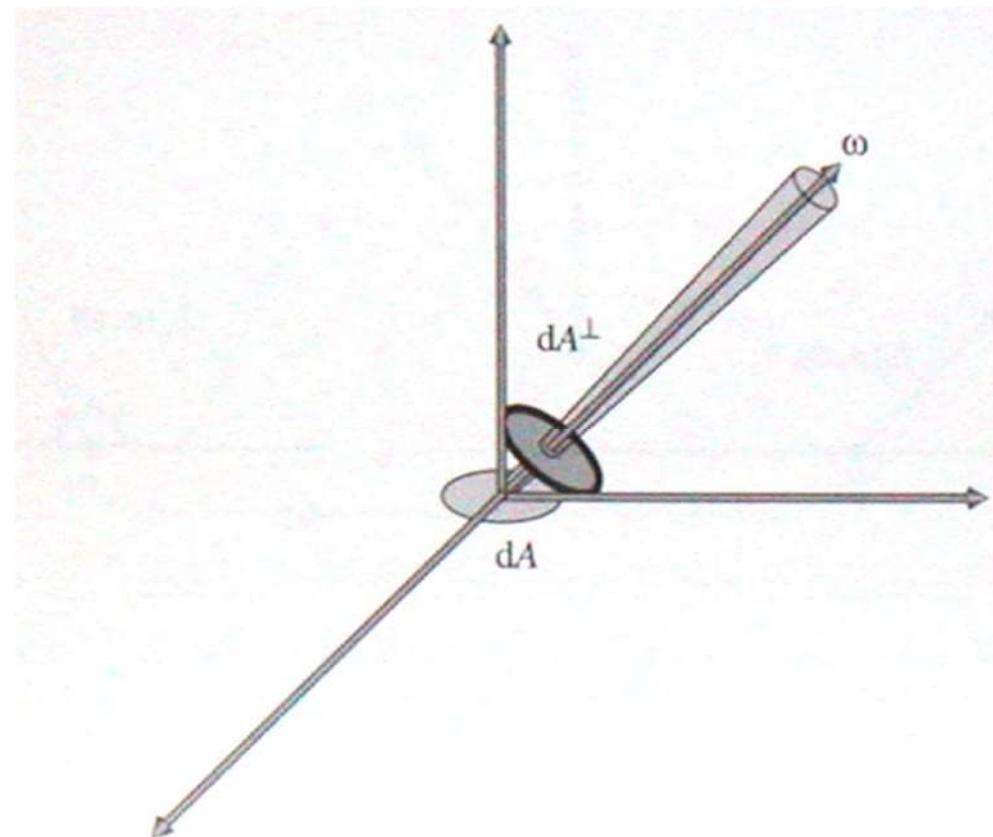


Intensitat I

- Definició:
- Unitats:
- Exemple:
- Ús:

Radiància $L(p,\omega)$

- Definició:
- Unitats:
- Exemple:
- Ús:



Radiància $L(p,\omega)$

Propietats:



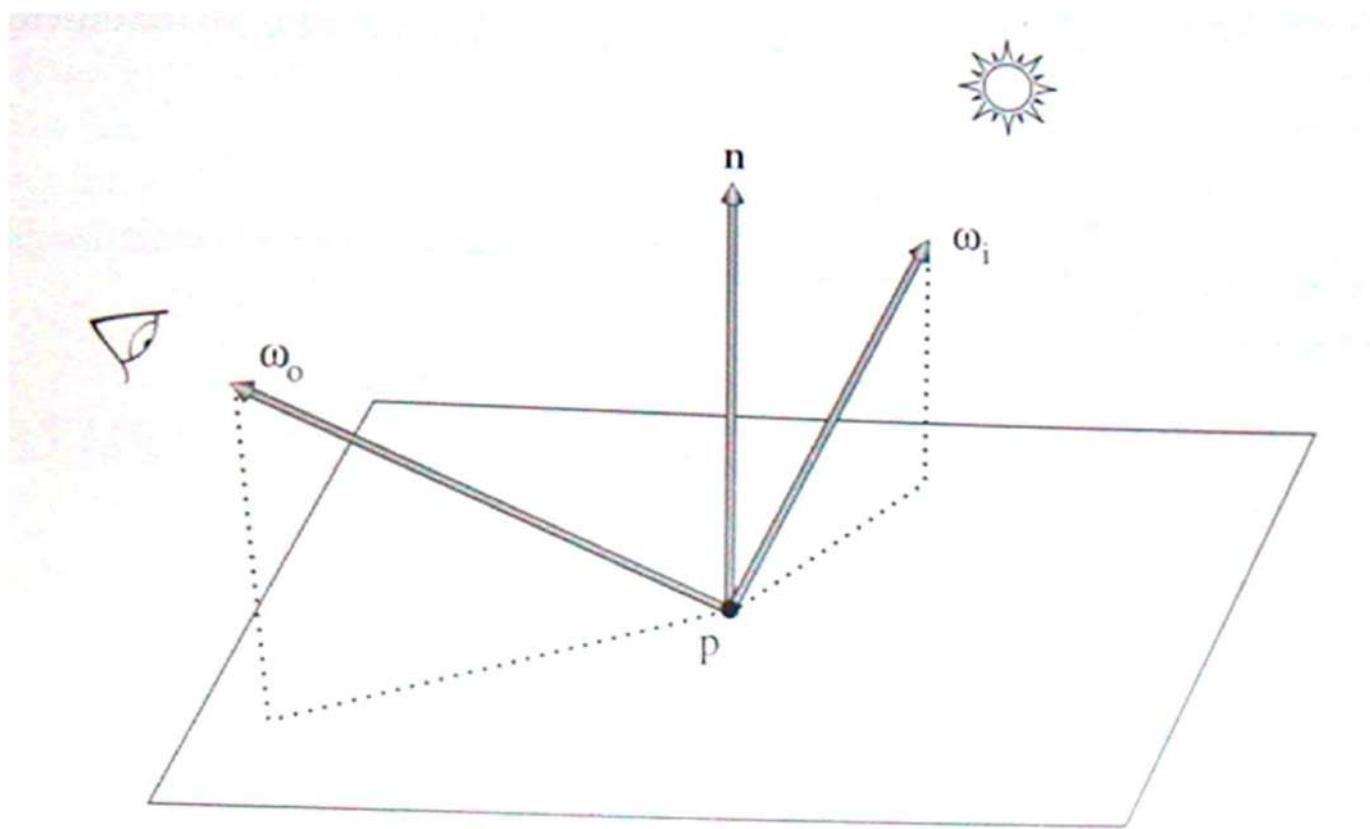
BRDF – BTDF - BSDF

Reflectivitat / transmissió d'una superfície

```
uniform vec4 lightAmbient;           // similar a gl_LightSource[0].ambient
uniform vec4 lightDiffuse;           // similar a gl_LightSource[0].diffuse
uniform vec4 lightSpecular;          // similar a gl_LightSource[0].specular
uniform vec4 lightPosition;          // similar a gl_LightSource[0].position
                                    // (sempre estarà en eye space)

uniform vec4 matAmbient;            // similar a gl_FrontMaterial.ambient
uniform vec4 matDiffuse;             // similar a gl_FrontMaterial.diffuse
uniform vec4 matSpecular;            // similar a gl_FrontMaterial.specular
uniform float matShininess;          // similar a gl_FrontMaterial.shininess
```

$$\text{BRDF } f(p, \omega_o, \omega_i)$$



$$\text{BRDF } f(p, \omega_o, \omega_i)$$

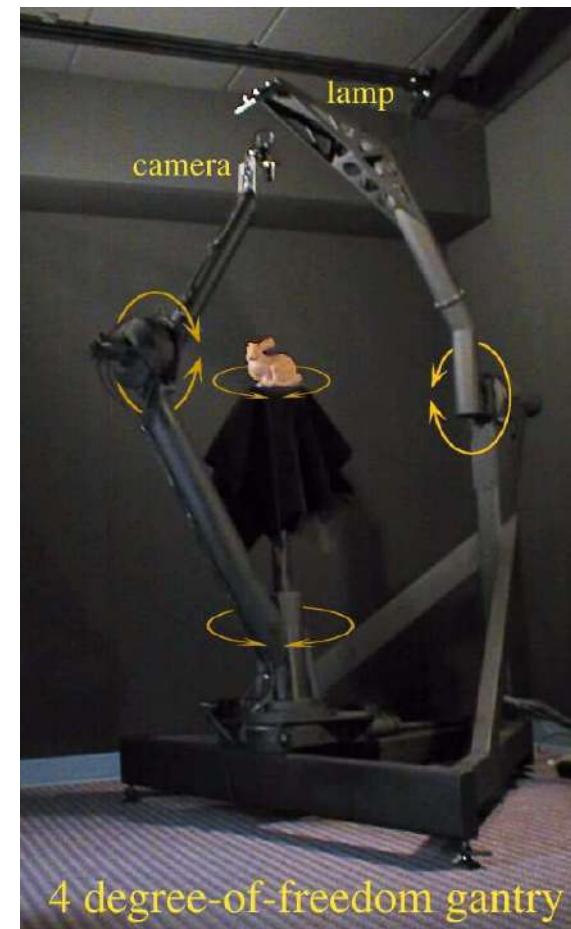
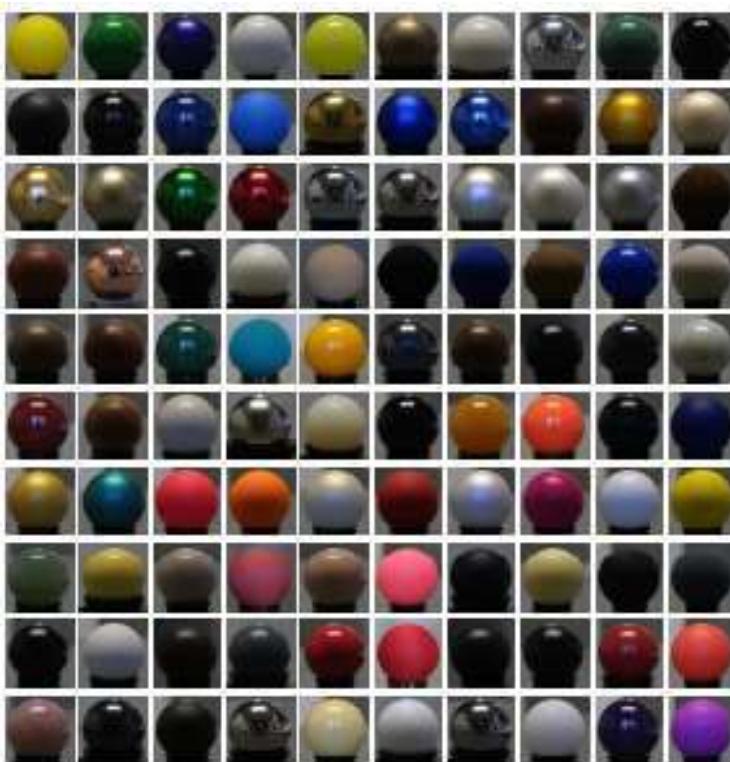
Propietats BRDFs basats en física:

BRDF



MERL BRDF database

- www.merl.com/brdf/



BRDF de pinturas



Phong vs BRDF mesurat



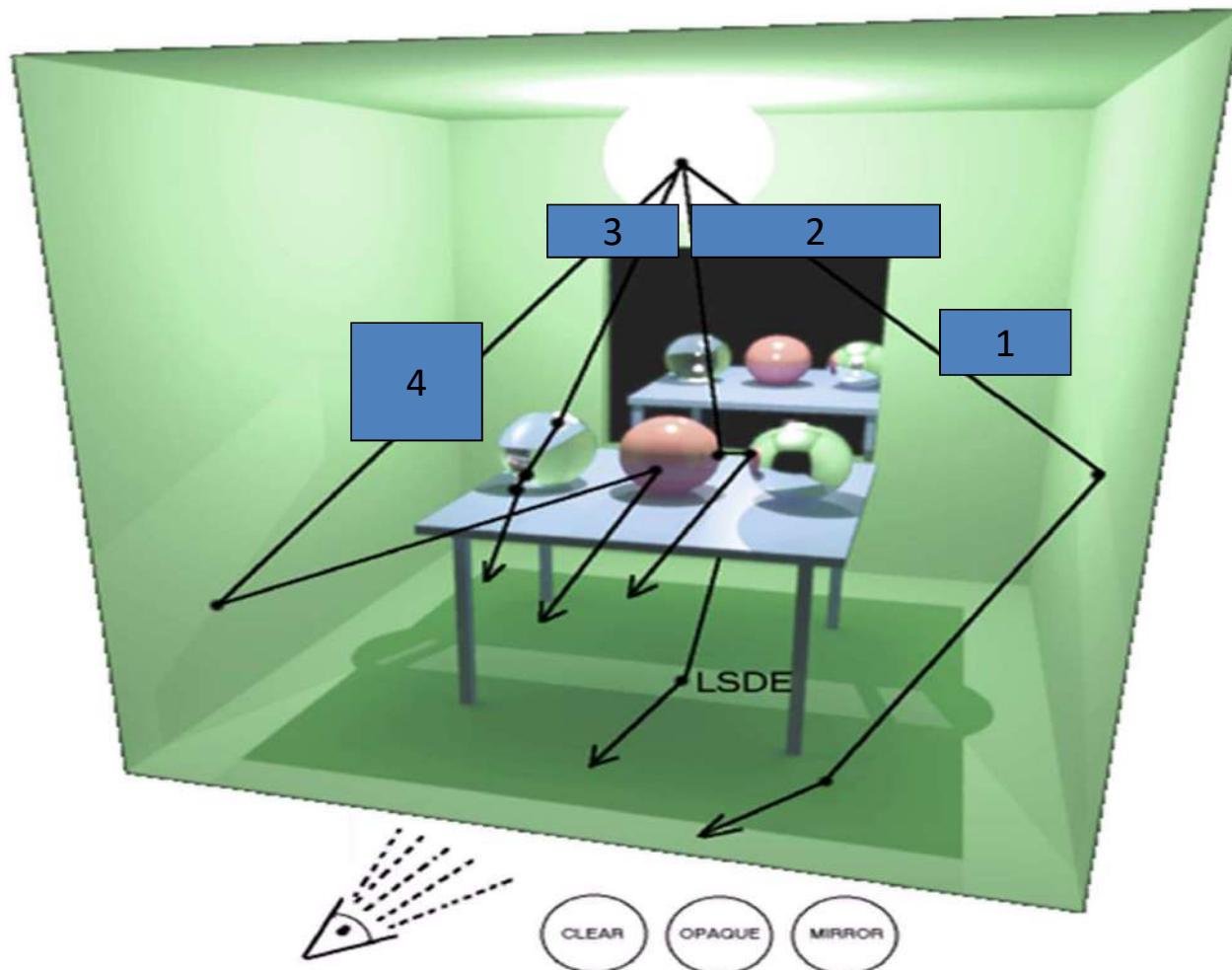


EQÜACIÓ DEL RENDERING

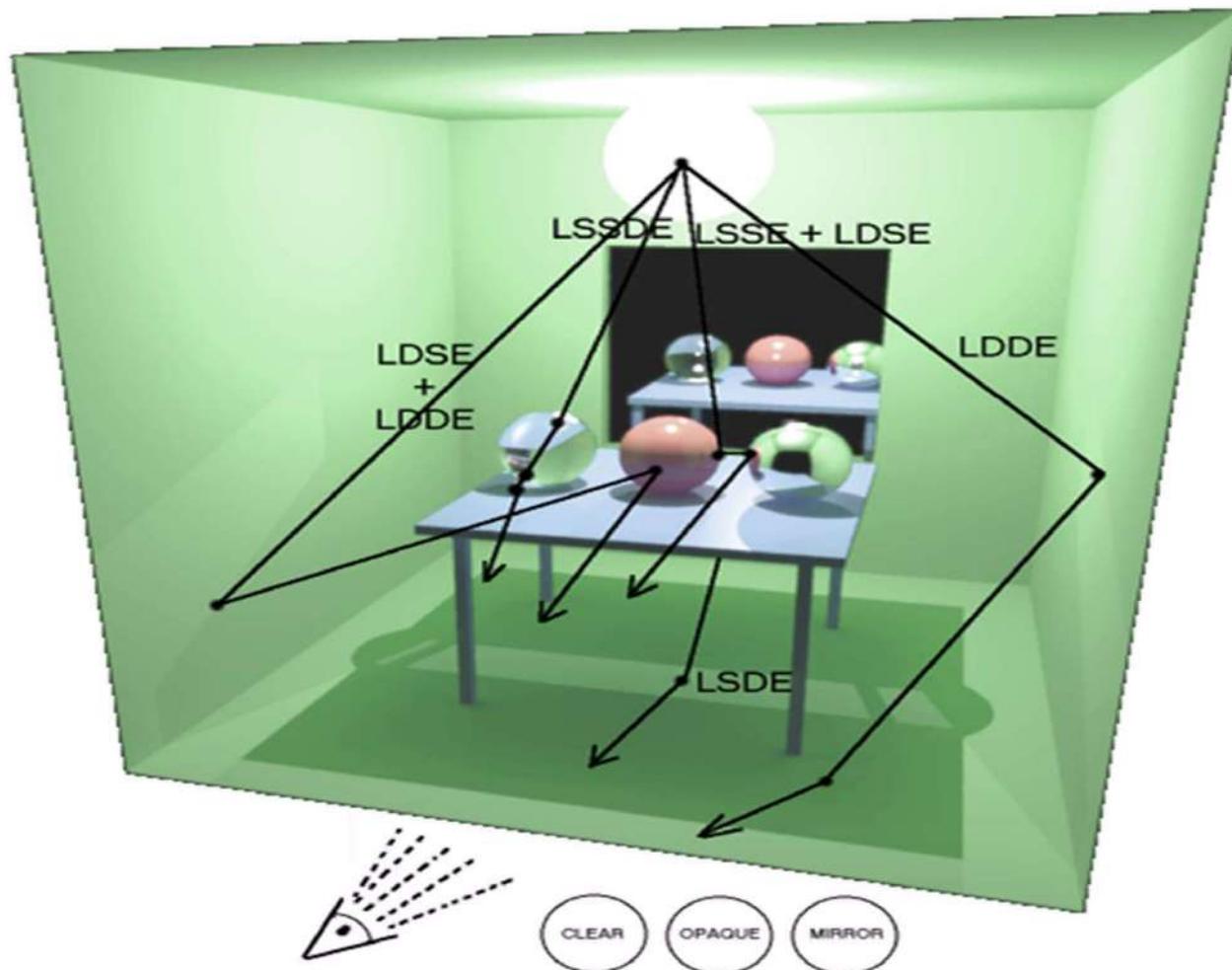
Eqüació general del rendering (Kajiya 1986)

LIGHT PATHS

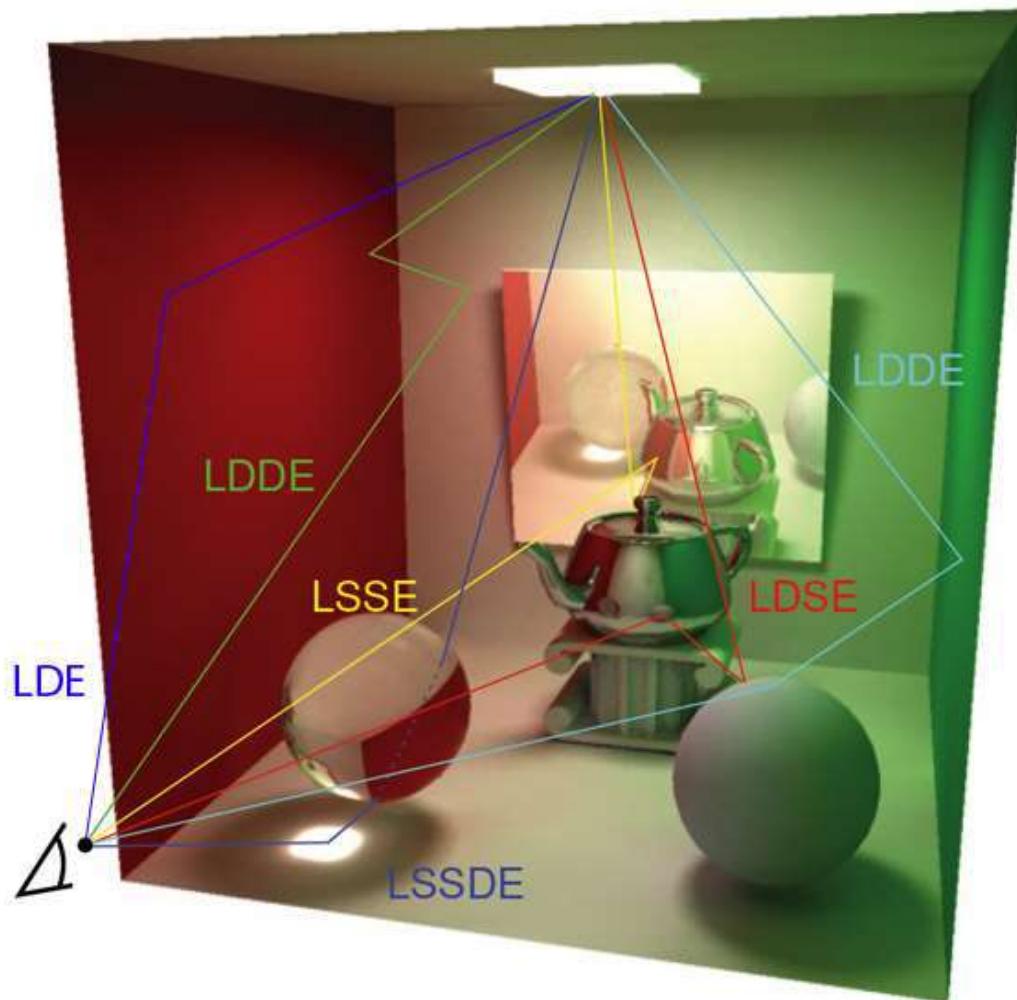
Light paths



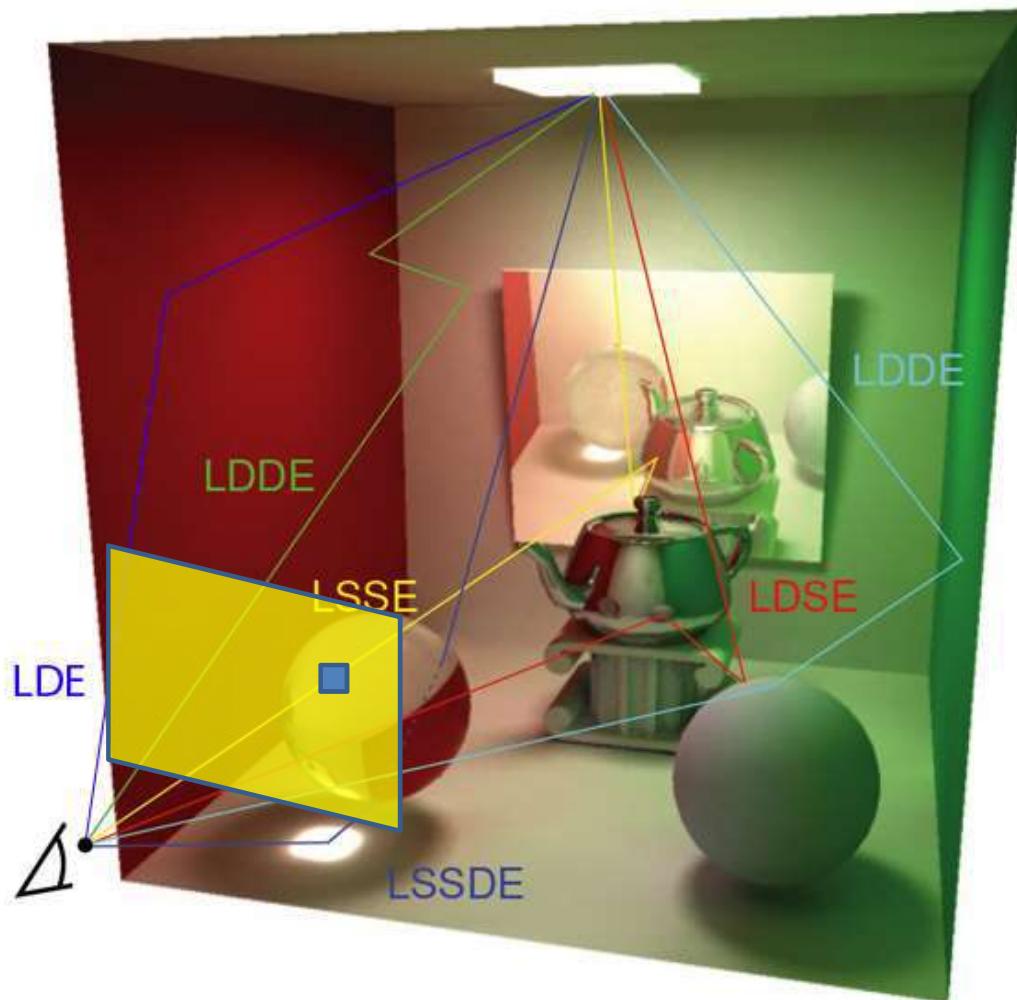
Light paths



Light paths



Light paths

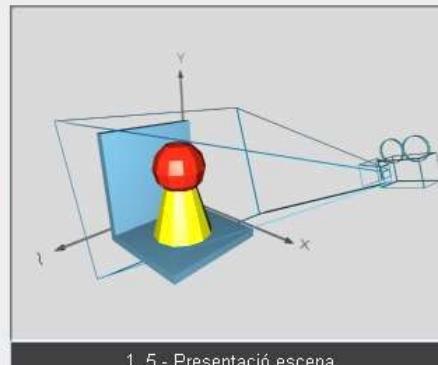


RAY CASTING - RAY TRACING

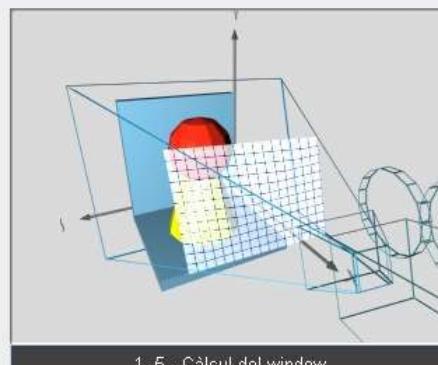
Ray-casting

```
accio ray_casting
    per y en [vyM...vym] fer
        per x en [vxm...vxM] fer
            r=calcul_raig(x,y,Obs)
            per cada cara fer
                trobar_interseccions_raig_cara
            fper
            ordenar_interseccions
        fper
    faccio
```

[Anterior](#) [Següent](#)



1.5 - Presentació escena.



1.5 - Càlcul del window.

Ray-casting

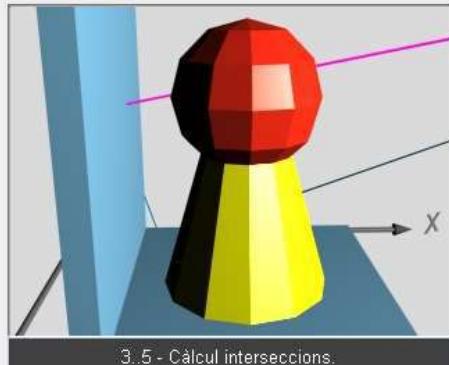
```
accio ray_casting
    per y en [vyM...vym] fer
        per x en [vxm...vxM] fer
            r=calcul_raig(x,y,Obs)
            per cada cara fer
                trobar_interseccions_raig_cara
            fper
            ordenar_interseccions
        fper
        faccio
```

[Anterior](#) [Següent](#)

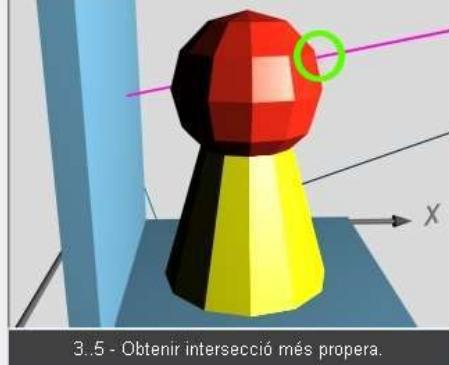
Ray-casting

```
accio ray_casting
    per y en [vyM...vym] fer
        per x en [vxm...vxM] fer
            r=calcul_raig(x,y,Obs)
            per cada cara fer
                trobar_interseccions_raig_cara
            fper
            ordenar_interseccions
            fper
            faccio
```

Anterior Següent



3.5 - Càcul interseccions.



3.5 - Obtenir intersecció més propera.

Ray-casting

The screenshot displays a software interface for implementing a ray-casting algorithm. On the left, a code editor contains the following pseudocode:

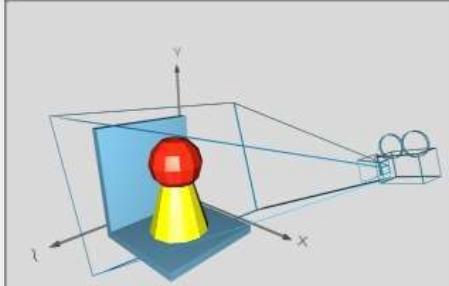
```
accio ray_casting
    per y en [vyM...vym] fer
        per x en [vxm...vxM] fer
            r=calcul_raig(x,y,Obs)
            per cada cara fer
                trobar_interseccions_raig_cara
            fper
            ordenar_interseccions
        fper
    fper
    faccio
```

Below the code editor are two buttons: "Anterior" and "Següent". To the right of the code editor is a 16x16 grid showing a single red pixel at position (8,8). Below this grid is a caption: "4..5 - Pintar pixel del color de la intersecció." To the right of the first grid is a larger 16x16 grid showing a low-resolution 3D rendered scene of a character or object composed of colored pixels (red, blue, yellow, green, black). Below this larger grid is a caption: "4..5 - Resultat en baixa resolució."

Ray-casting

```
accio ray_casting
    per y en [vyM...vym] fer
        per x en [vxm...vxM] fer
            r=calcul_raig(x,y,Obs)
            per cada cara fer
                trobar_interseccions_raig_cara
            fper
            ordenar_interseccions
        fper
    fper
faccio
```

[Anterior](#) [Següent](#)



5.5 - Presentació escena.



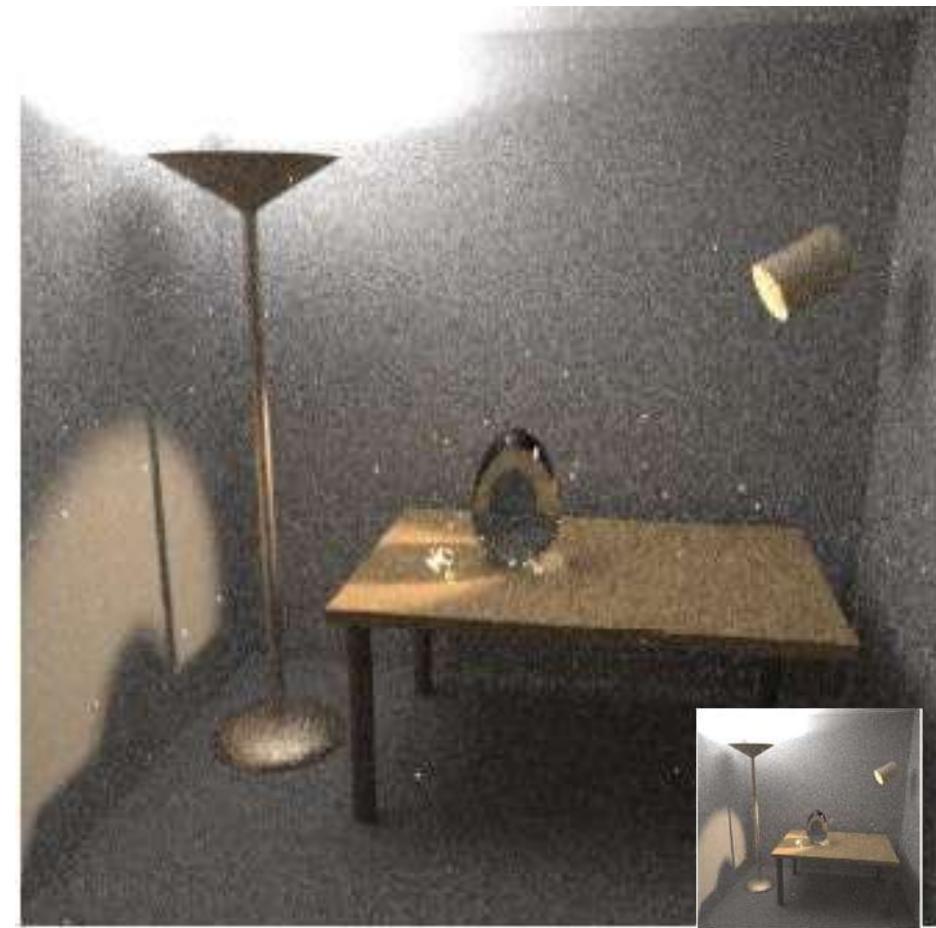
5.5 - Visió final en resolució estàndard.

RAY TRACING

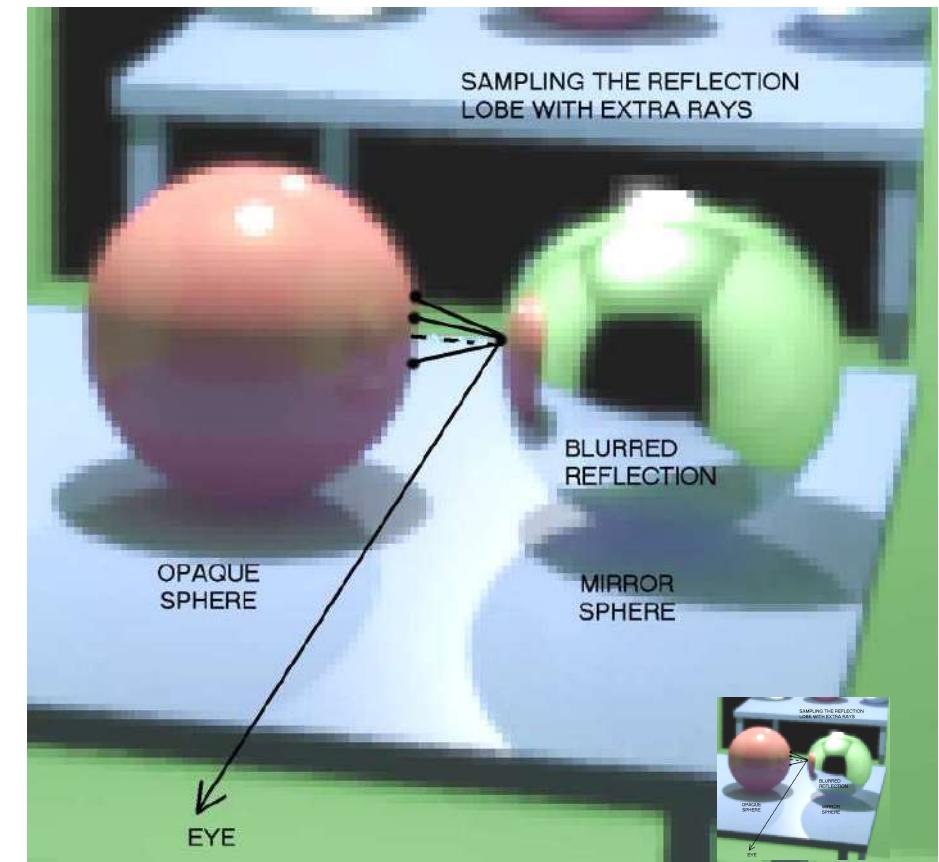
Ray-tracing clàssic



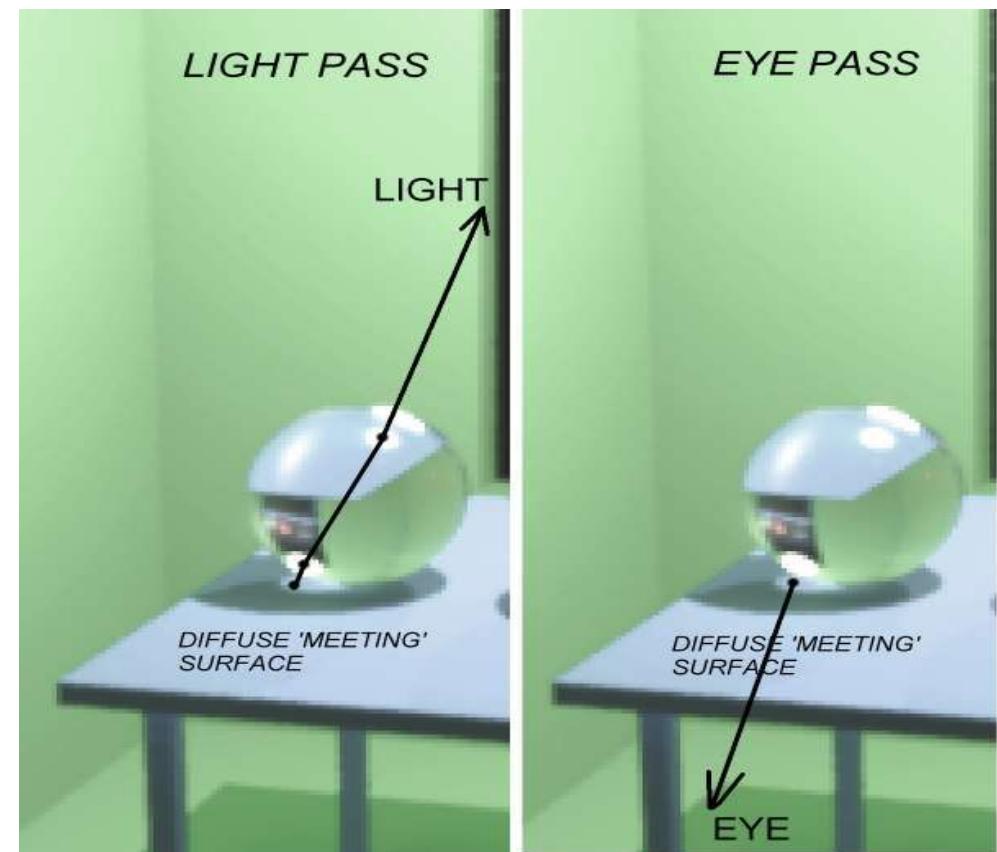
Path tracing



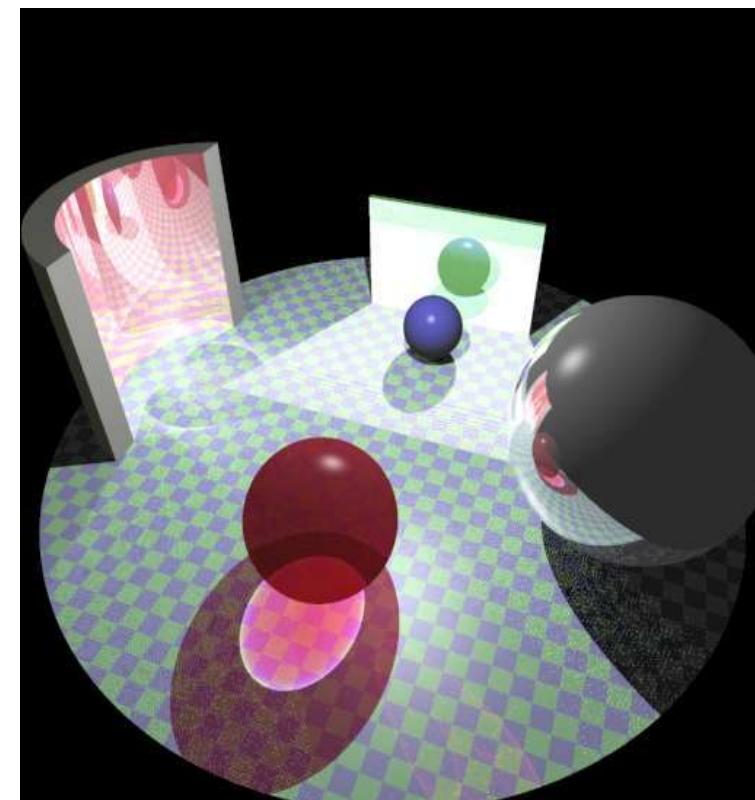
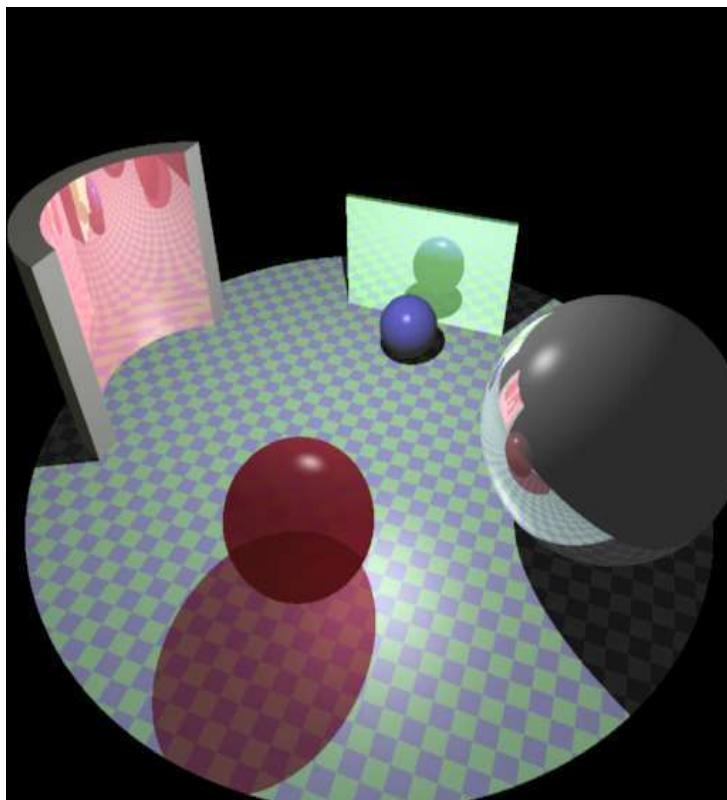
Distributed ray-tracing



Two-pass ray-tracing



Classic vs Two-pass ray-tracing



RAY TRACING CLÀSSIC

Color d'un punt P

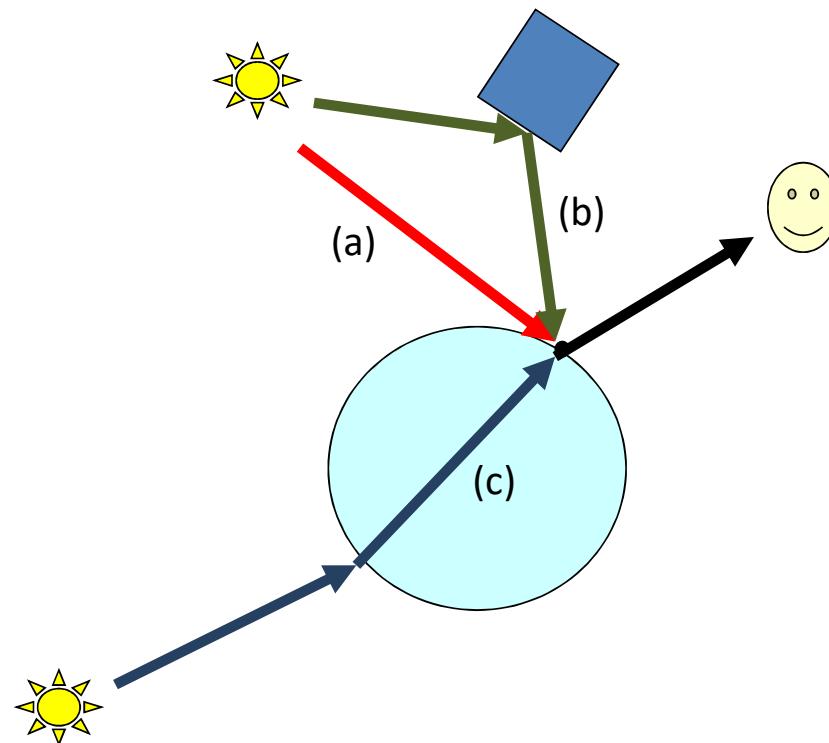
- El color amb que un determinat observador veu un cert punt P es calcula:

$$I(P) = I_D(P) + I_R(P) + I_T(P)$$

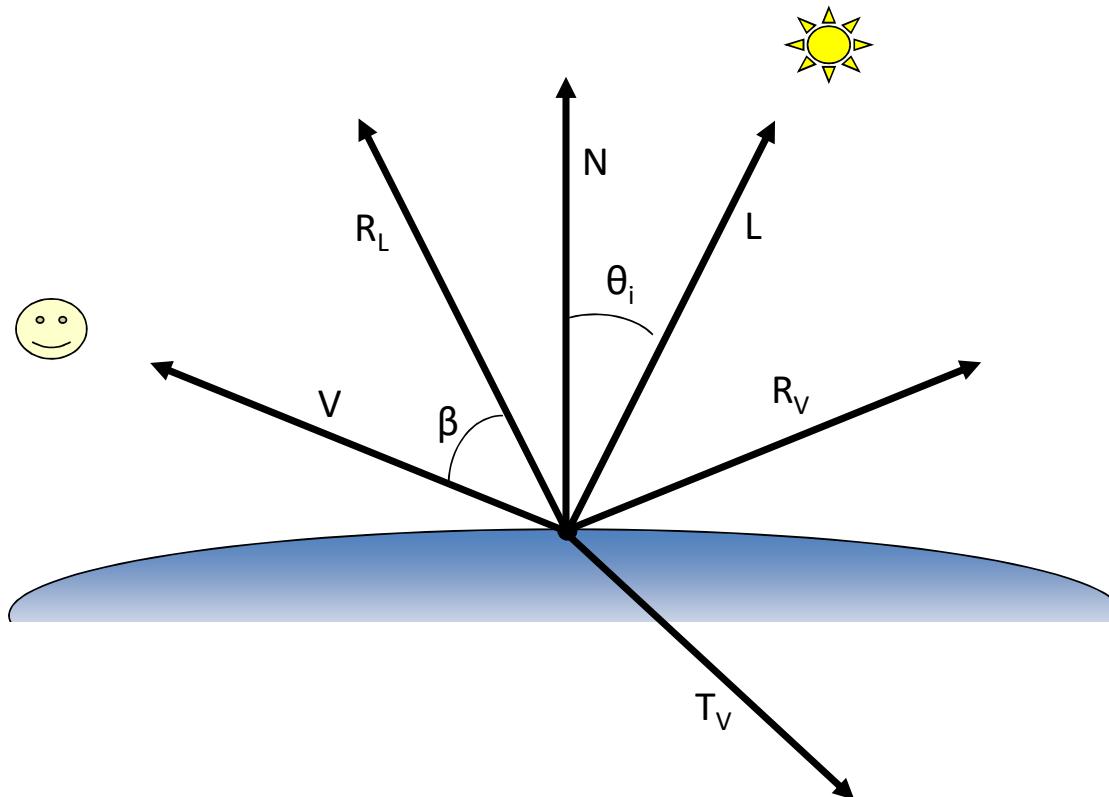
- $I_D(P)$ és color degut a la **llum directa** dels focus.
- $I_R(P)$ és color degut a la **llum indirecta que es reflecteix** a P en direcció cap a l'observador.
- $I_T(P)$ és color degut a la **llum indirecta que es transmet** des de P en direcció cap a l'observador.

Color d'un punt P

$$I(P) = I_D(P) + I_R(P) + I_T(P)$$



Color d'un punt P: notació

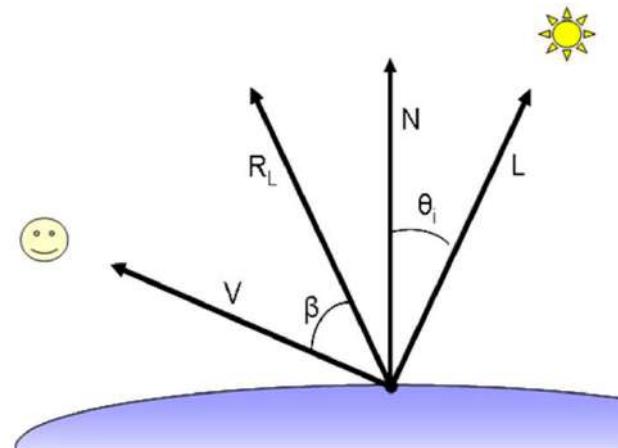


Contribució llum directa I_D

$$I(P) = I_D(P) + I_R(P) + I_T(P)$$

$$I_D(P) = K_a I_a + K_d \sum I_L \cos(\theta_i) + K_s \sum I_L \cos^n(\beta)$$

- $\cos(\theta_i) = N \cdot L$
- $\cos(\beta) = R_L V$
- El sumatori només considera les fonts de llum no ocluides (ombres)

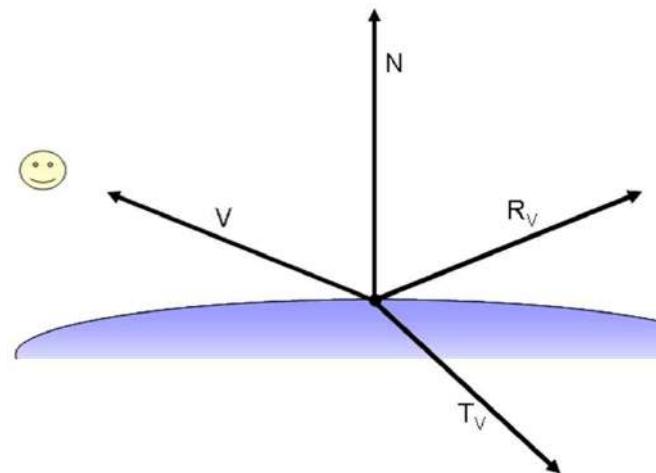


Contribución llum indirecta I_R , I_T

$$I(P) = I_D(P) + I_R(P) + I_T(P)$$

$$I_R(P) = K_R L_R$$

$$I_T(P) = K_T L_T$$



- K_R , K_T coeficients empírics de reflexió/transmissió especular
- L_R = llum que incideix en P en la direcció R_V
- L_T = llum que incideix en P en la direcció T_V

Es calculen recursivament, traçant un nou raig reflectit i un altre transmès

Algorisme

```
acció rayTracing
    per i en [0..w-1] fer
        per j en [0..h-1] fer
            raig:=raigPrimari(i, j, camera);
            color:=traçarRaig(raig, escena, μ);
            setPixel(i, j, color);
        fper
    fper
faccio
```

Algorisme

```
funció traçar_raig(raig, escena, μ)
    si profunditat_correcta() llavors
        info:=calcula_interseccio(raig, escena)
        si info.hi_ha_interseccio() llavors
            color:=calcular_ID(info,escena); // ID
            si es_reflector(info.obj) llavors
                raigR:=calcula_raig_reflectit(info, raig)
                color+= KR*traçar_raig(raigR, escena, μ) //IR
            fsi
            si es_transparent(info.obj) llavors
                raigT:=calcula_raig_transmès(info, raig, μ)
                color+= KT*traçar_raig(raigT, escena, info.μ) //IT
            fsi
            sino color:=colorDeFons
            fsi
            sino color:=Color(0,0,0); // o colorDeFons
            fsi
            retorna color
ffunció
```

Shader (FS)

```
void main() {  
    vec3 obs = gl_ModelViewMatrixInverse[3].xyz;  
    vec3 dir = normalize(pos - obs);  
    Ray ray = Ray(obs,dir);  
    gl_FragColor = trace(ray, 1.0);  
}
```

Shader (FS)

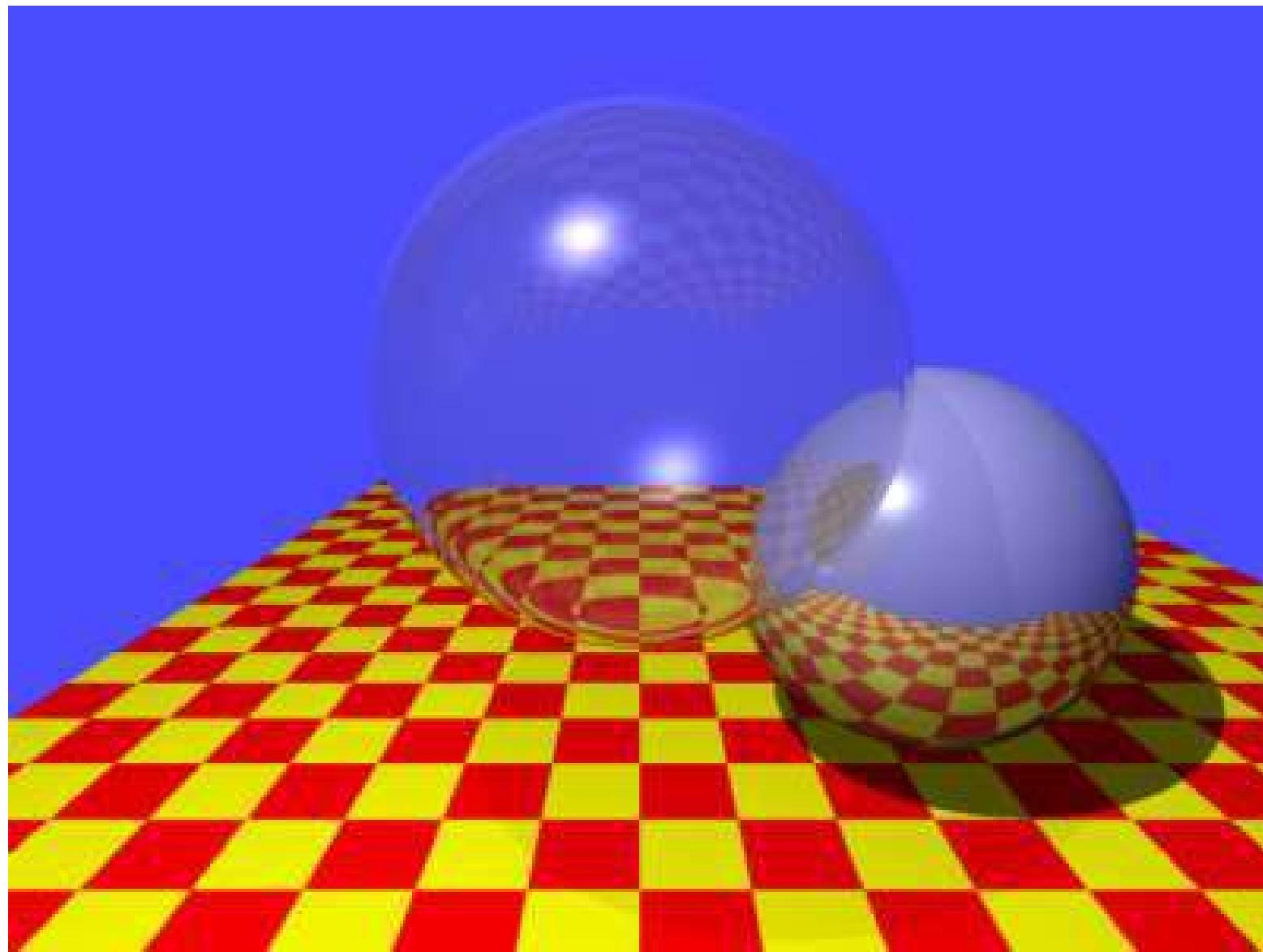
```
vec4 trace(Ray ray, float mu) {  
    if ( intersectScene(ray, Phit, Nhit,Kd, Kr, Kt) ) {  
        Ray shadowRay = Ray(Phit, normalize(lightPos-Phit));  
        bool inShadow = intersectScene(shadowRay, aux, aux,aux4, aux4, aux4);  
        if (inShadow) shadow = 0.2; else shadow = 1.0;  
        color+= shadow*light(Nhit, -ray.dir, lightPos-Phit, Kd, vec4(1.0));  
        vec3 R = reflect(ray.dir, Nhit);  
        color += Kr*trace1(Ray(Phit, R), mu);  
        if (mu==1.0) muHit=1.5;  
        else { muHit = 1.0; Nhit*=-1.0;}  
        vec3 T = refract(ray.dir, Nhit, mu/muHit);  
        if (length(T)>0.0) color+=Kt*trace1(Ray(Phit, T), muHit);  
    }  
    else color+=samplePanorama(ray.dir);  
    return color;  
}
```

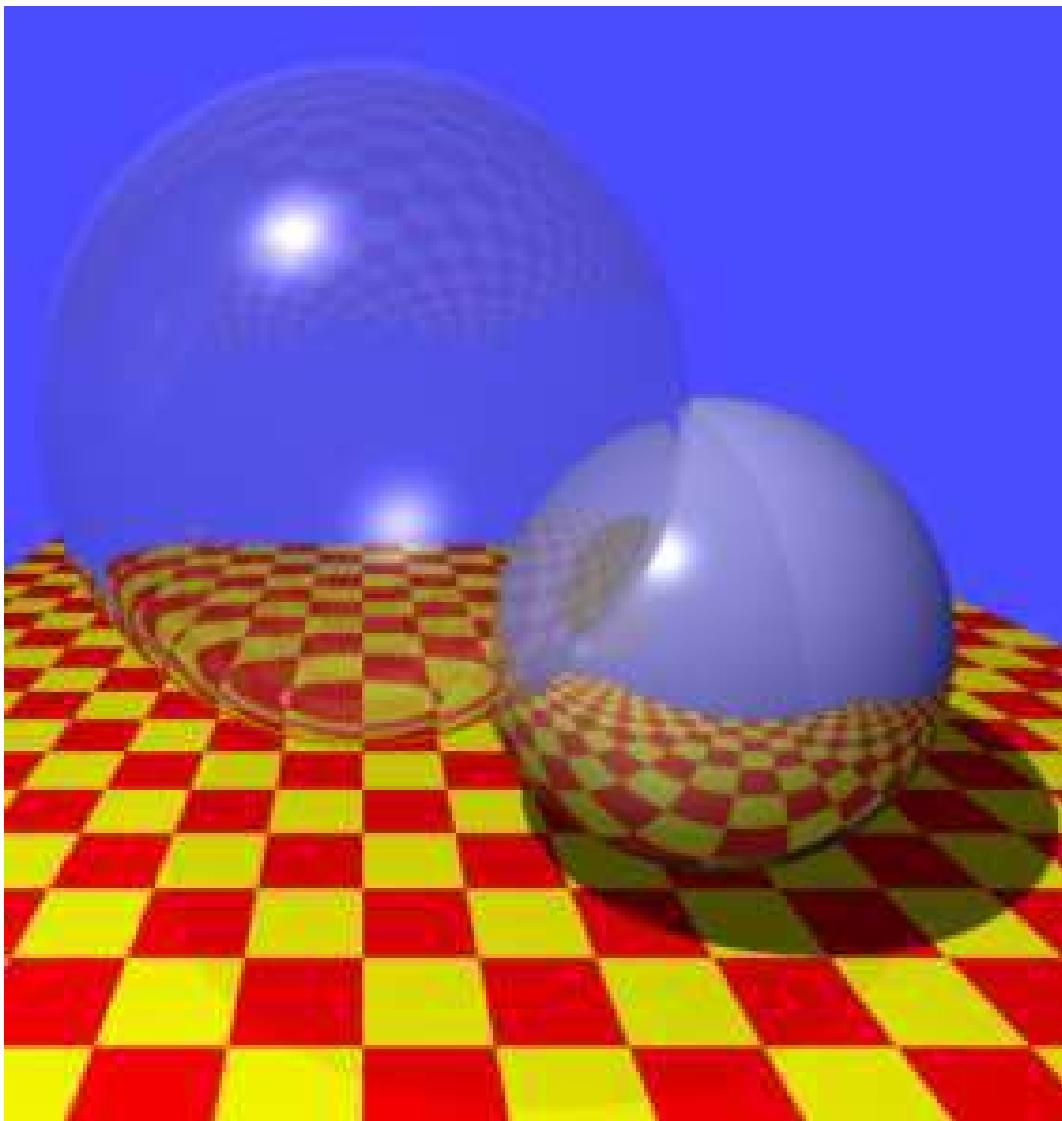
Shader (FS)

```
vec4 samplePanorama(vec3 R)
{
    float psi = asin(R.y);
    float theta = atan(R.z, R.x);
    float t = 1.0-psi/PI + 0.5;
    float s = theta/(2.0*PI) +0.7;
    return texture(panorama,vec2(s,t));
}
```

Shader (FS)

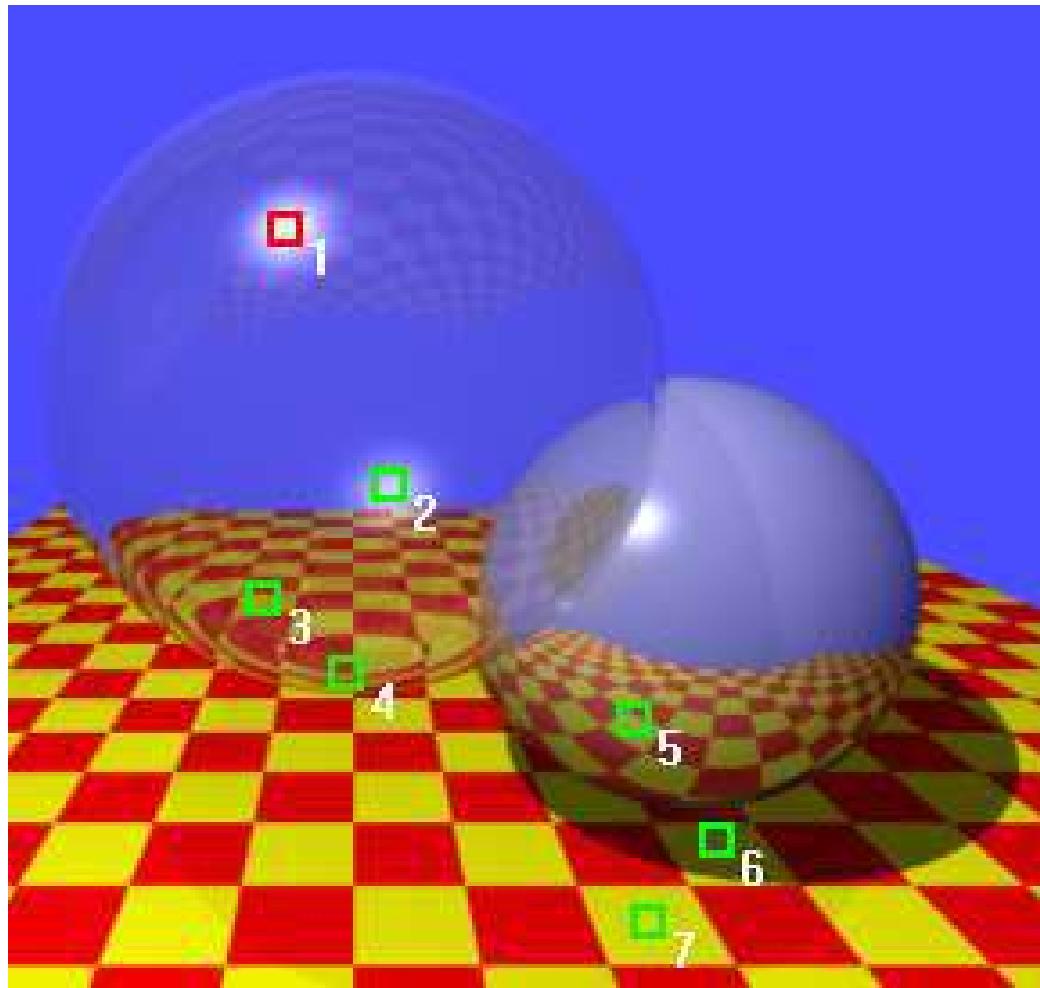
```
vec4 samplePanorama(vec3 R)
{
    float psi = asin(R.y);
    float theta = atan(R.z, R.x);
    float t = 1.0-psi/PI + 0.5;
    float s = theta/(2.0*PI) +0.7;
    return texture(panorama,vec2(s,t));
}
```





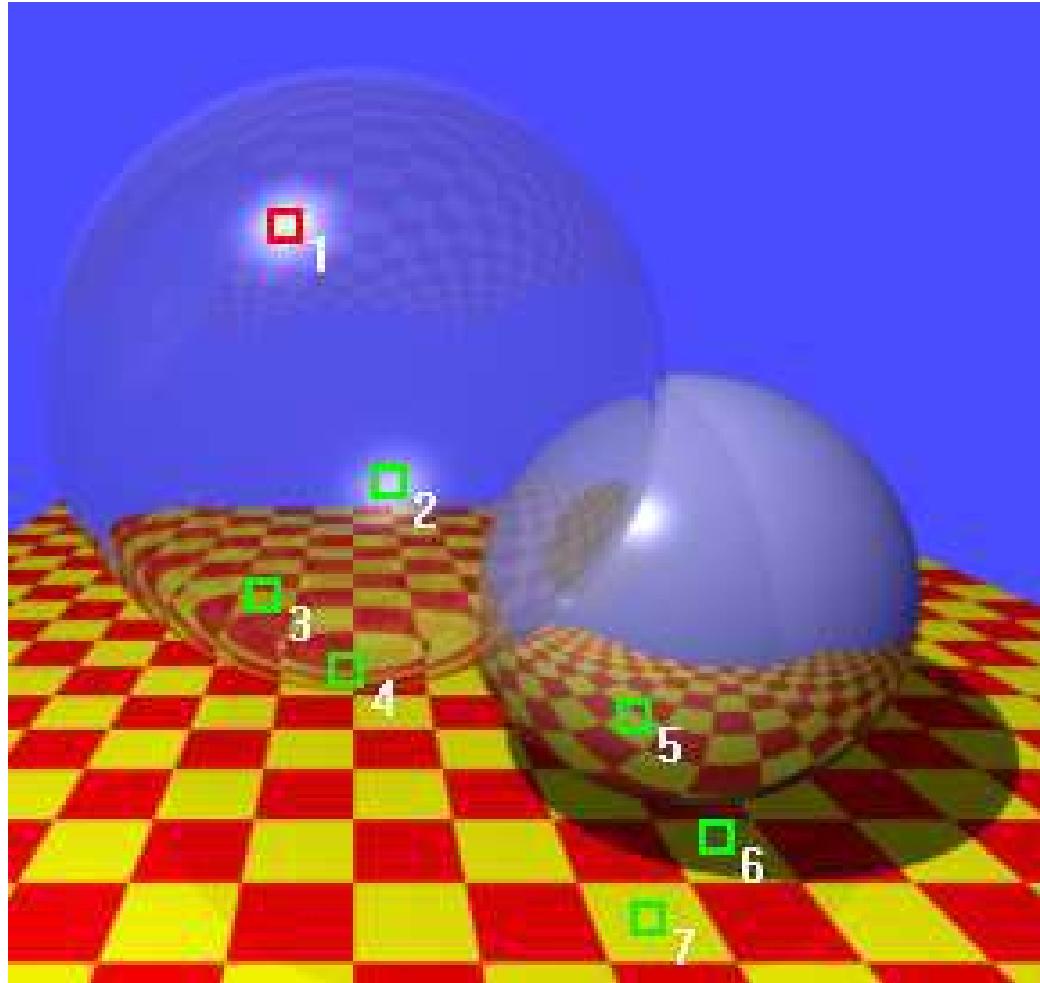
Escena:

- Esfera **buida** transparent.
- Esfera opaca blanca mirall.
- Tauler escacs
- Fons blau
- Llum blanca



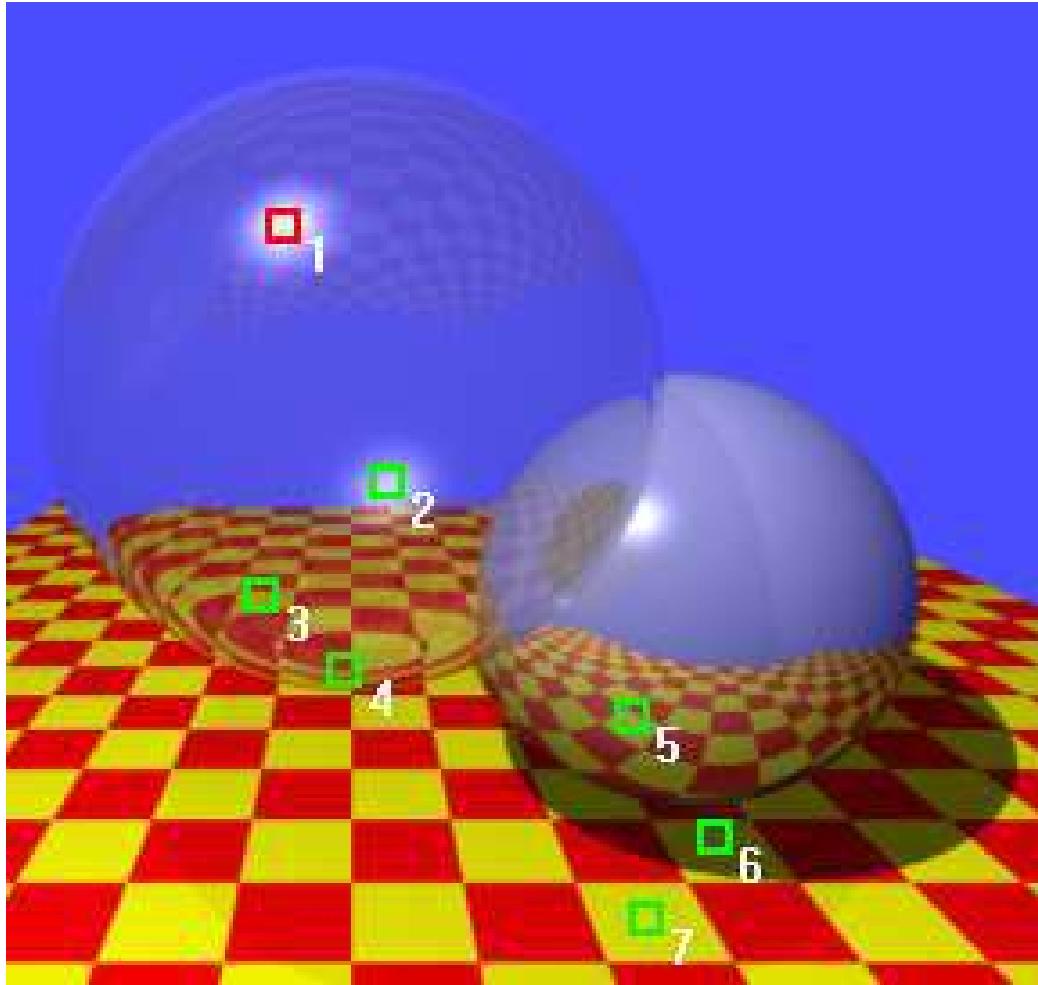
Rraig 1

- El rraig primari gairebé coincideix amb la reflexió del vector L: podem veure una taca espectral (specular highlight).
- La contribució principal és la component espectral de $I_D(P)$.



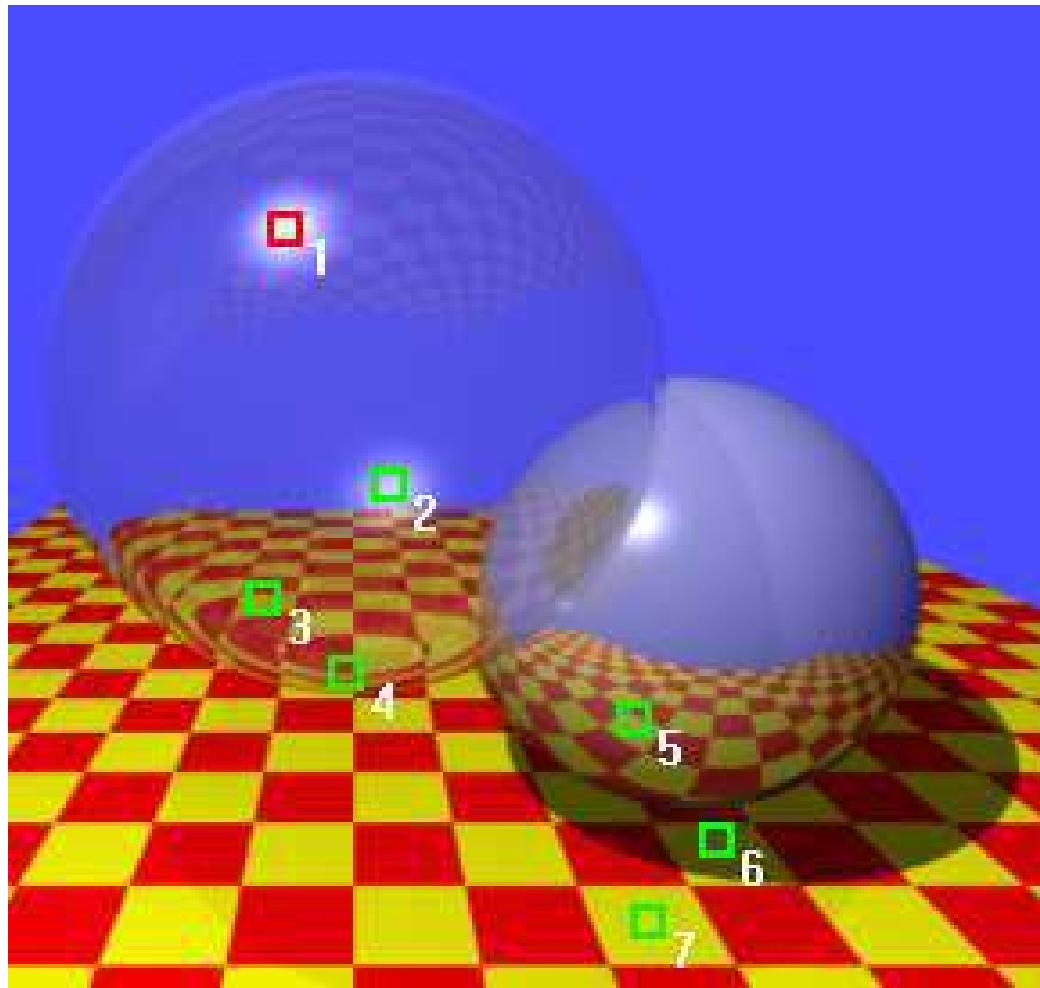
Rraig 2

- És la mateixa situació que el raig 1, però ara la taca specular apareix a la paret interna de l'esfera (que és buida).
- Aquest raig mostra un error acceptat en ray-tracing: el raig de la font de llum travessa l'esfera sense refracció (només comparem L, N ignorant el fet que som a dins de l'esfera). Per tant, la taca specular està en una posició errònia, però no tenim cap intuïció de la posició correcta.



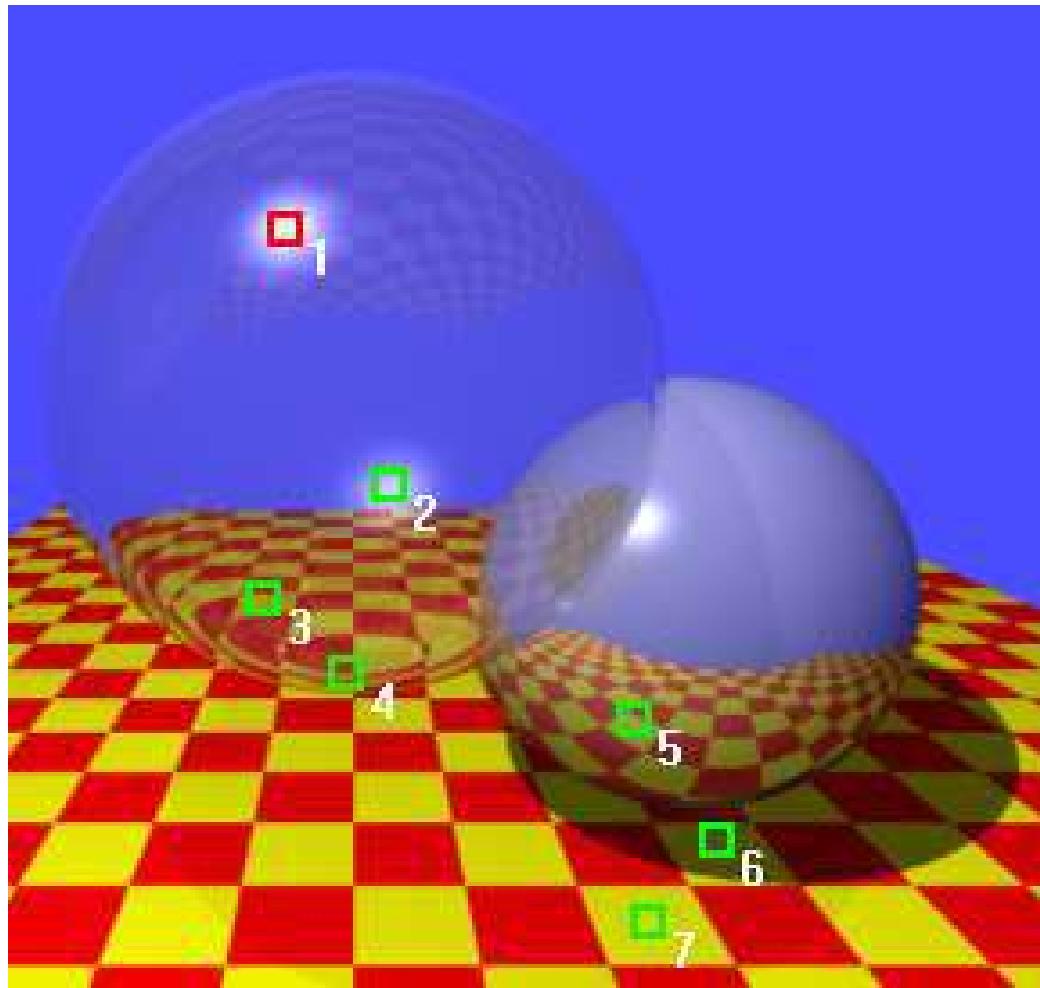
Rraig 3

- En totes les interseccions d'aquest raig amb l'esfera la contribució local és nula.
- La contribució dominant és el tauler vermell-groc.
- Hi ha una lleugera distorsió deguda a la refracció.
- Hi ha una barreja de dos taulers: el transmès i el reflectit a la superficie exterior de l'esfera.



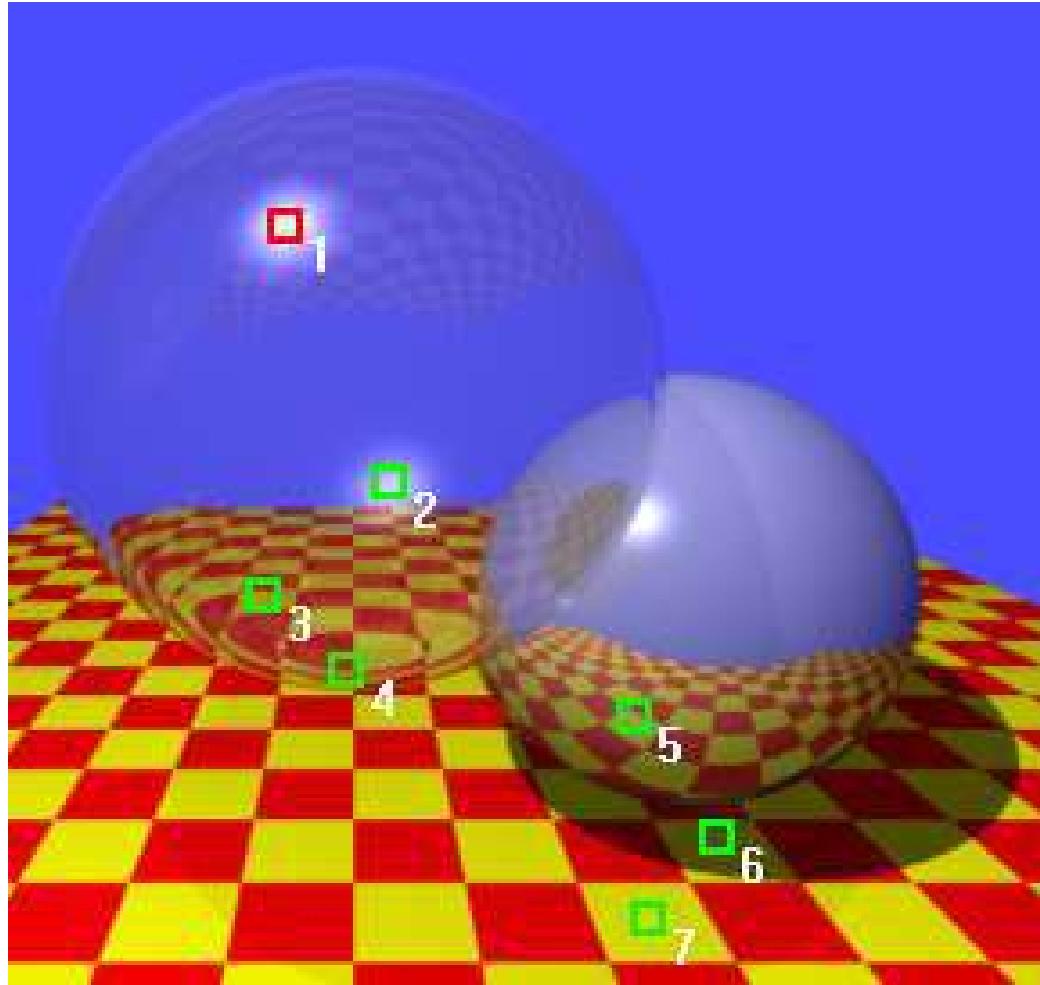
Rraig 4

- Igual que el raig anterior, però ara la distància recorreguda dins el cristall és més significativa, amb la qual cosa l'efecte de la refracció és més notori.



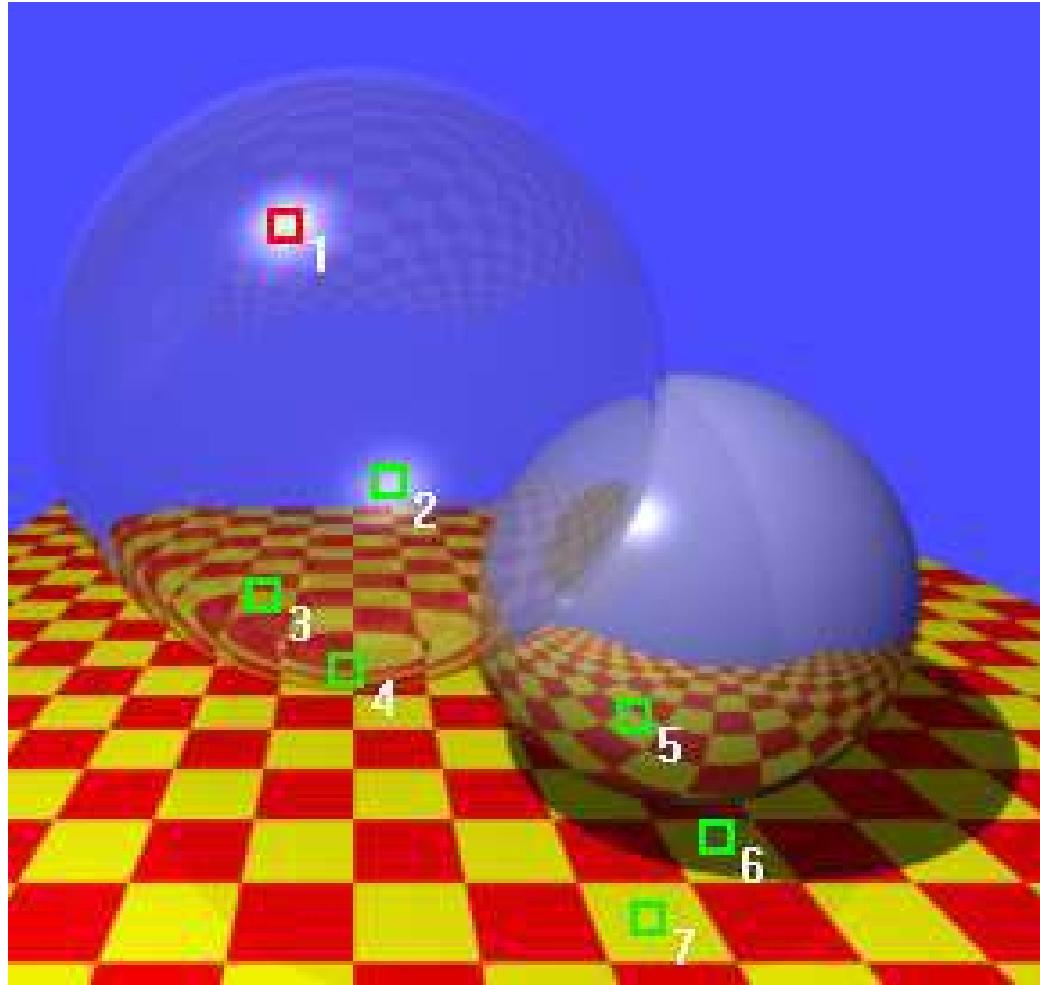
Rraig 5

- El color resultant és la barreja del tauler (raig reflectit) amb el blanc de l'esfera (lleugerament difosa).



Rraig 6

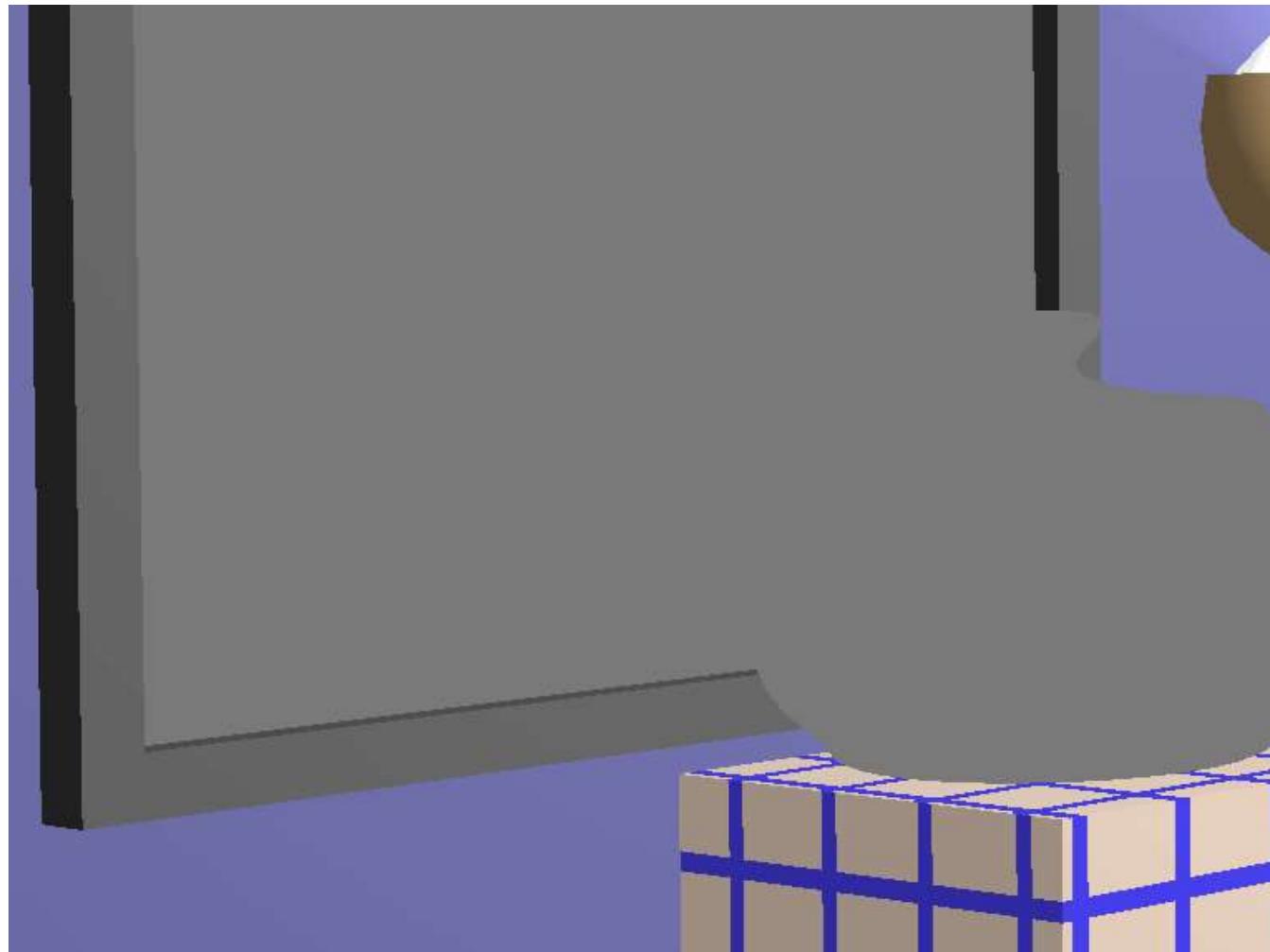
- El raig intersecta el tauler; el punt d'intersecció està a l'ombra (el shadow ray intersecta l'esfera opaca).



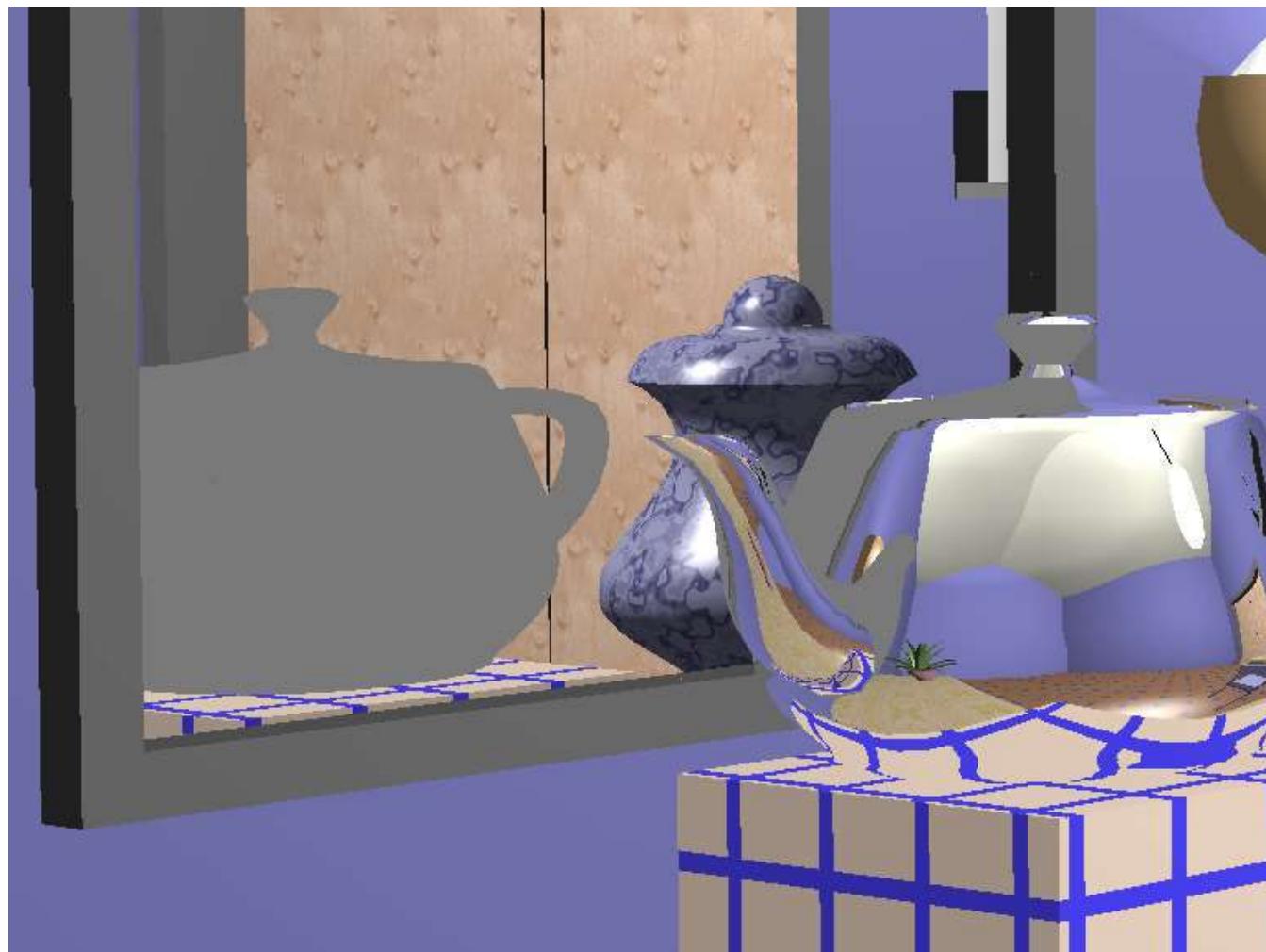
Rraig 7

- Exactament igual que abans però ara el shadow ray intersecta l'esfera transparent i per tant gairebé no reduïm la contribució de la font de llum.
- Novament, no estem tenint en compte que la llum es refractaria a l'esfera i per tant el contorn de l'ombra no està a la posició correcta.

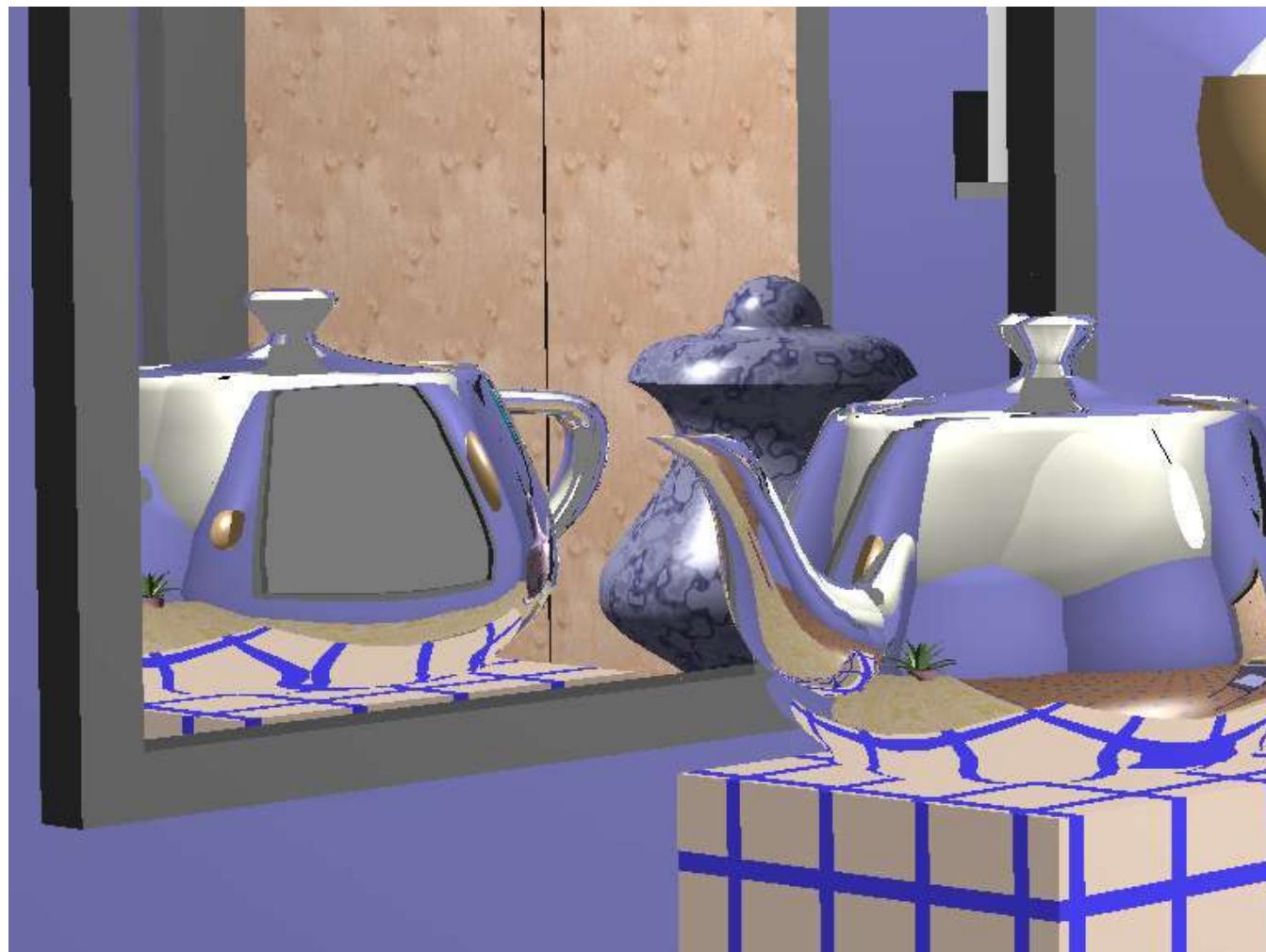
Ray-tracing clàssic. Profunditat 0



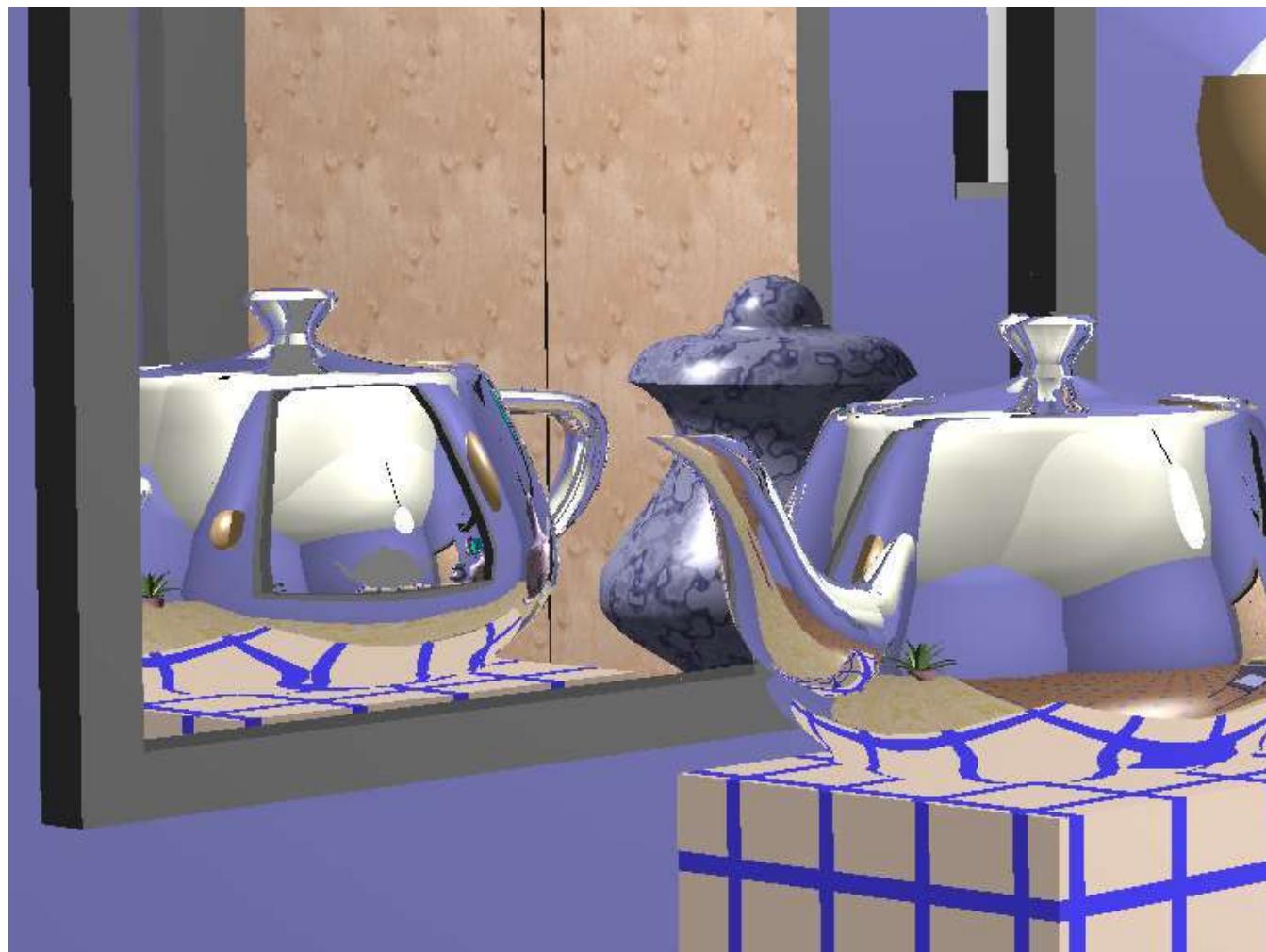
Ray-tracing clàssic. Profunditat 1



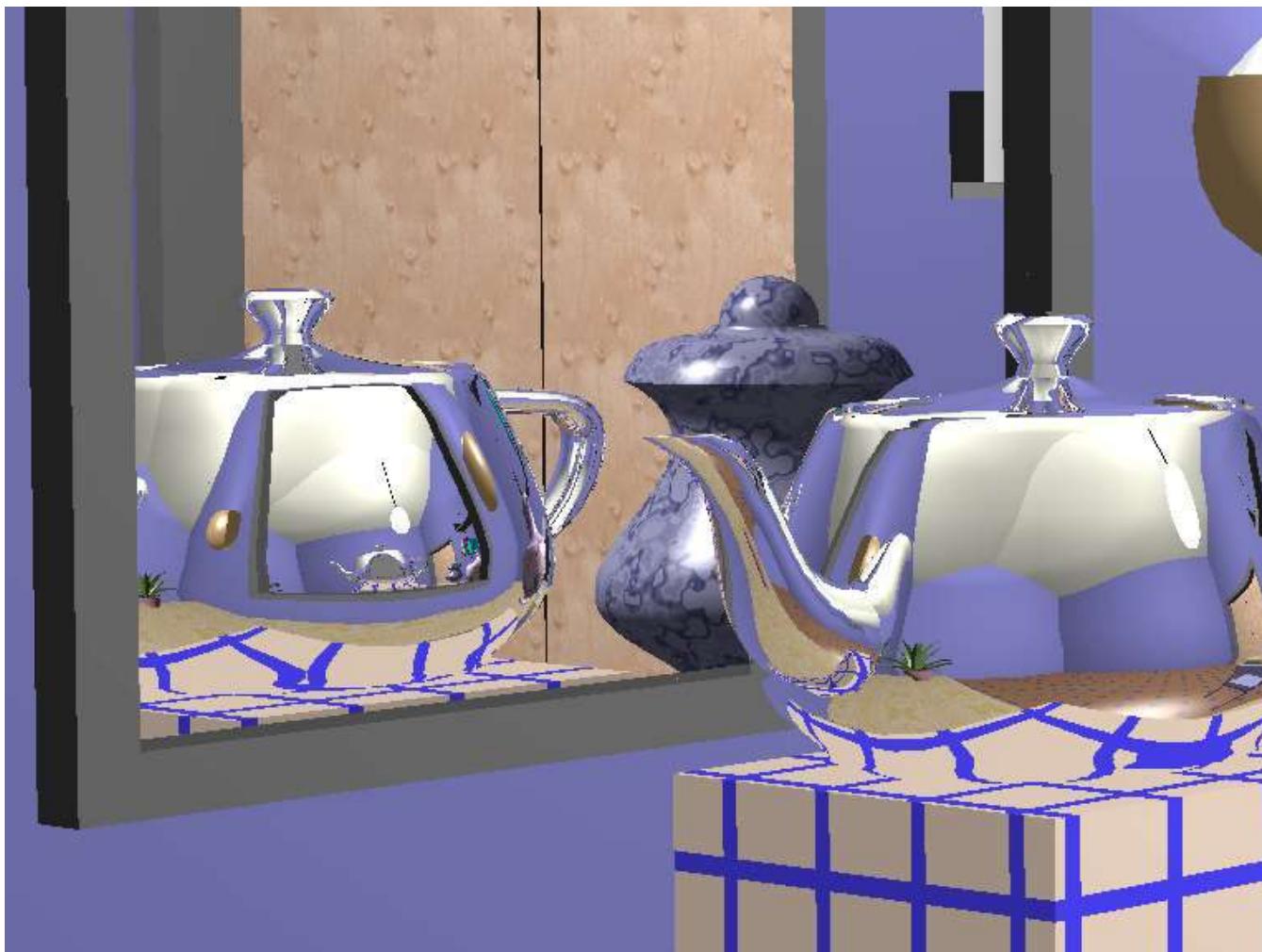
Ray-tracing clàssic. Profunditat 2



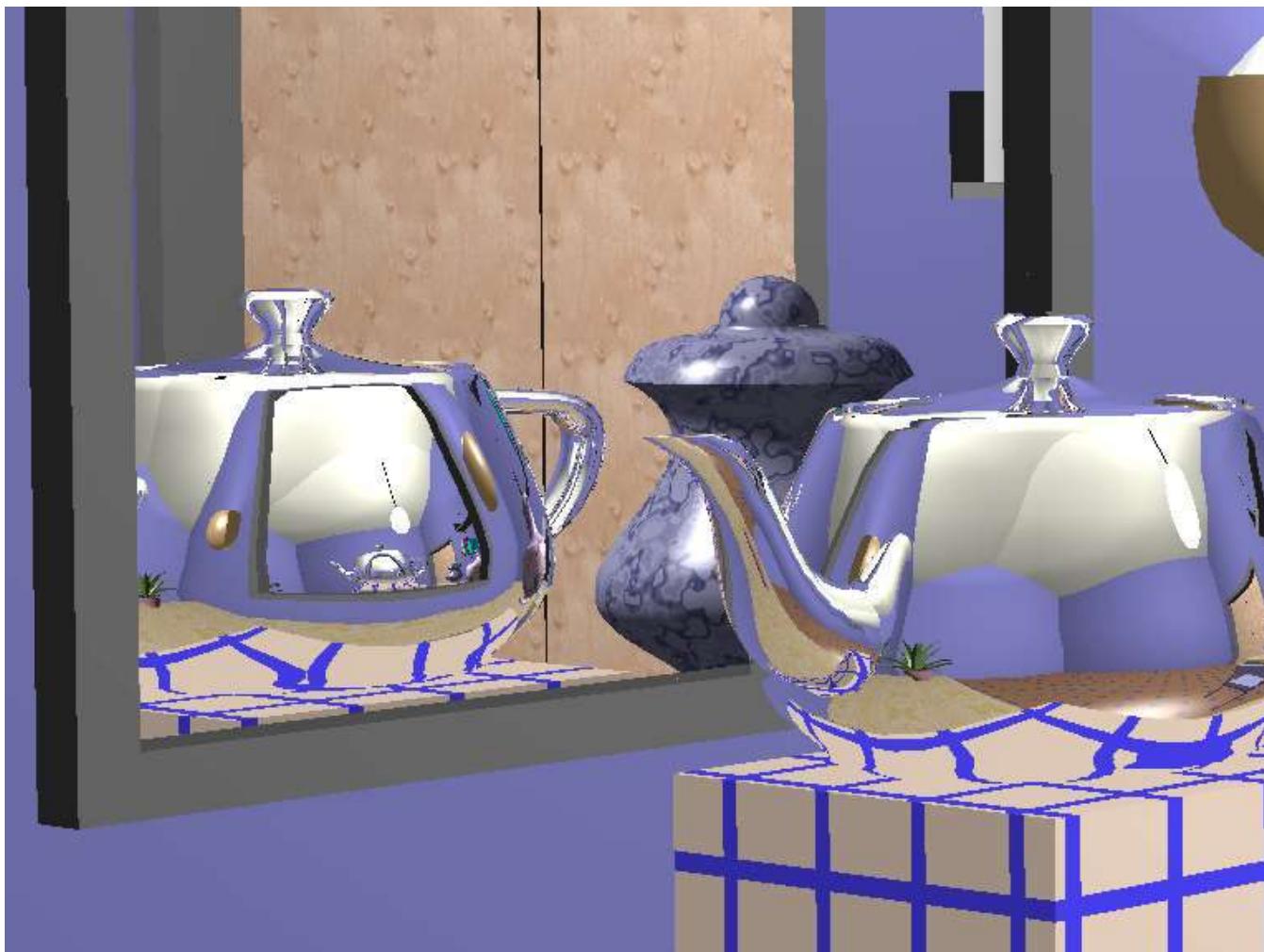
Ray-tracing clàssic. Profunditat 3



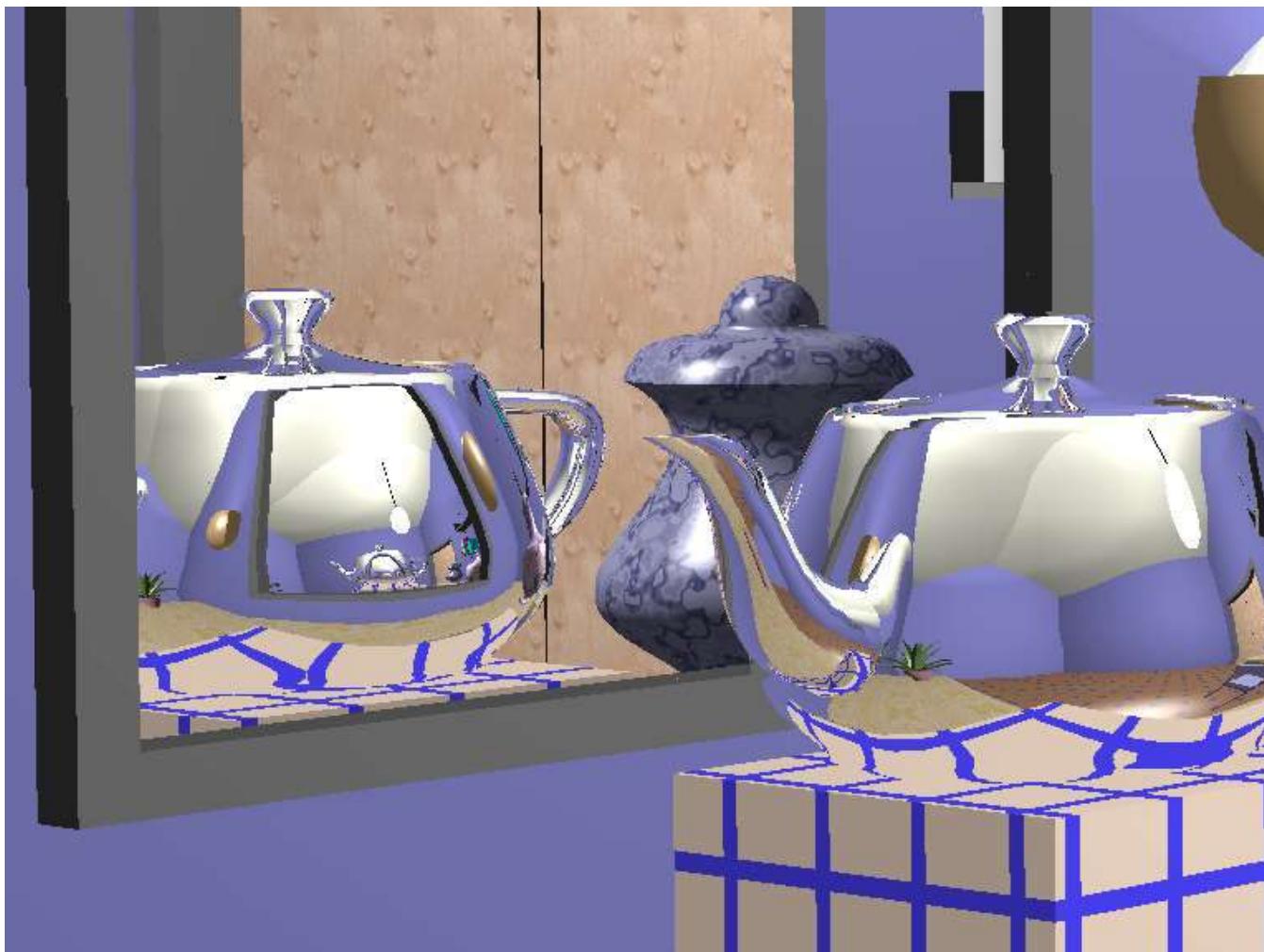
Ray-tracing clàssic. Profunditat 4



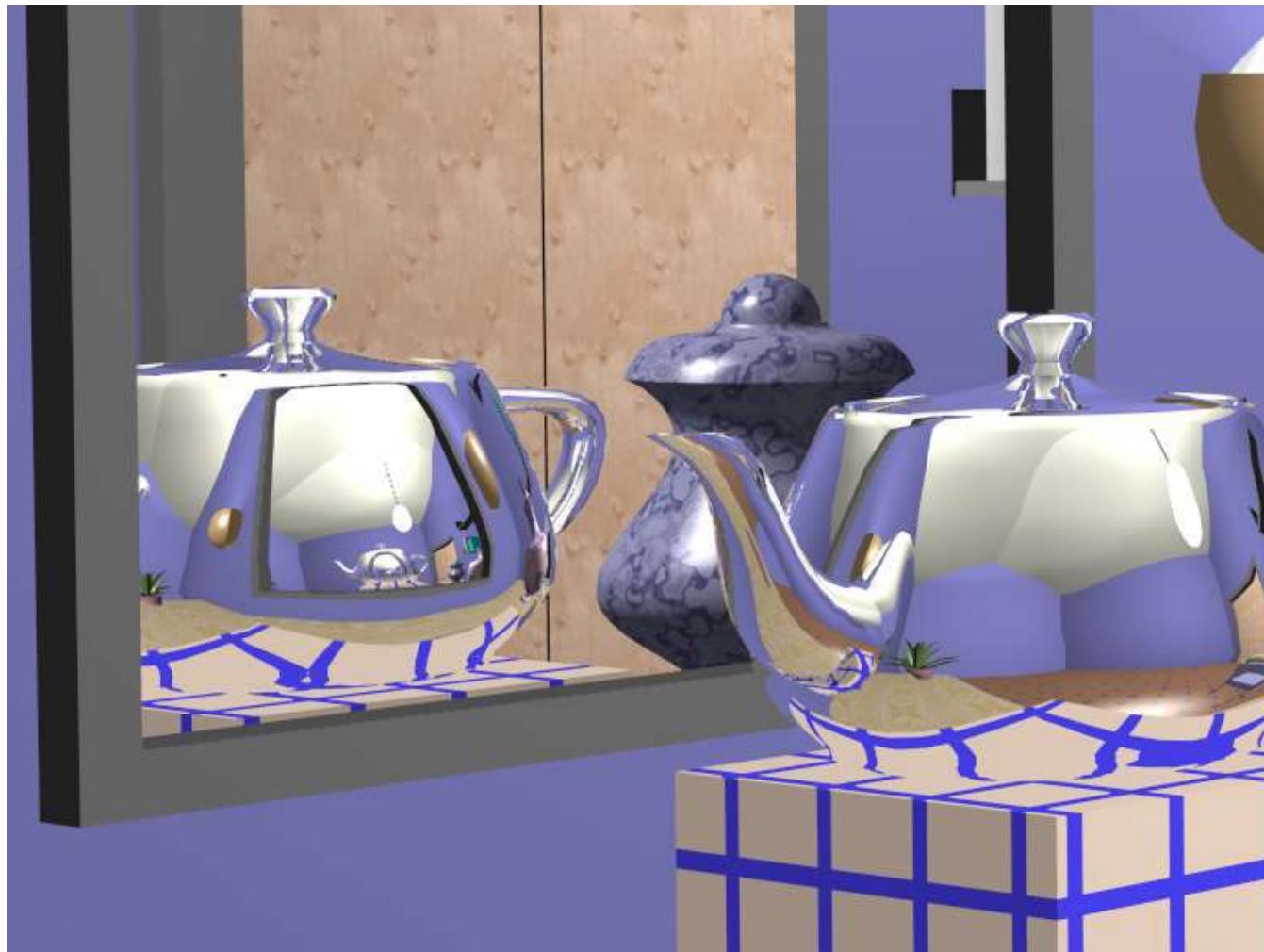
Ray-tracing clàssic. Profunditat 6



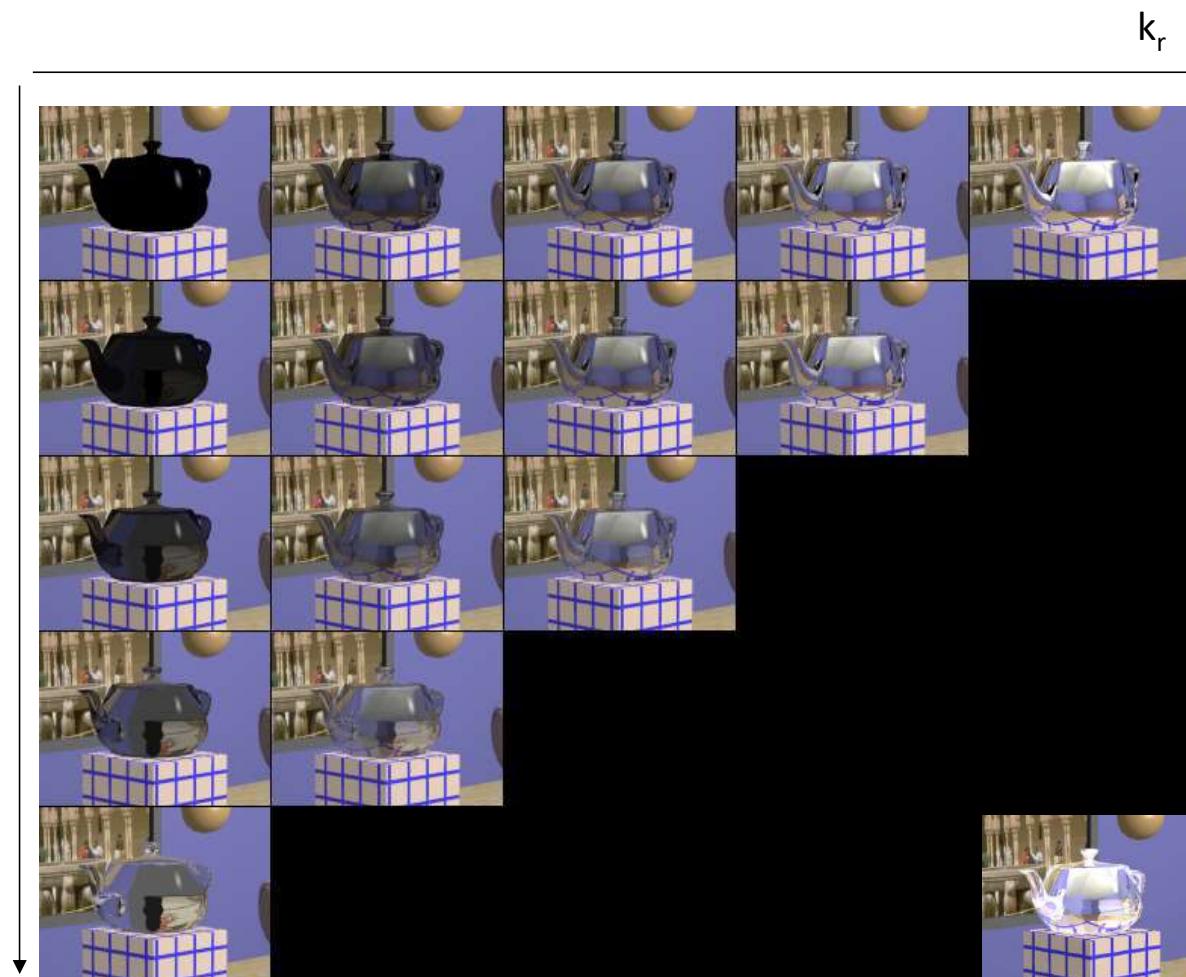
Ray-tracing clàssic. Profunditat 8

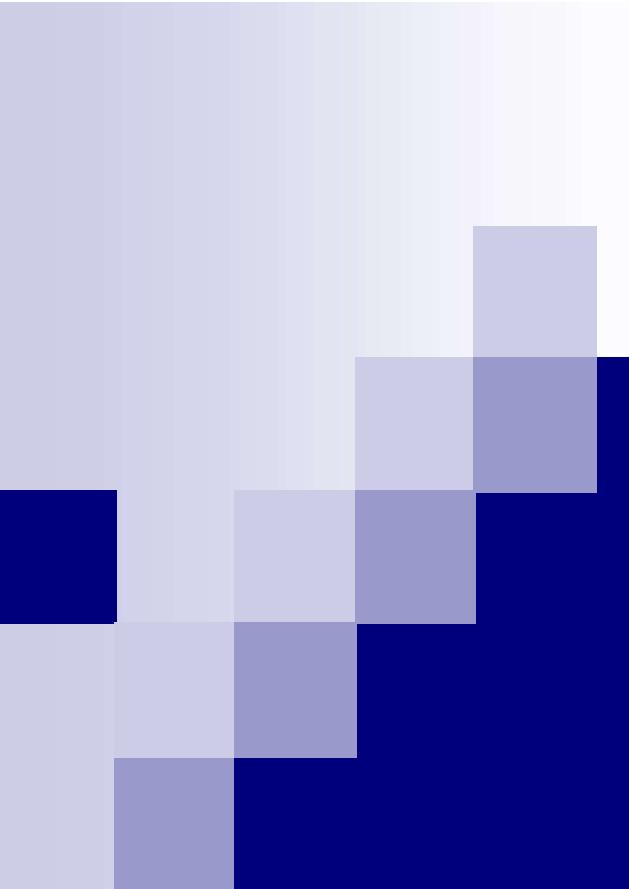


Ray-tracing clàssic. P. 8 + AA



Contribució local i global



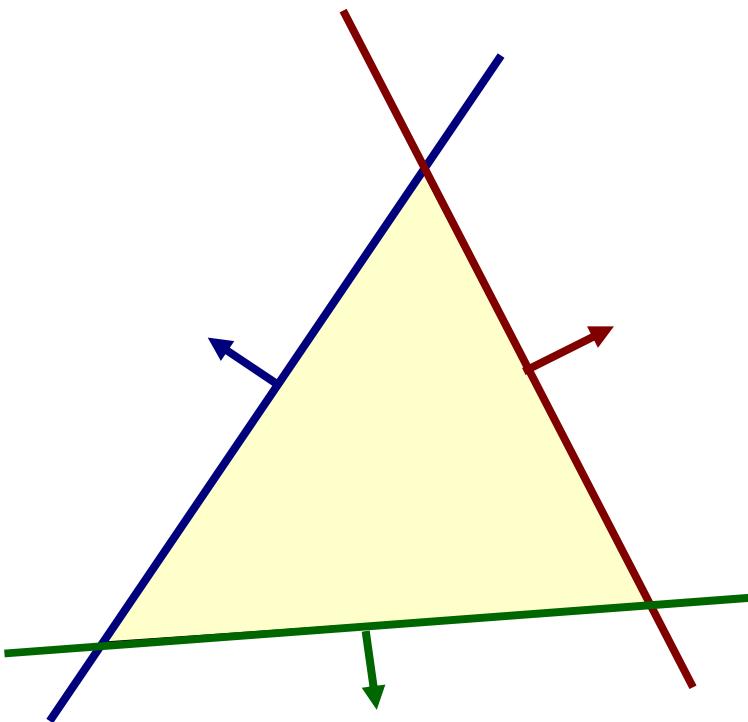


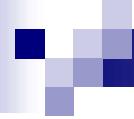
Intersecció raig-geometria

Carlos Andujar
Maig 2022

Intersecció recta-poliedre convex

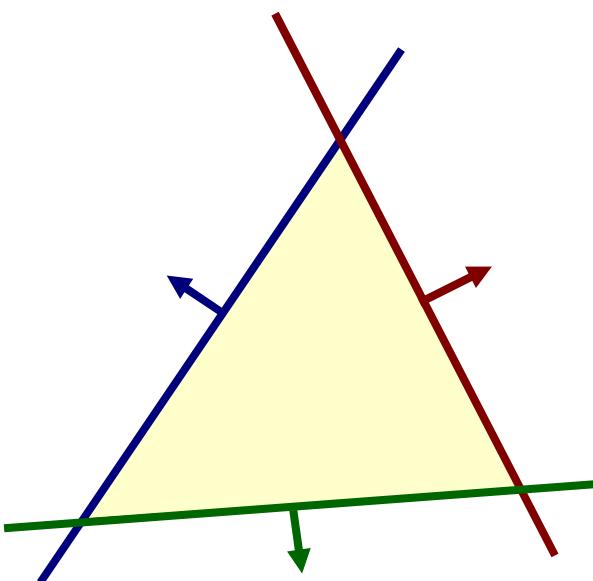
Un poliedre convex està definit per una col·lecció de plans:



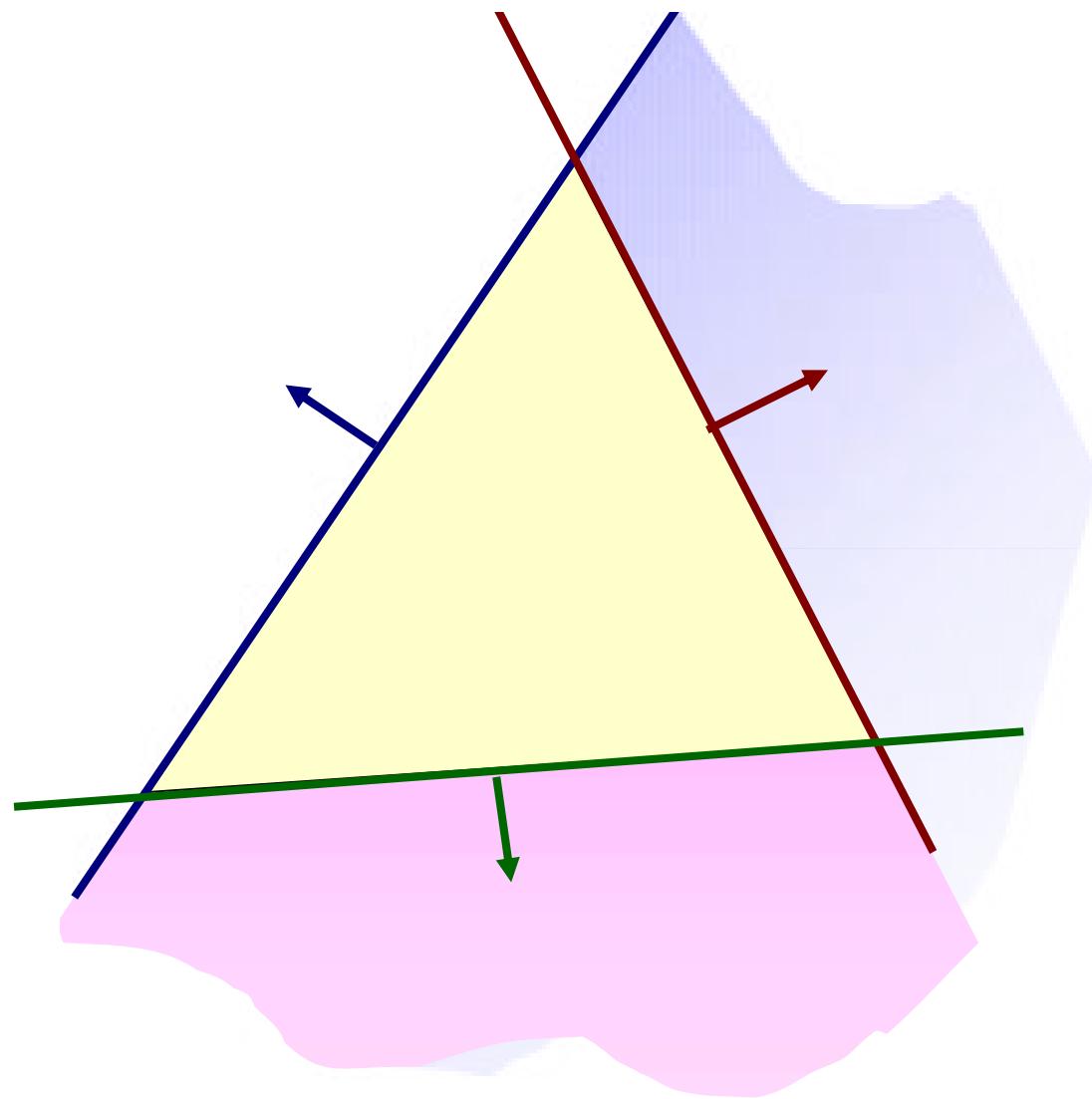


Intersecció recta-poliedre convex

L'interior del poliedre convex coincideix amb el volum intersecció dels **semi-espais negatius** dels plans:



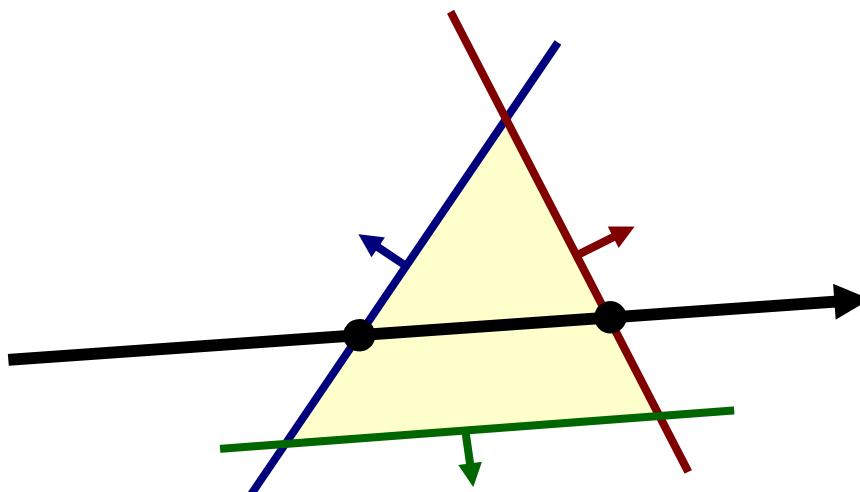
Intersecció recta-poliedre convex



Intersecció recta-poliedre convex

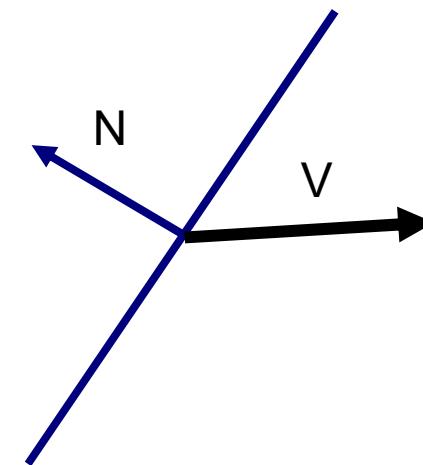
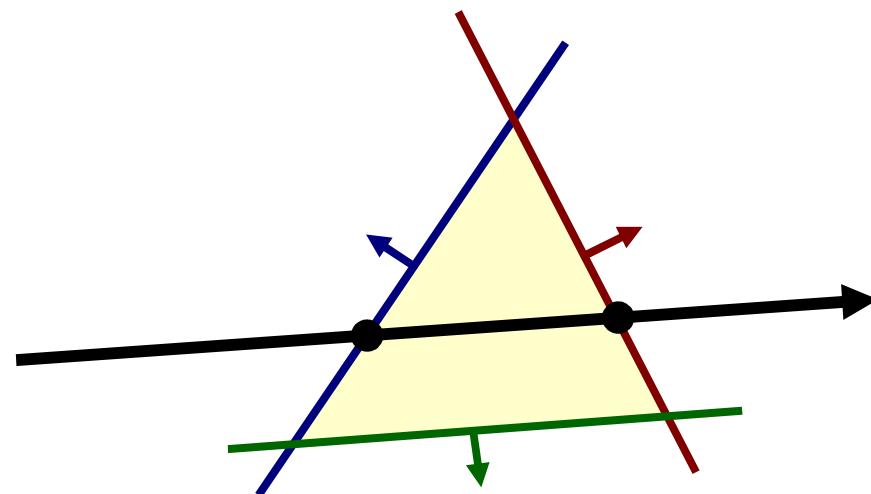
El test d'intersecció té dos possibles resultats:

- La recta no intersecta el poliedre
- La recta intersecta **dos cops** la superfície del poliedre: una intersecció d'**entrada** i una altra de **sortida**.



Intersecció recta-poliedre convex

- Les interseccions d'**entrada** sempre són amb plans **front-face** respecte la **direcció del raig**: $V \cdot N < 0$
- Les interseccions **de sortida** sempre són amb plans **back-face** respecte la **direcció del raig**: $V \cdot N > 0$

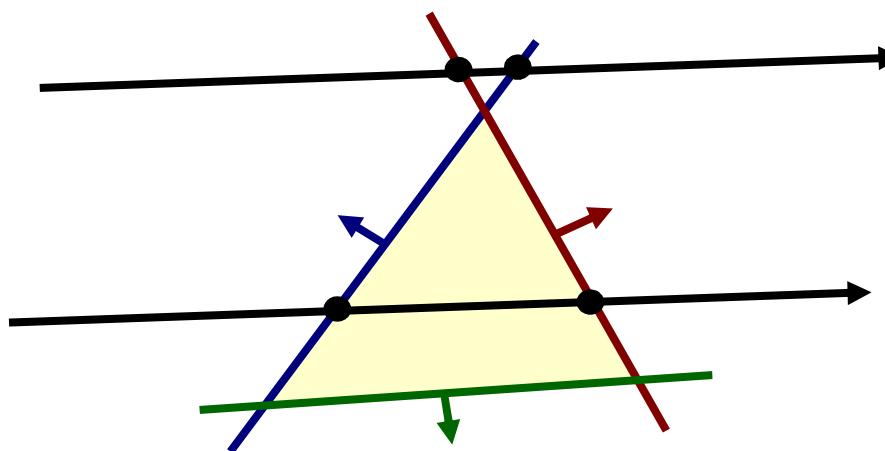


Intersecció raig-poliedre convex

L'algorisme de Haines calcula iterativament la intersecció d'entrada λ_{near} i la intersecció de sortida λ_{far}

Per cada pla Π del poliedre:

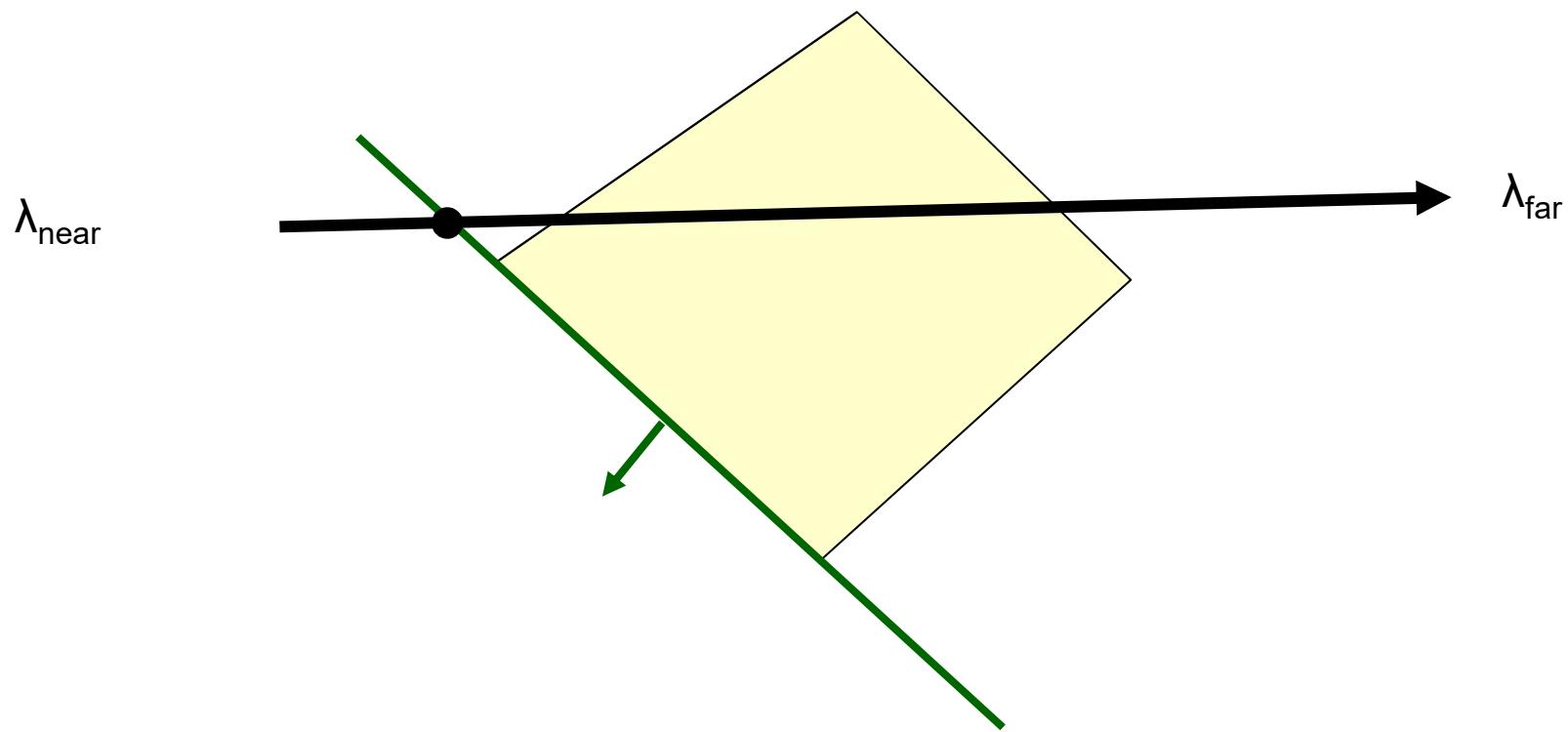
- Es calcula la intersecció de la recta amb el pla Π
- S'actualitza λ_{near} ó λ_{far} dependent de si Π és front/back.
- Si en algun moment $\lambda_{\text{near}} > \lambda_{\text{far}}$ \rightarrow no hi ha intersecció





Intersecció raig-poliedre convex

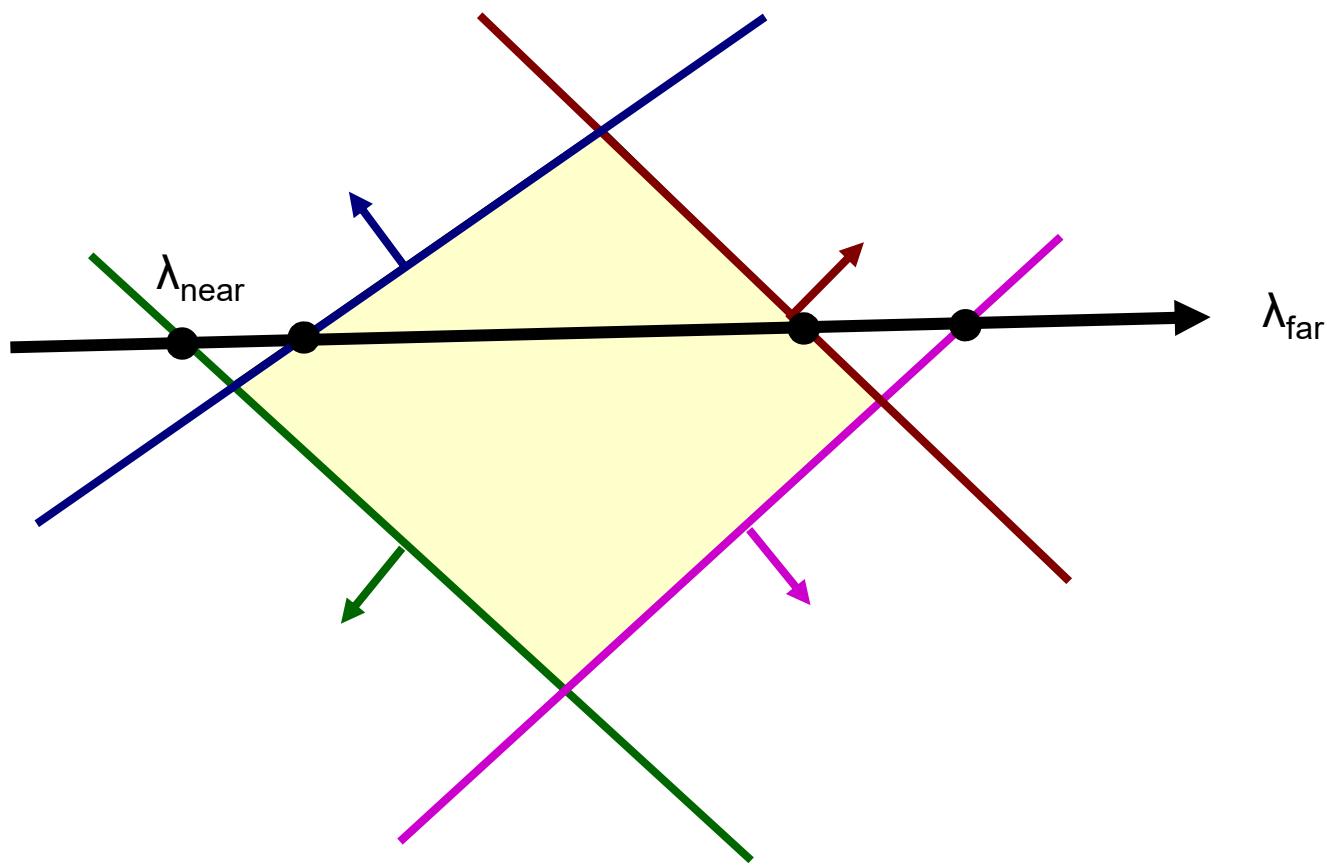
Exemple:





Intersecció raig-poliedre convex

Exemple:



Intersecció raig-poliedre convex

Exemple de no-intersecció:

