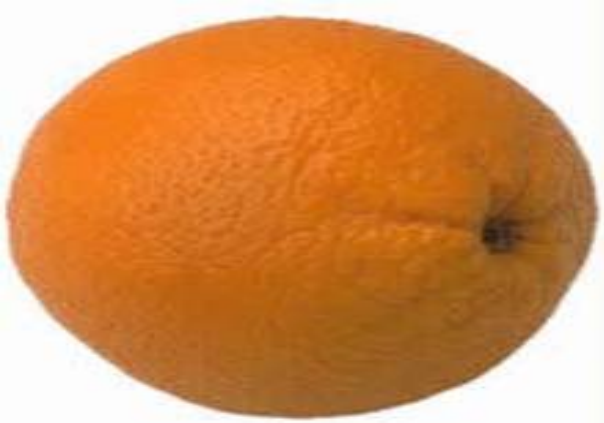
Textures

Carlos Andujar

Setembre 2015

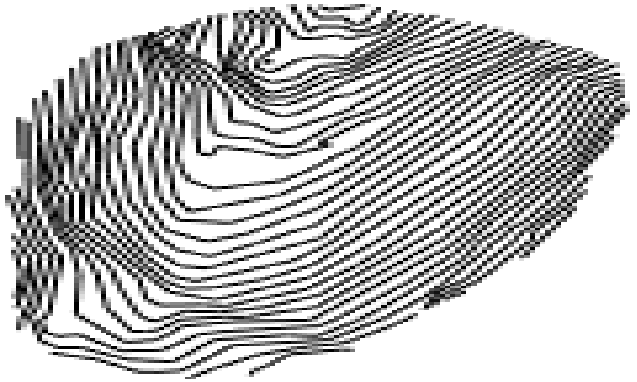
Representació de detalls superficials

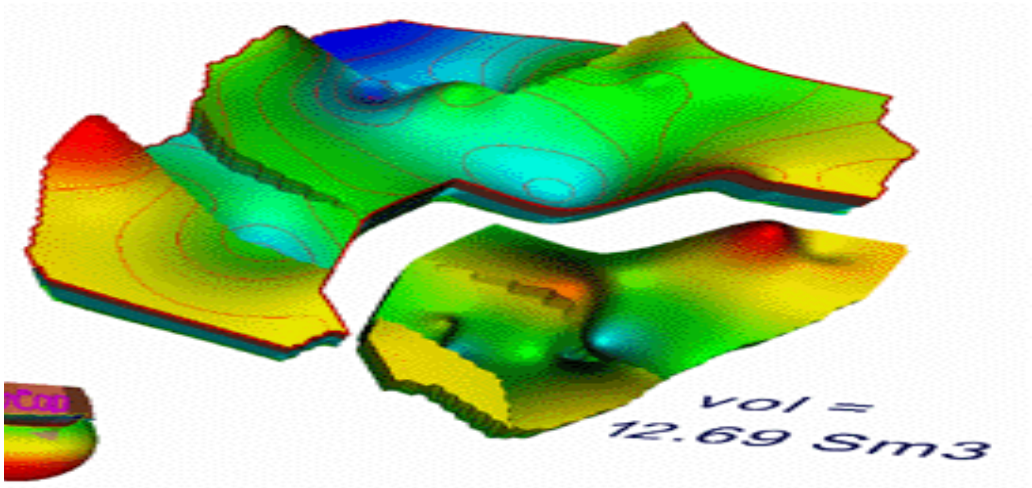


Deguts a variacions de la geometria

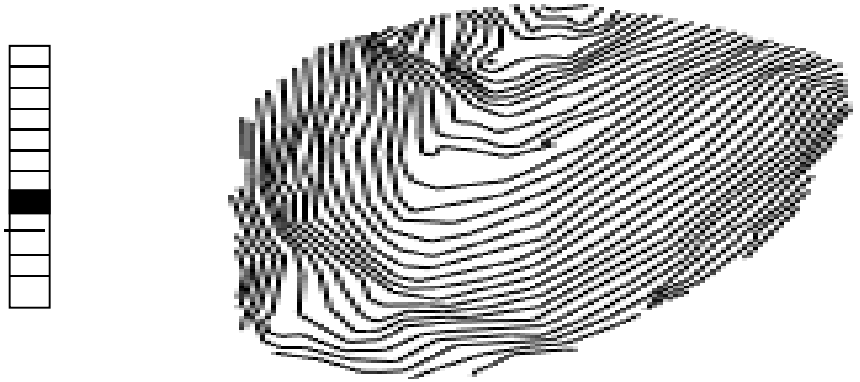
Deguts a variacions de les propietats òptiques

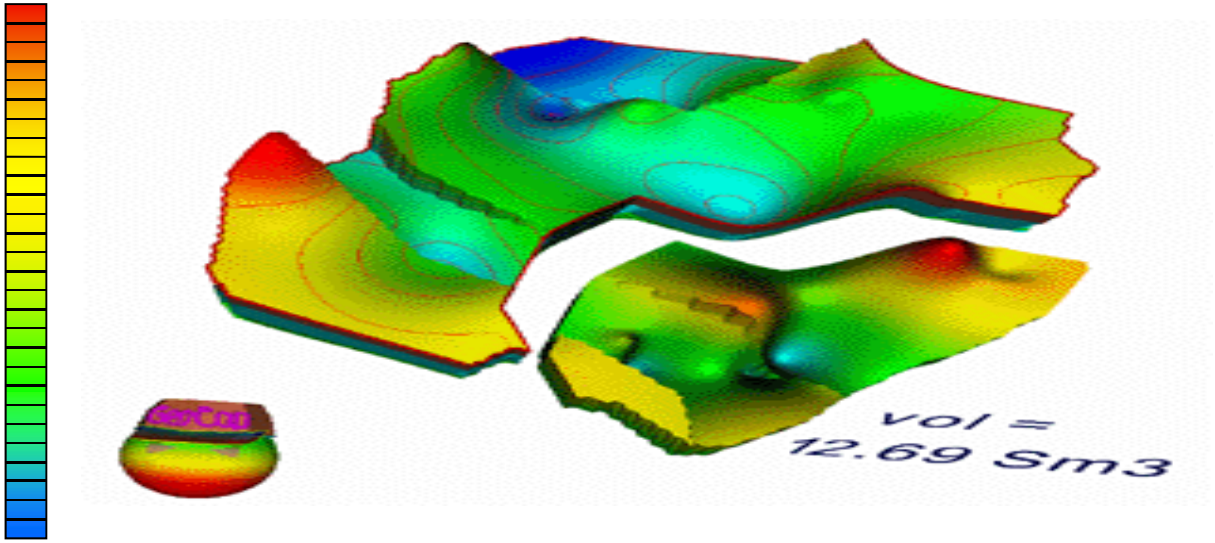
Dimensió de les textures. Textures 1D



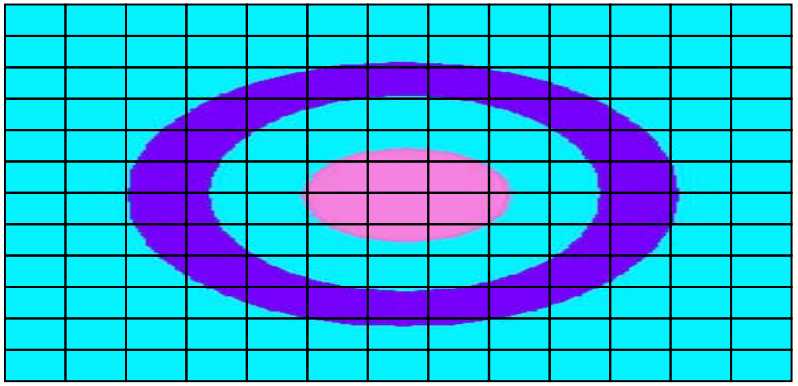


Dimensió de les textures. Textures 1D

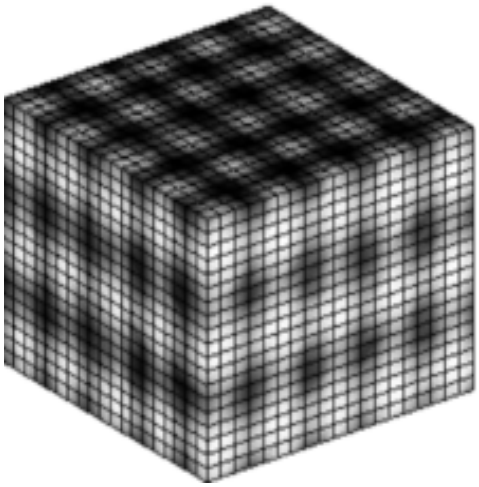


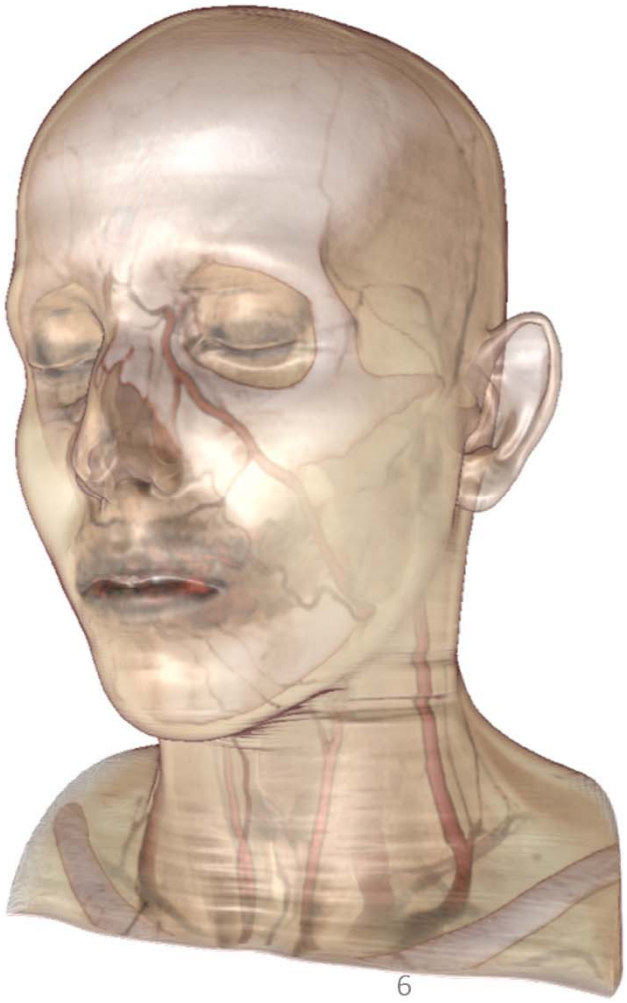


Dimensió de les textures. Textures 2D

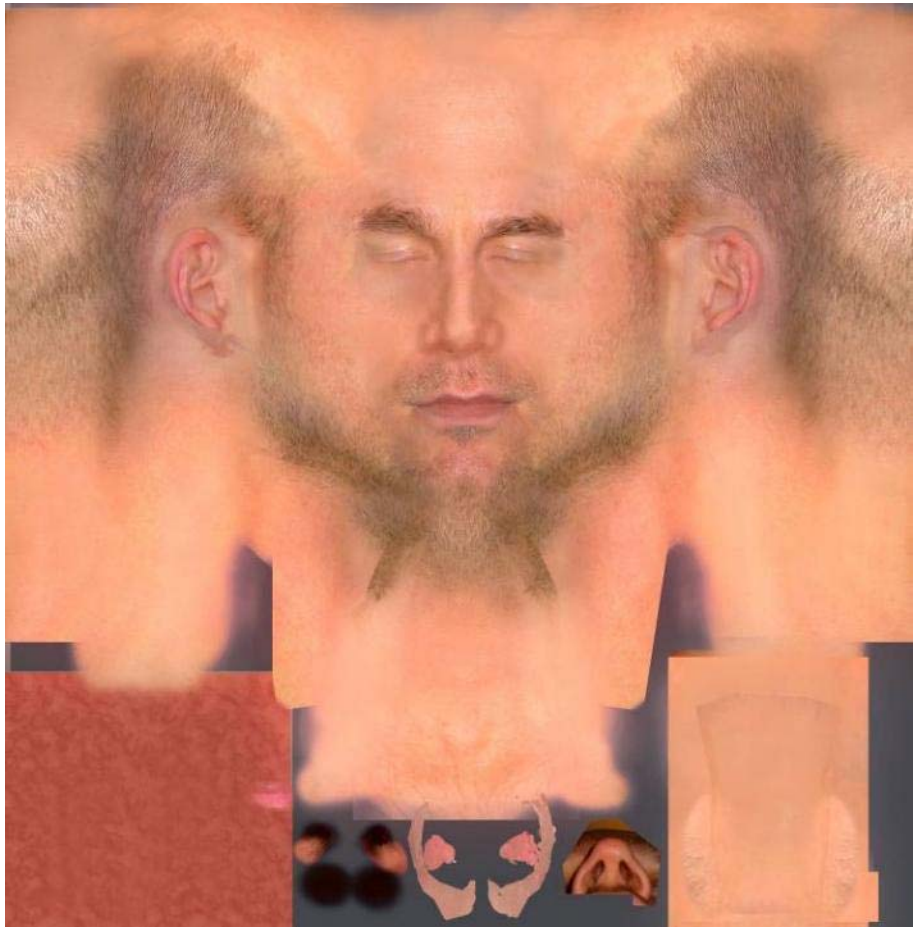


Dimensió de les textures. Textures 3D

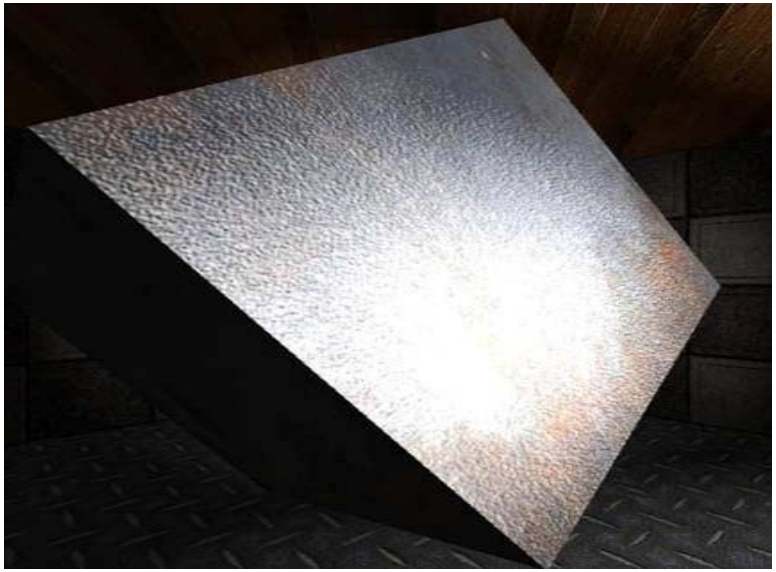




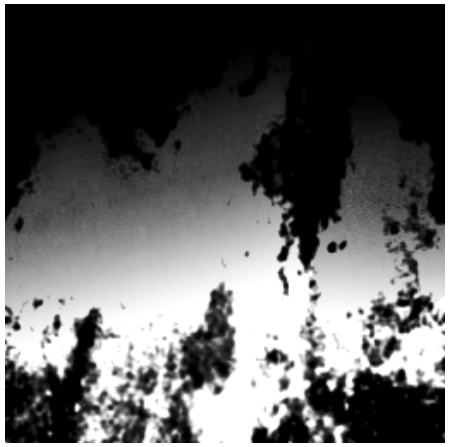
Tipus de textures: K_d (color map)

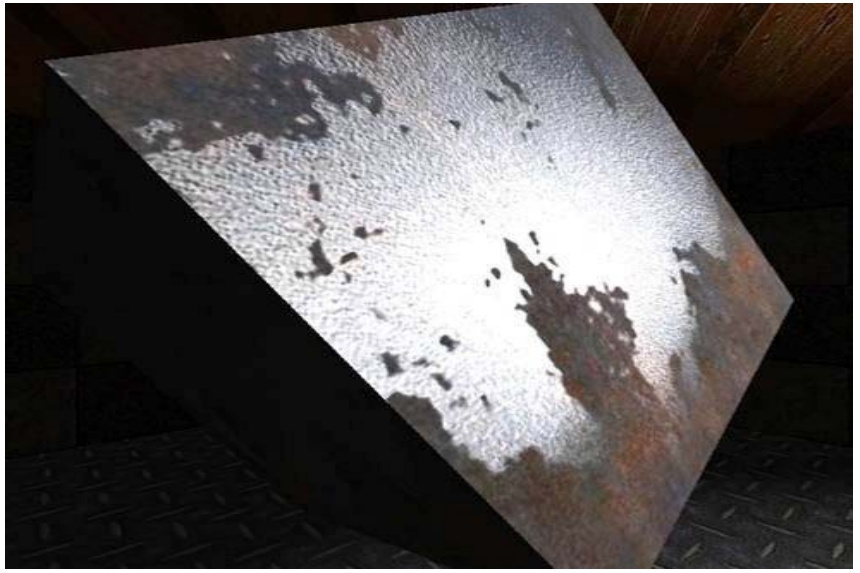


Tipus de textures: K_s(gloss map)

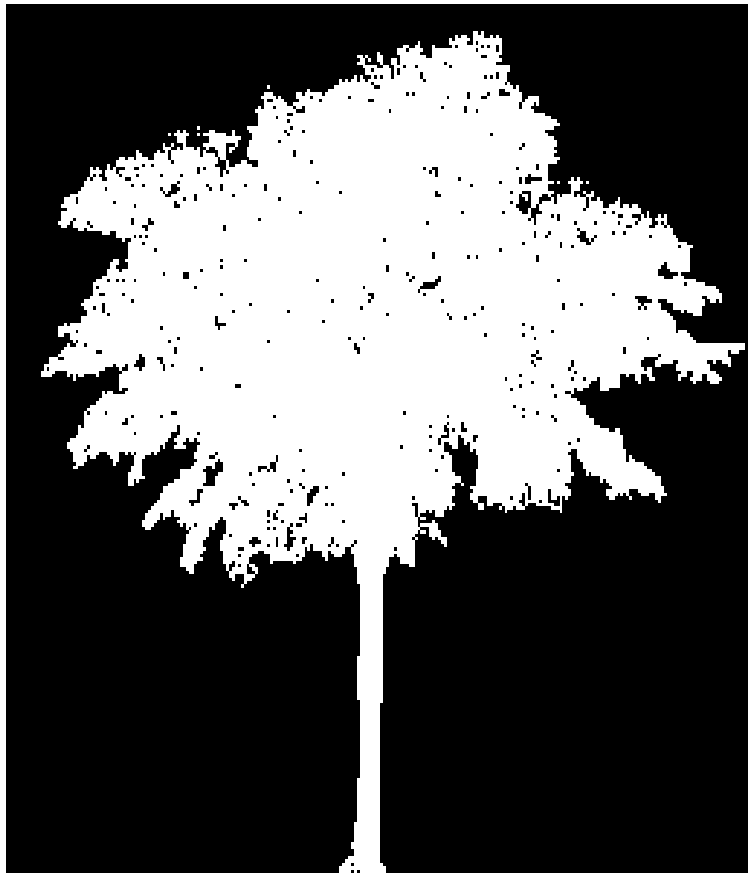


Tipus de textures: K_s(gloss map)

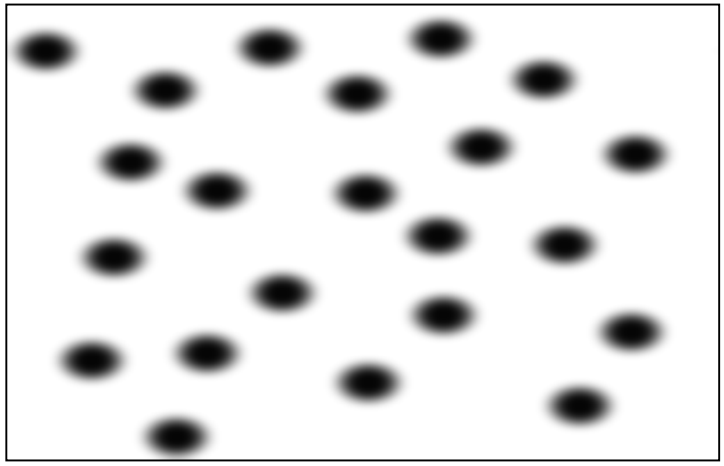


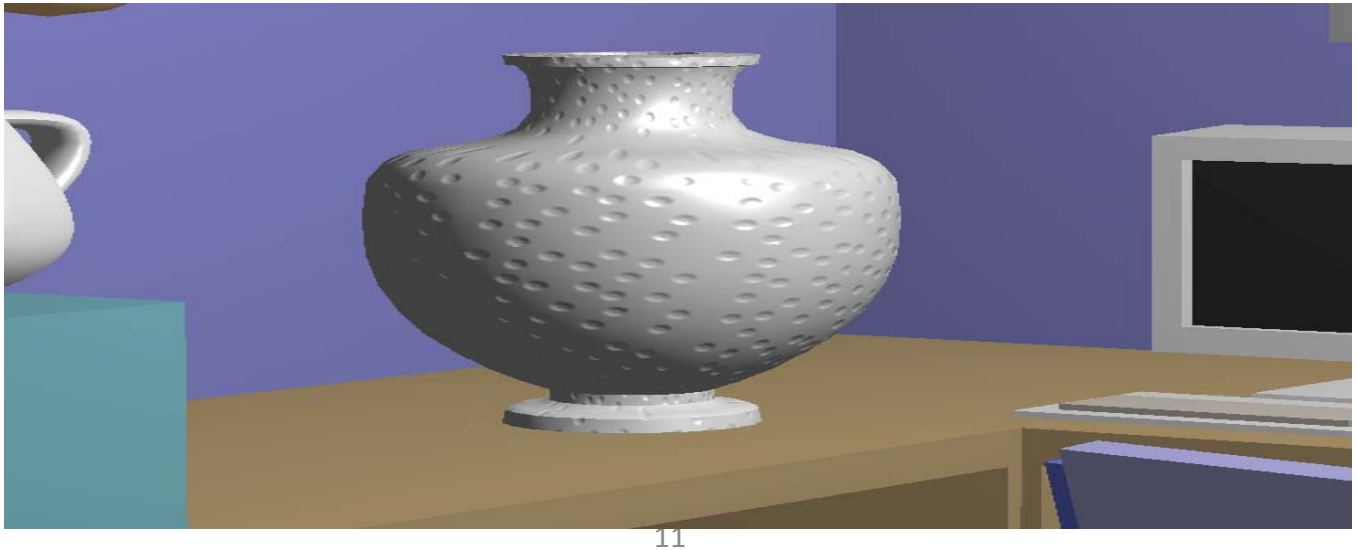


Tipus de textures: alpha (opacity map)

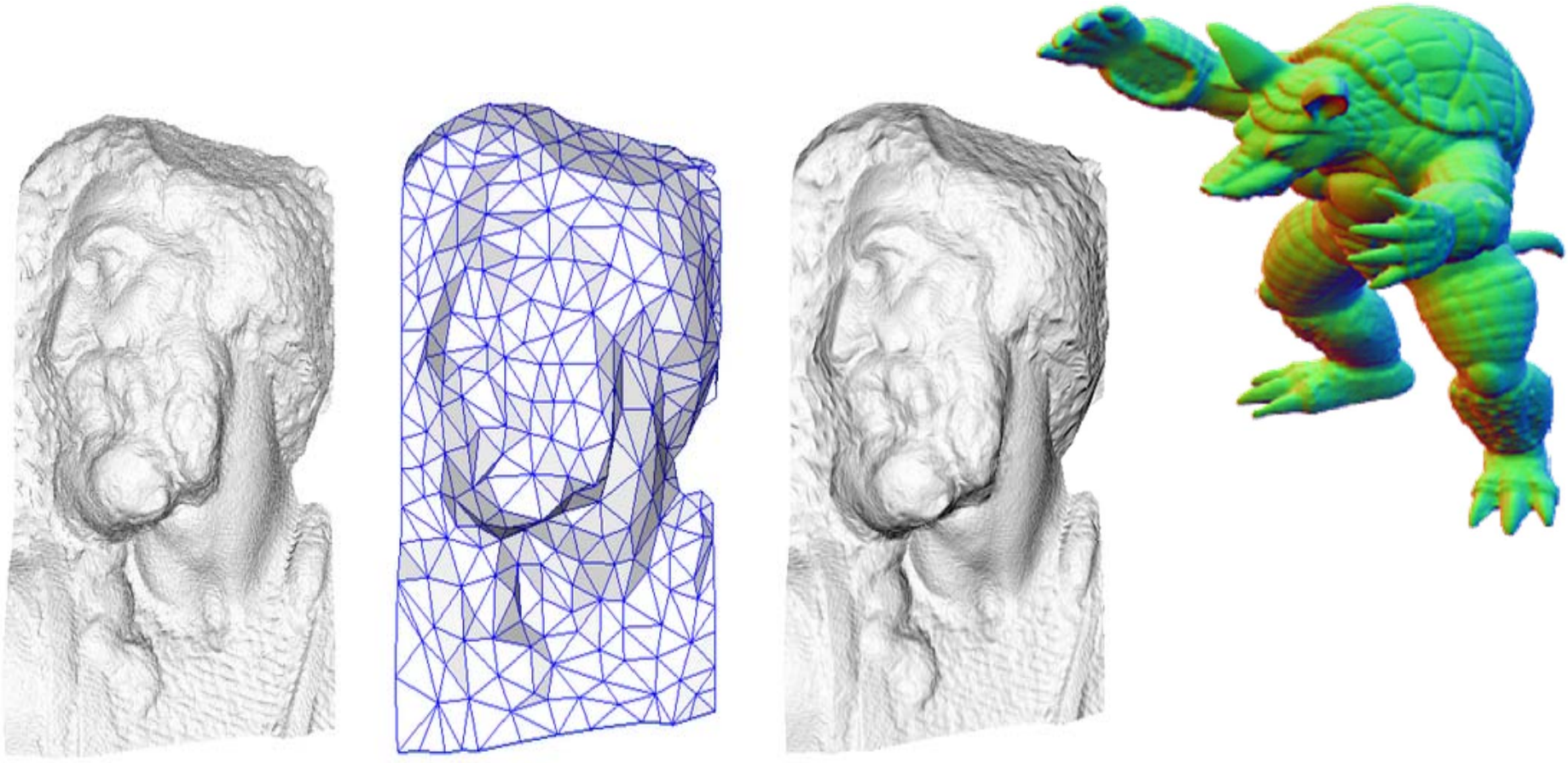


Tipus de textures: bump mapping





Tipus de textures: normal mapping

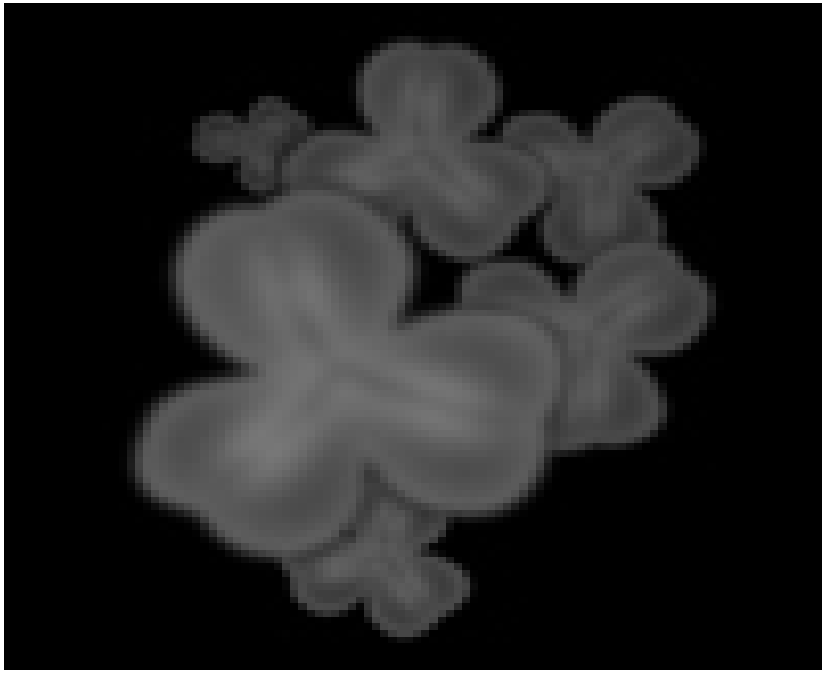


original mesh
4M triangles

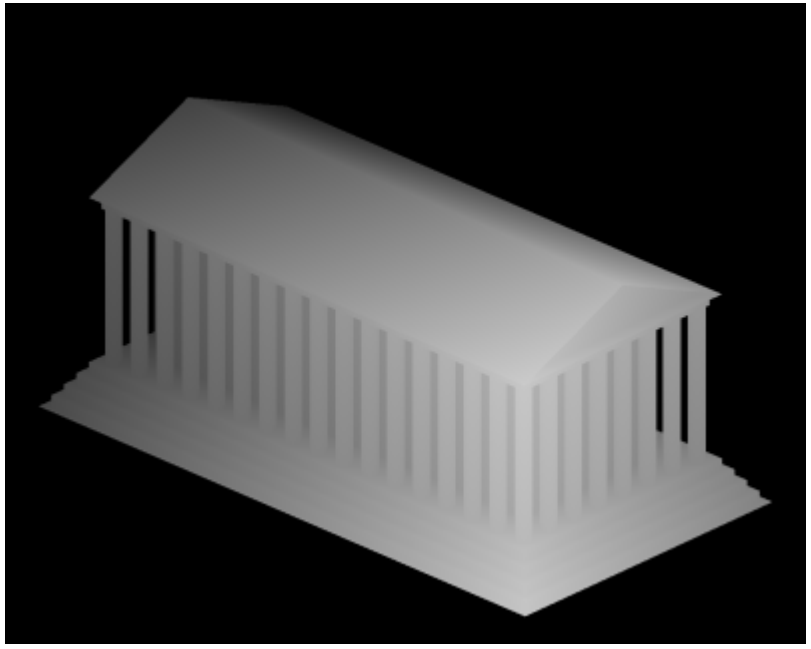
simplified mesh
500 triangles

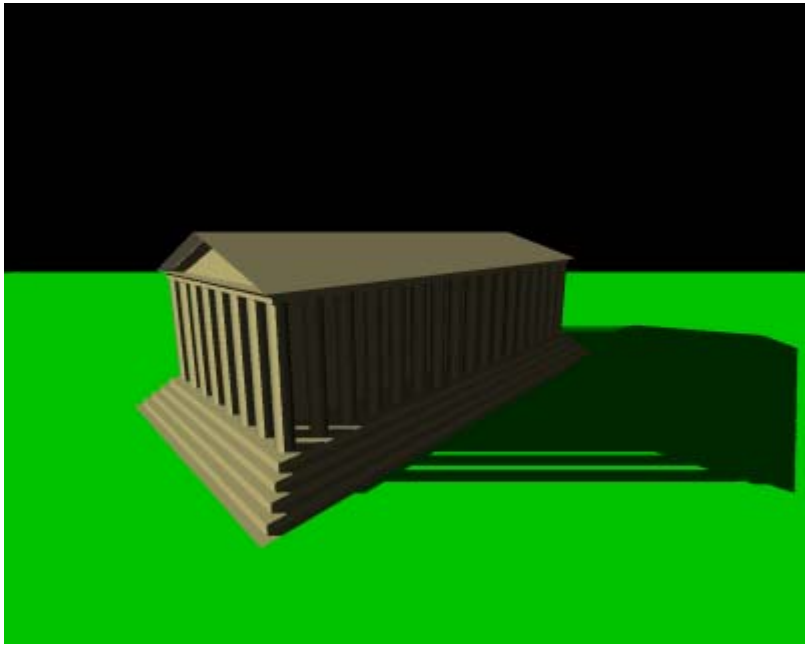
simplified mesh
and normal mapping
500 triangles

Tipus de textures: displacement map

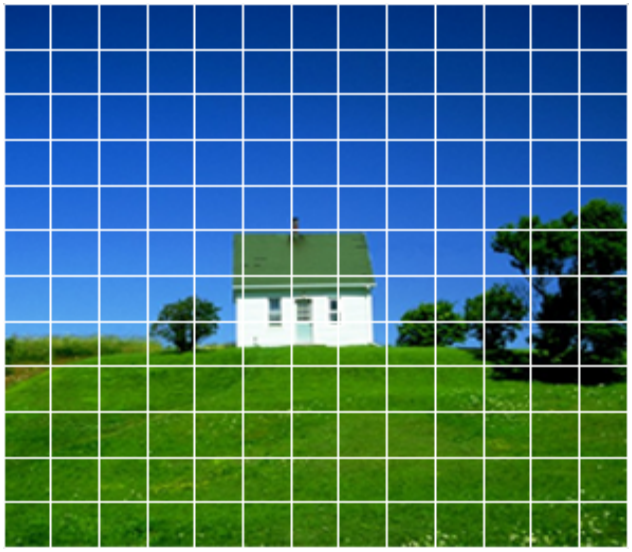


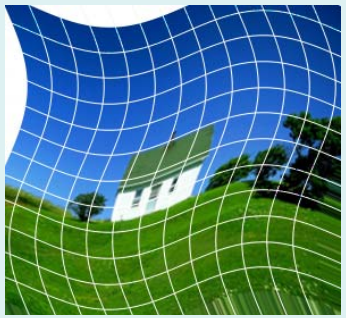
Tipus de textures: shadow maps

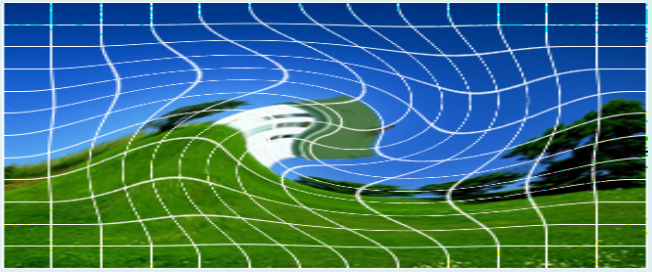


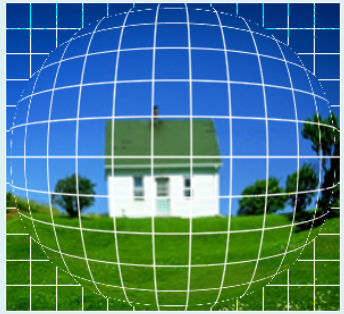


Mapping directe: image warping









Exemple 1: Mapping esfèric

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

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

// pas (s, t) → (Θ, Ψ)

$\Theta = 2\pi s;$

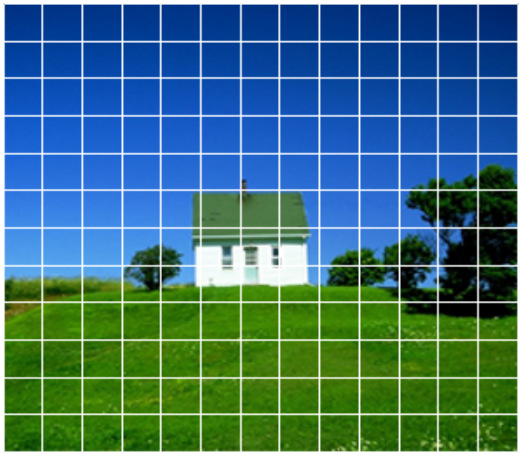
$\Psi = \pi(t-0.5);$

// pas esfèriques → (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) → (Θ, h)

$\Theta = 2\pi s;$

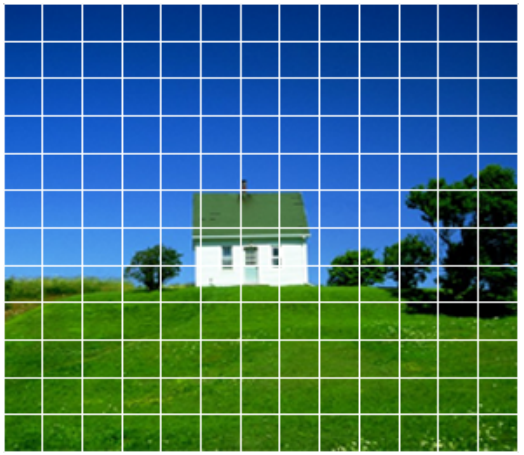
$h = t;$

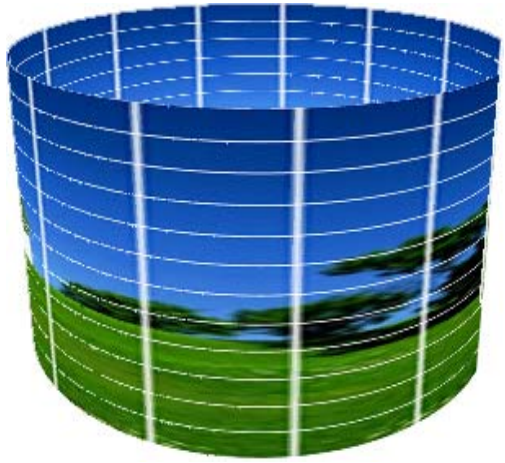
// pas cilíndriques → (x,y,z)

$x = \sin(\Theta);$

$y = h;$

$z = \cos(\Theta);$

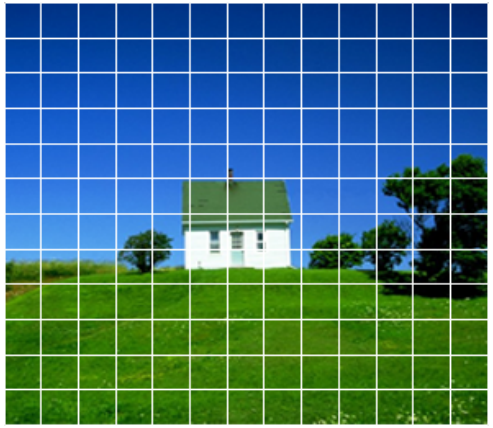




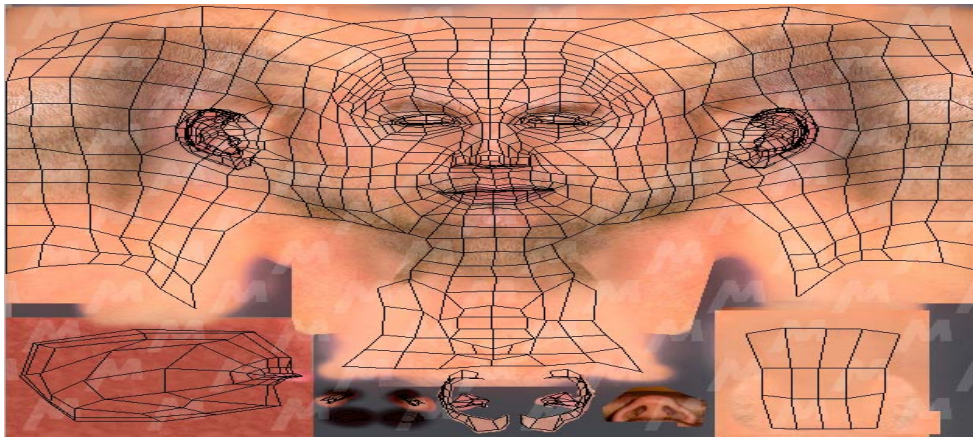
Exemple 3: Swirl

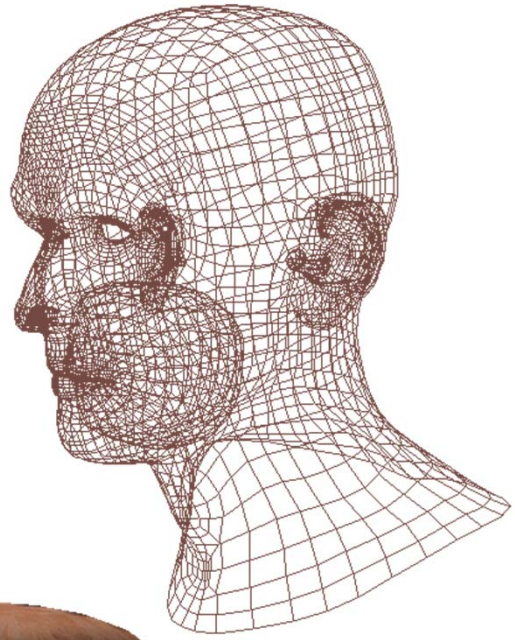
Input: $s \in [0,1]$, $t \in [0,1]$
Output: $x \in [0,1]$, $y \in [0,1]$

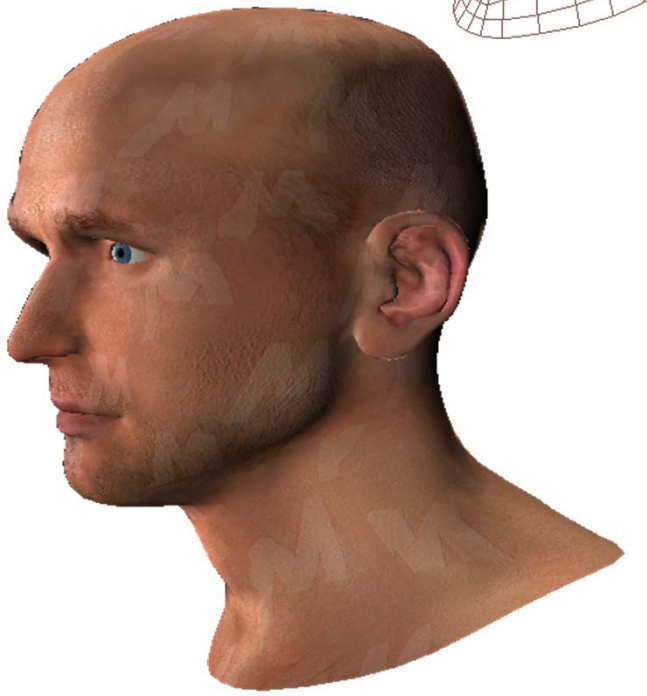
```
dist = distance(s, t, xc, yc);
angle = angleMax * (1-dist);
// rotació respecte (xc, yc)
s -= xc;
t -= yc;
x = s*cos(angle) - t*sin(angle);
y = s*sin(angle) + t*cos(angle);
x += xc;
y += yc;
```



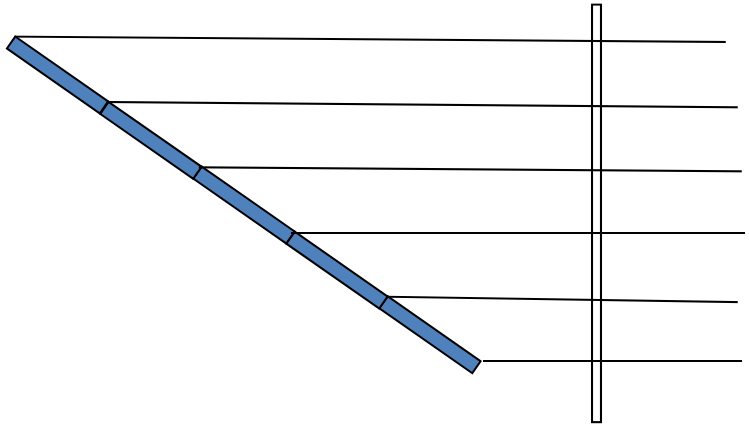
Mapping invers

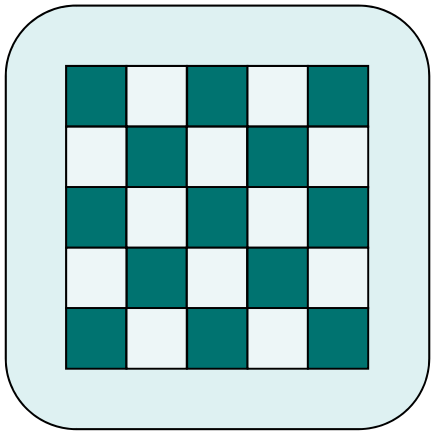




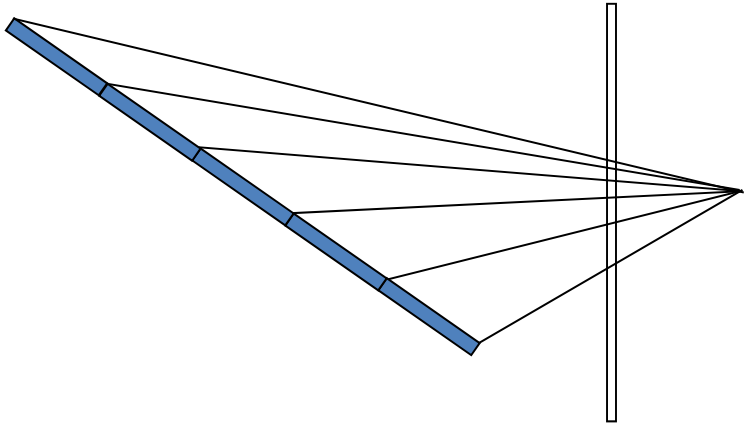


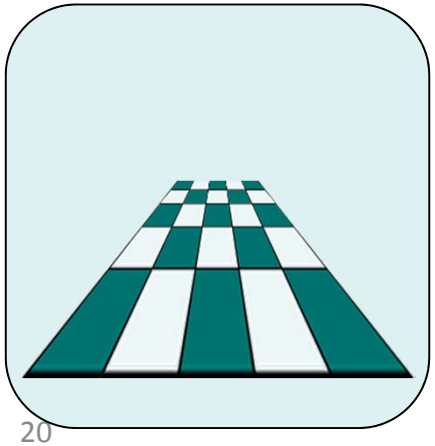
Interpolació segons la projecció





Axonomètrica

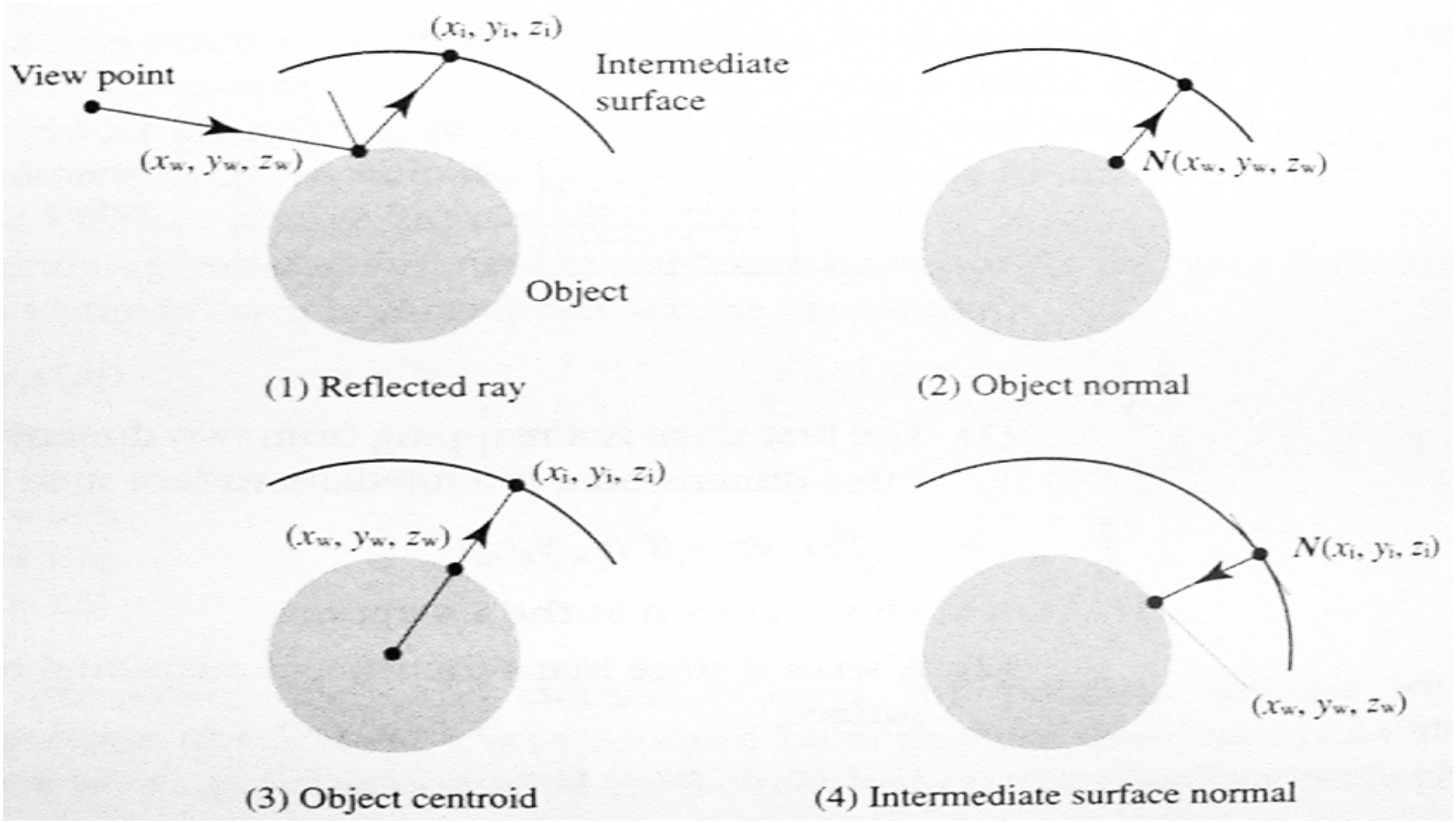




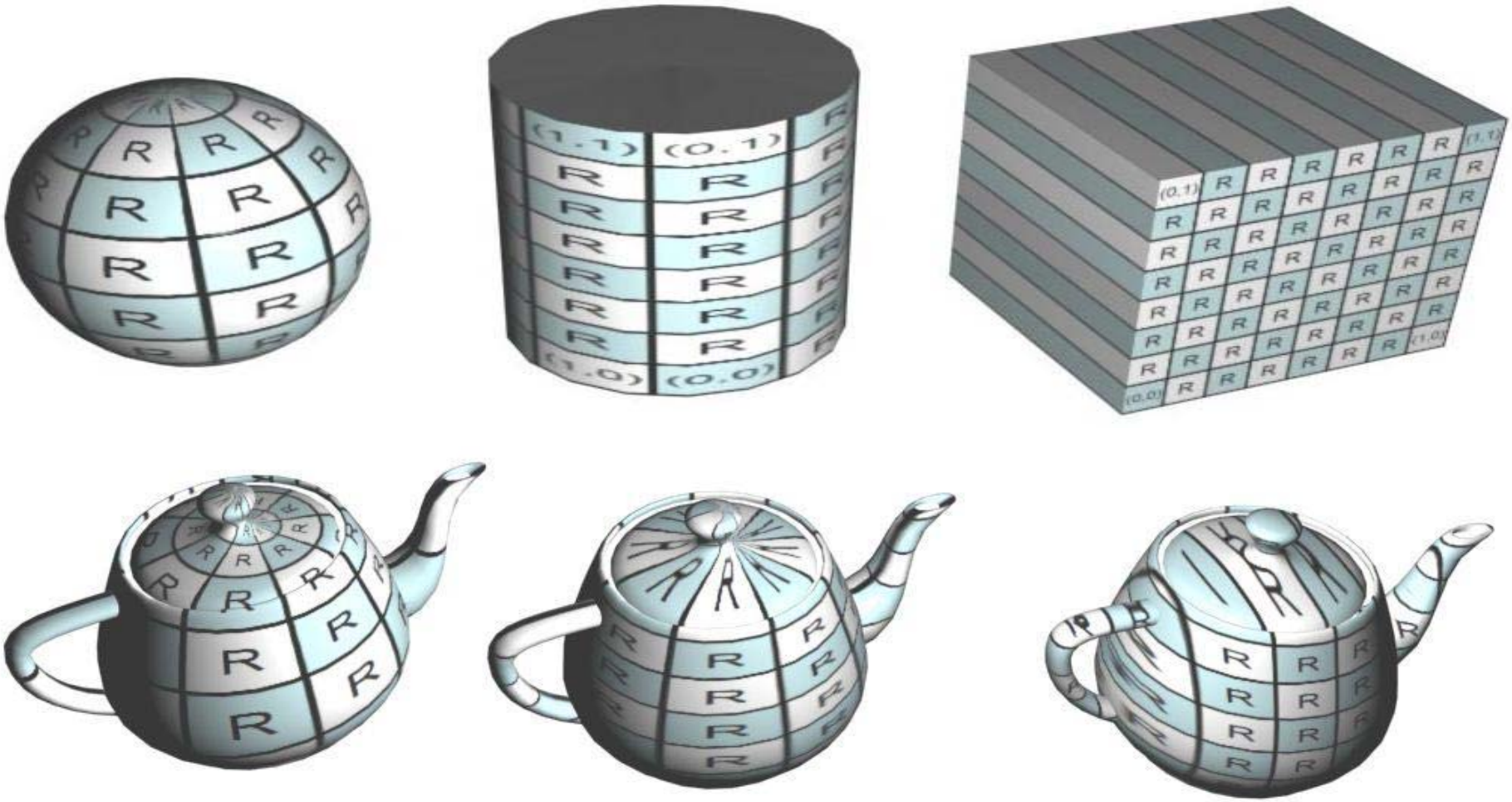
Perspectiva

GENERACIÓ DE COORDENADES DE TEXTURA

O mappings


(x_i, y_i, z_i)
View point
Intermediate surface
(x_w, y_w, z_w)
Object
(1) Reflected ray
$N(x_w, y_w, z_w)$
(2) Object normal
(x_i, y_i, z_i)
(x_w, y_w, z_w)
(3) Object centroid
$N(x_i, y_i, z_i)$
(x_w, y_w, z_w)
(4) Intermediate surface normal

Projeccions esfèrica, cilíndrica i plana



Magnification filters, Minification filters, Mipmapping

FILTRAT

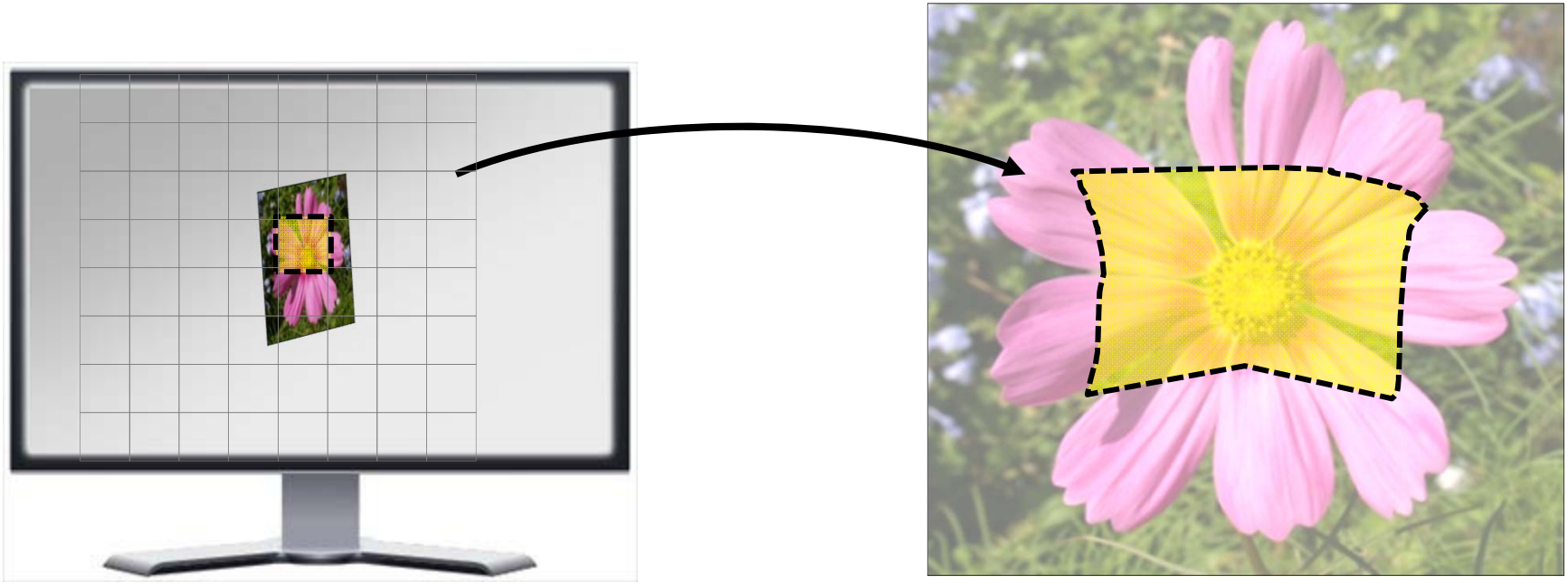
Necessitat del filtrat

Textura

Magnification ≈ upsampling

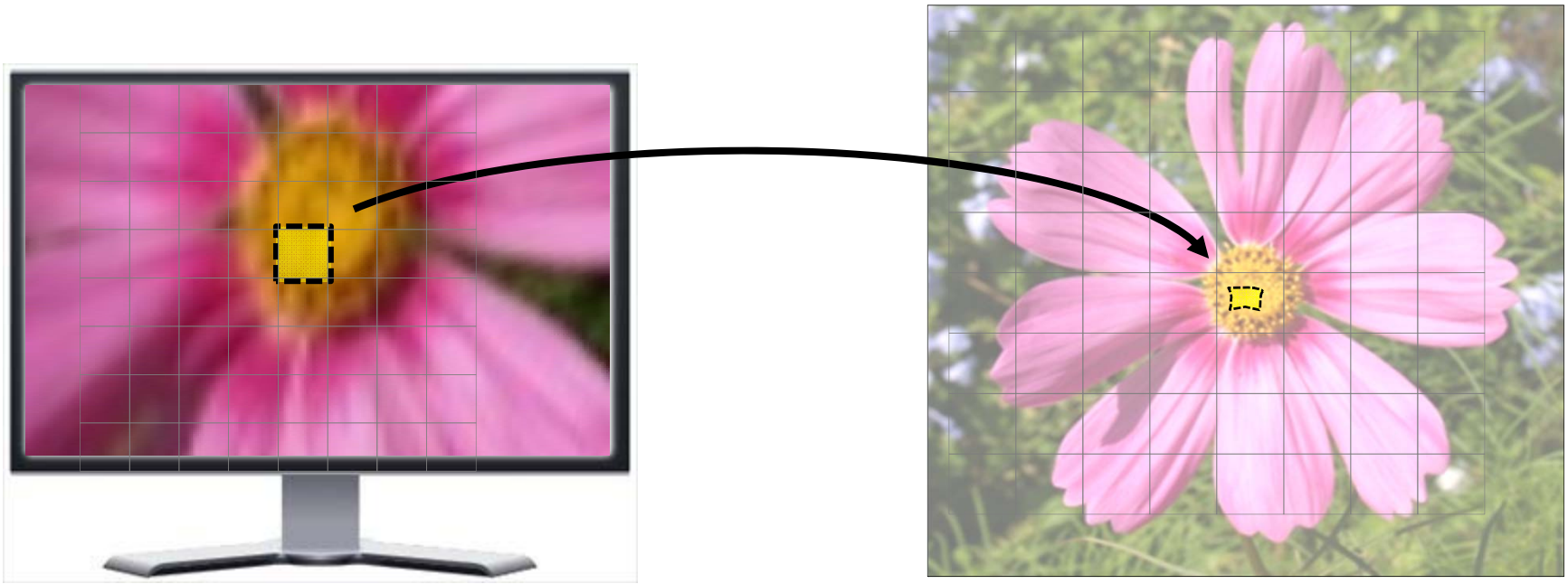
Minification ≈ downsampling

Necessitat del filtrat



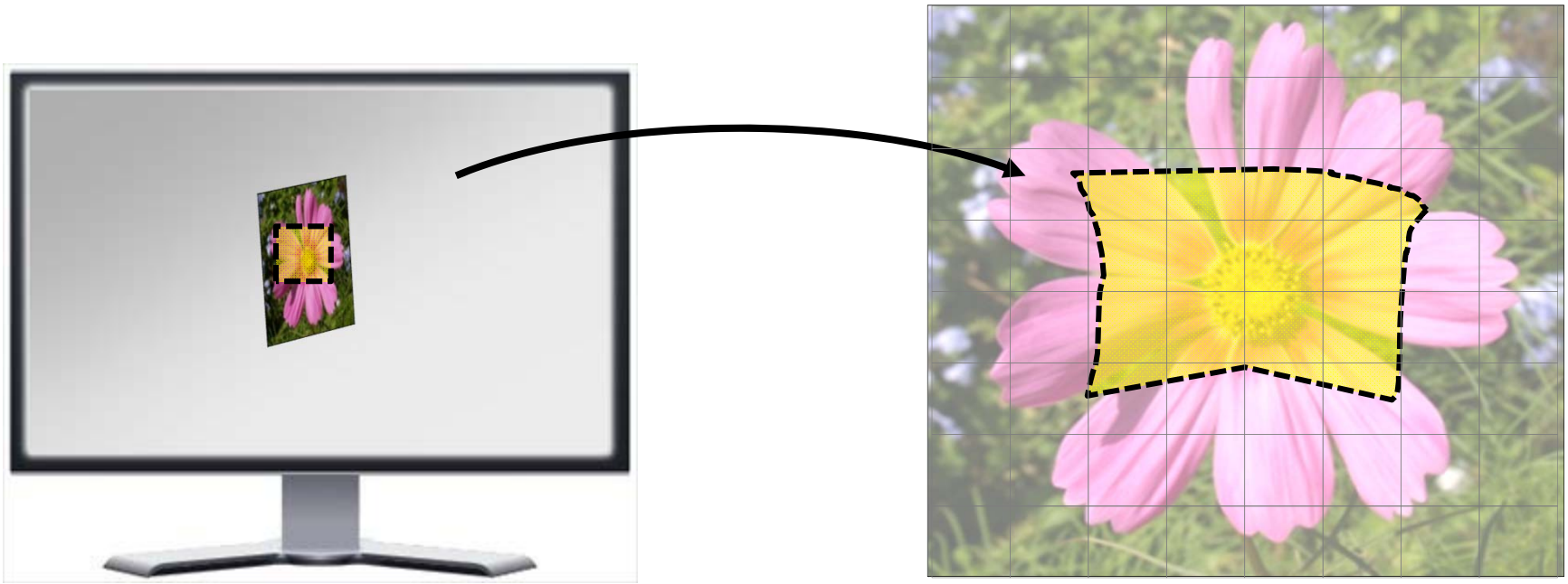
Ideal: color d'un pixel → color de la seva *preimatge* a la textura

Magnification



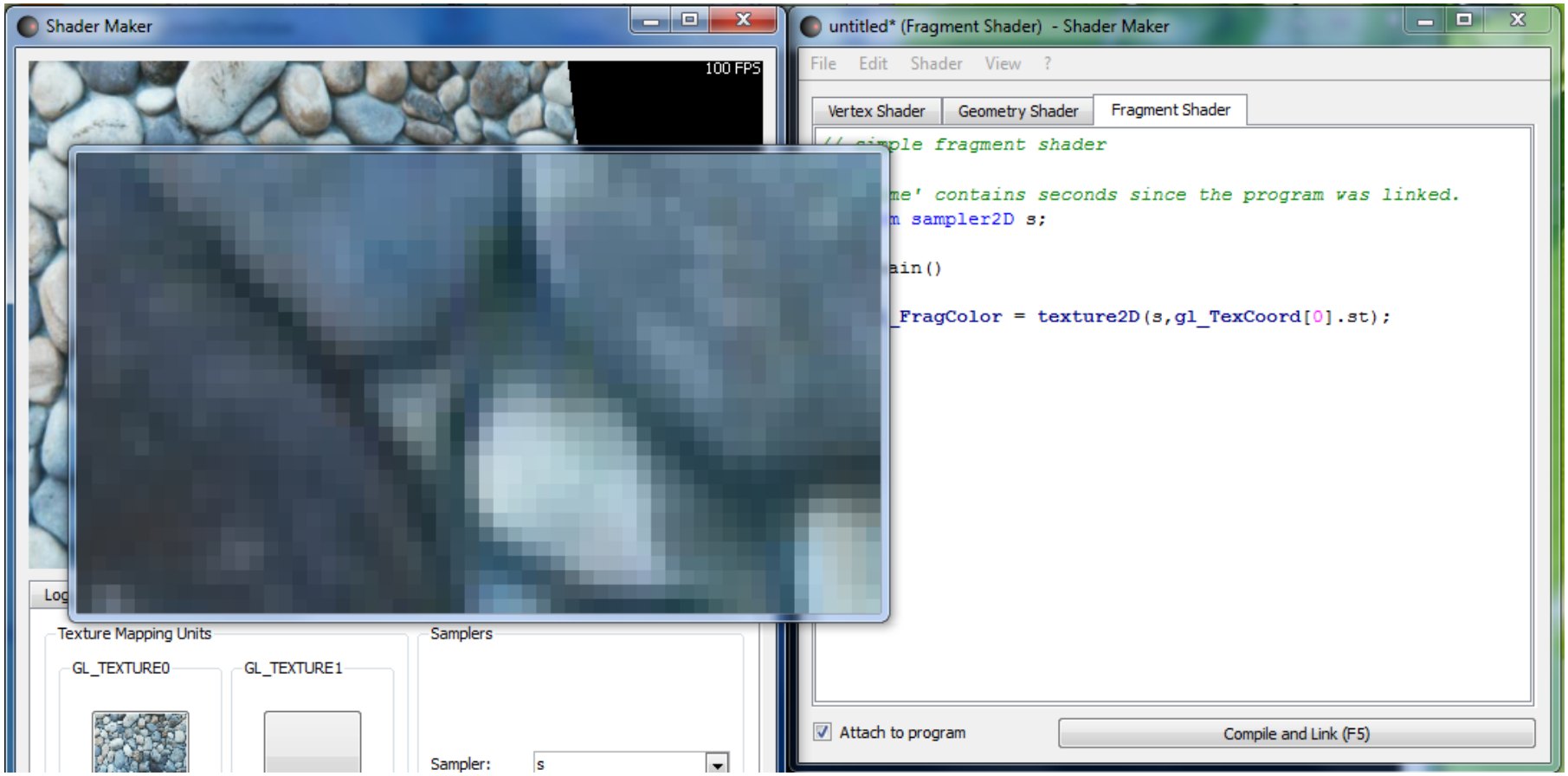
Magnification → la preimatge és < texel

Minification



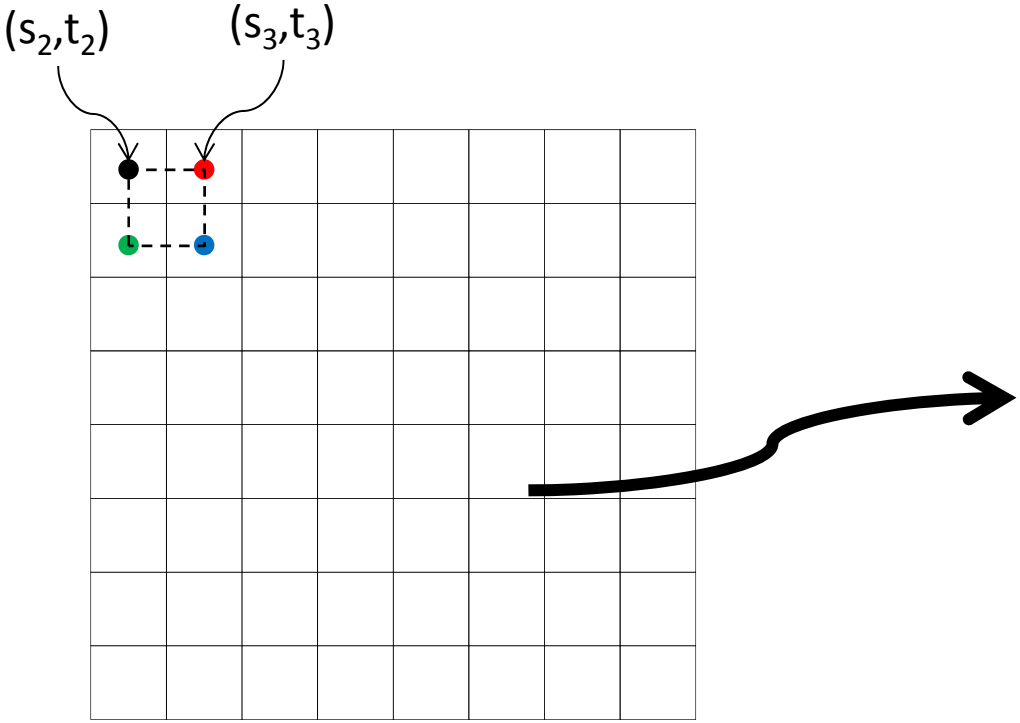
Minification → la preimatge és > texel

Aproximació que fa OpenGL

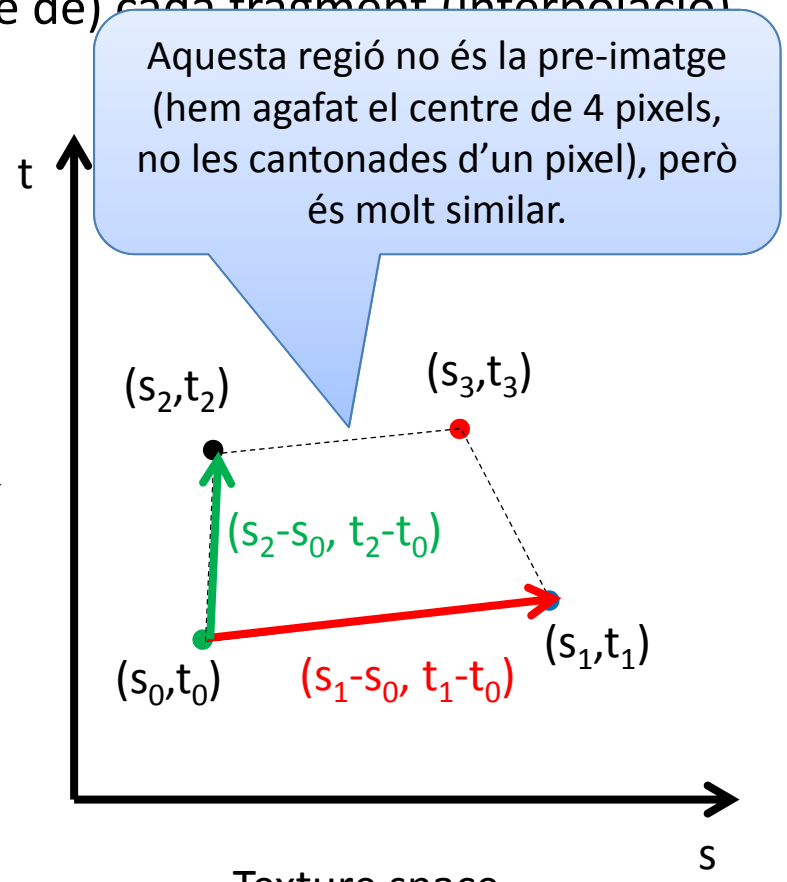


Aproximació que fa OpenGL

- En general no sabem la forma exacta de la pre-imatge…
- Però sí coneixem les coords (s,t) de (el centre de) cada fragment (interpolació)



Window space



Texture space

Aproximació que fa OpenGL

- Cada fragment (x,y) té les seves coordenades (s,t) →
- (s,t) depenen de (x,y) i podem escriure **s**(x,y), **t**(x,y) →
- Té sentit calcular les derivades parcials de **s**(x,y) i **t**(x,y)

$$\partial s/\partial x \approx \mathbf{s}(x+1,y) - \mathbf{s}(x,y)$$
$$\partial s/\partial y \approx \mathbf{s}(x,y+1) - \mathbf{s}(x,y)$$

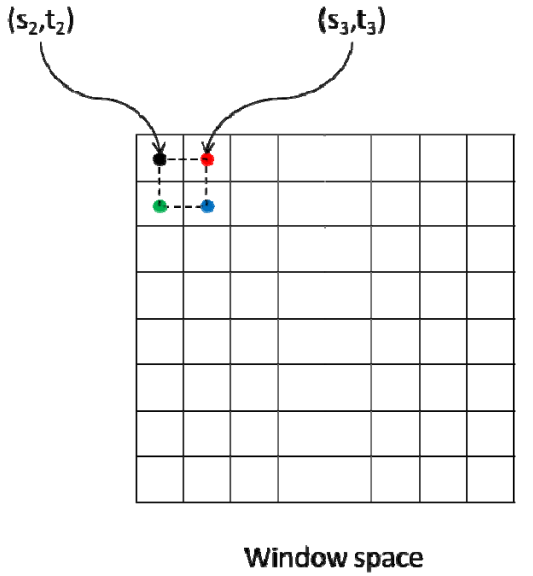
- Anàlogament per t:

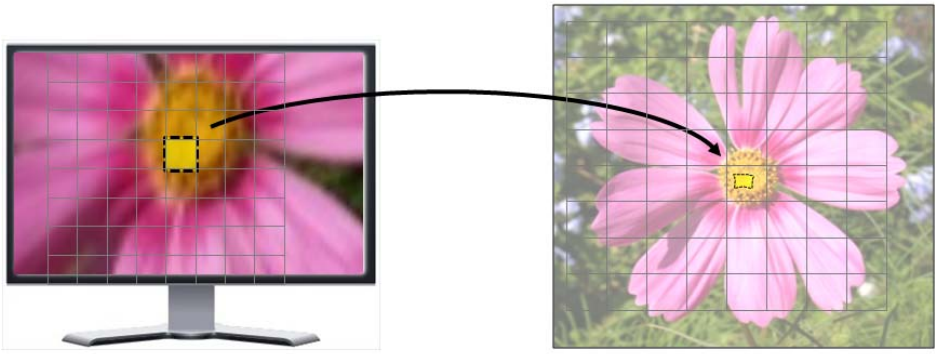
$$\partial t/\partial x \approx \mathbf{t}(x+1,y) - \mathbf{t}(x,y)$$
$$\partial t/\partial y \approx \mathbf{t}(x,y+1) - \mathbf{t}(x,y)$$

- En GLSL es calculen amb dFdx, dFdy:

$$\partial s/\partial x \approx dFdx(texCoord.s)$$
$$\partial s/\partial y \approx dFdy(texCoord.s)$$

(anàlogament per t)

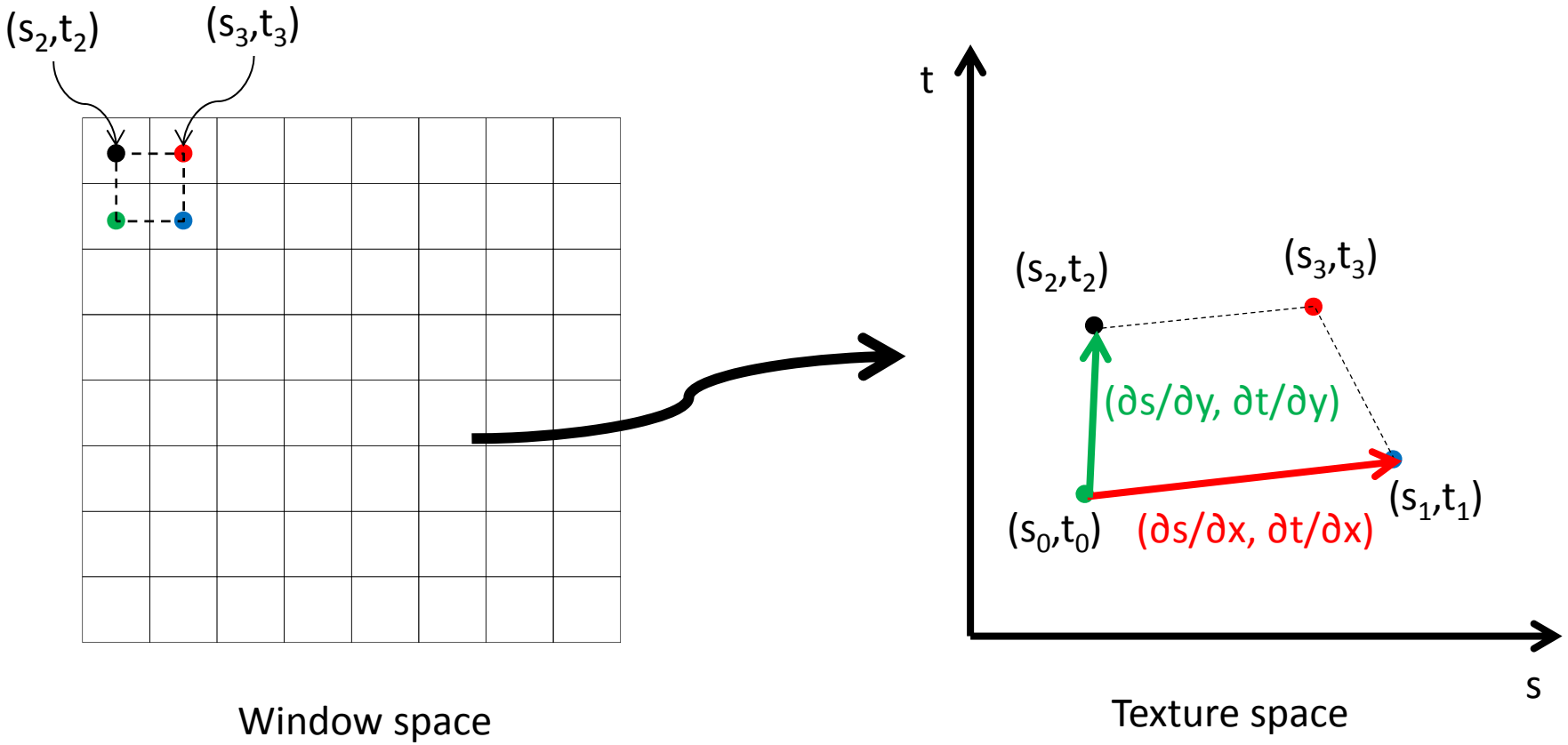




Magnification → la preimatge és < texel

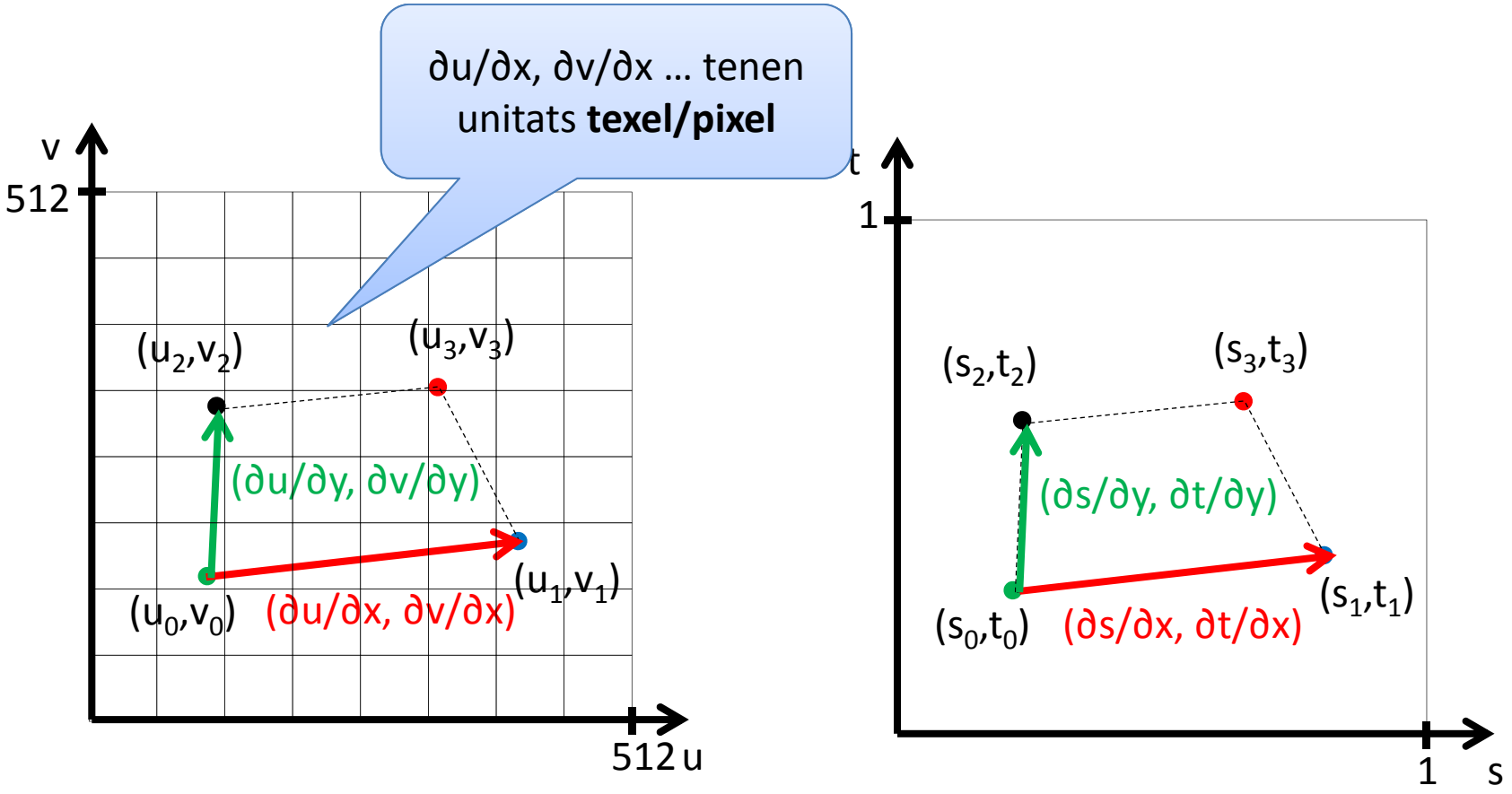
Aproximació que fa OpenGL

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 de WxH pixels

Textura WxH texels

En aquest cas un pixel correspon a un texel:

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

Exemple 2 (magnification x2)

Fragments veïns tenen coordenades (u,v) properes

Textura WxH texels

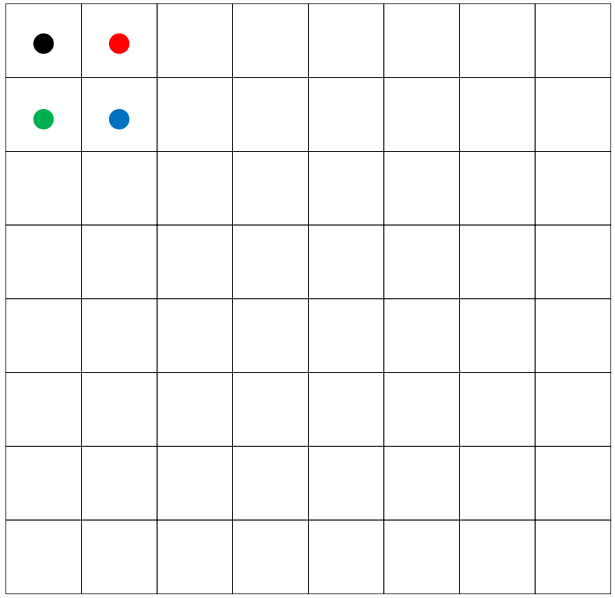
Polígon projectat en 2Wx2H pixels

En aquest cas un pixel correspon a mig texel:

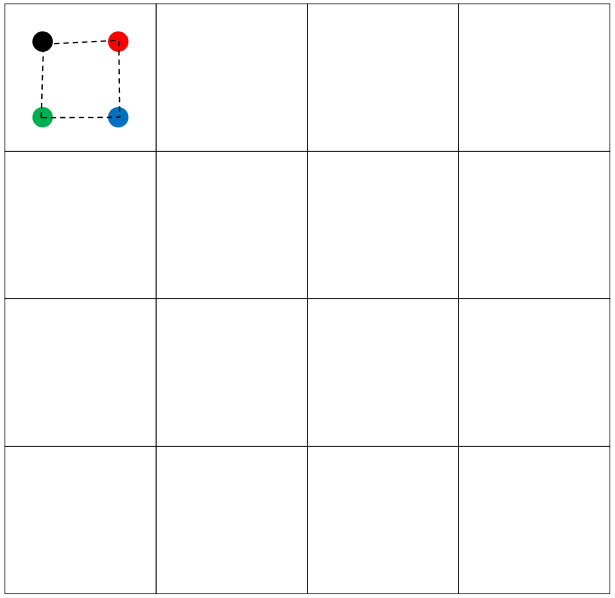
$$\partial u/\partial x = \partial v/\partial y = ½ \quad \partial v/\partial x = \partial u/\partial y = 0$$

Exemple 2 (magnification x2)

$\partial u/\partial x = \partial v/\partial x = \partial u/\partial y = \partial v/\partial y = ½ \rightarrow$
Aquesta regió és més petita que un texel



Polígon texturat 2Wx2H pixels



Textura WxH texels

Exemple 3 (minification x2)

Fragments veïns tenen coordenades (u,v) més separades

Textura WxH texels

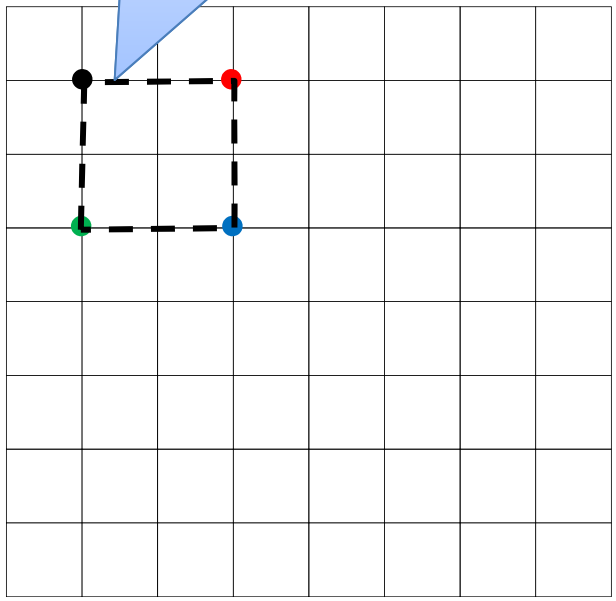
Polígon projectat en ½ W x ½ H pixels

En aquest cas un pixel correspon a dos texels:

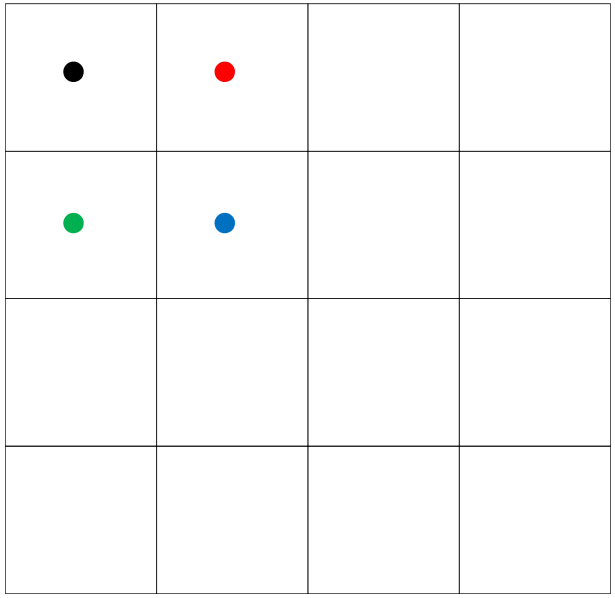
$\partial u/\partial x = \partial v/\partial y = 2 \quad \partial v/\partial x = \partial u/\partial y = 0$

Exemple 3 (minification x2)

∂u/∂x = ∂v/∂x = ∂u/∂y = ∂v/∂y = 2 →
Aquesta regió és més gran que un texel



Textura WxH texels



Polígon texturat ½W x ½H pixels

Exemple 4 (anisotròpic)

Textura WxH texels

En direcció horitzontal → magnification
En direcció vertical → minification

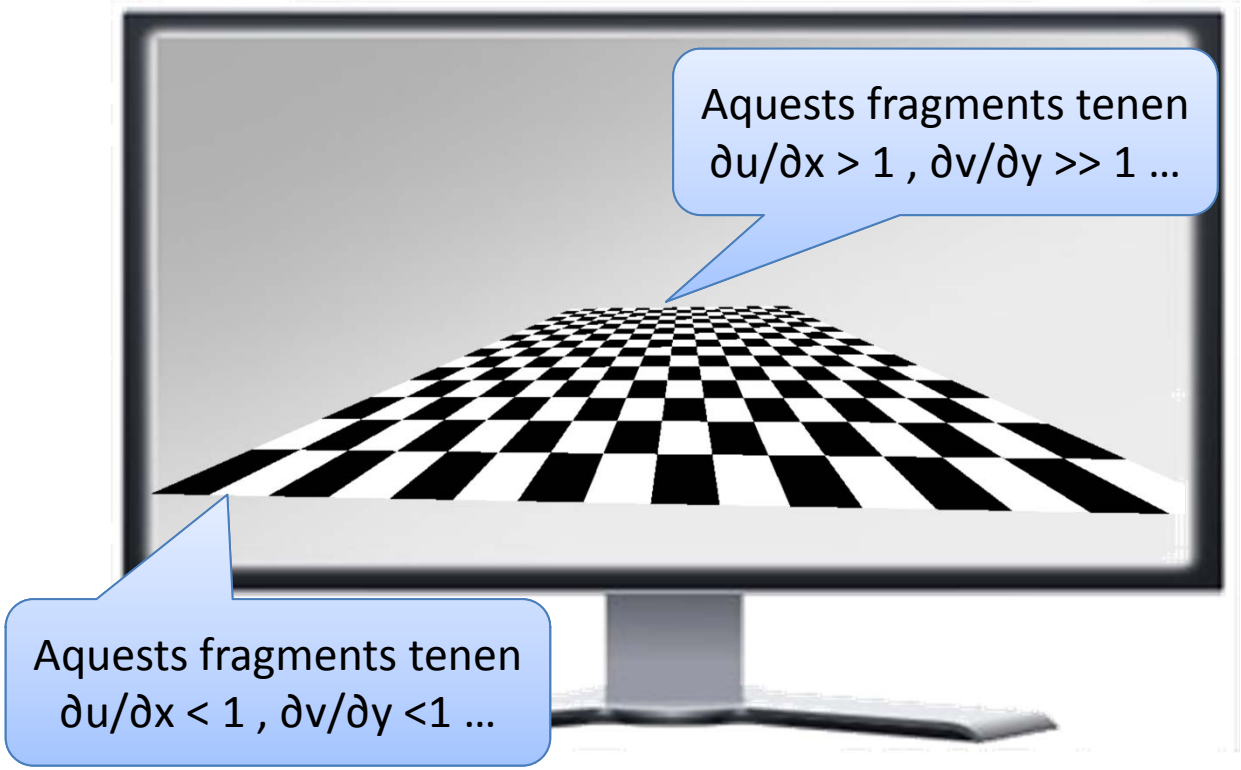
Exemple 4 (anisotròpic)

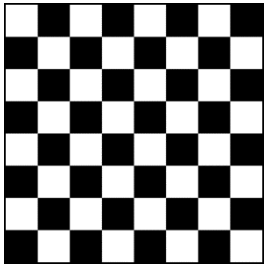
Textura WxH texels

En direcció horitzontal → magnification
En direcció vertical → minification

$\partial u/\partial x = ½ \quad \partial v/\partial x = 0$
$\partial u/\partial y = 0 \quad \partial v/\partial y = 2$

Exemple 5 (la dura realitat)



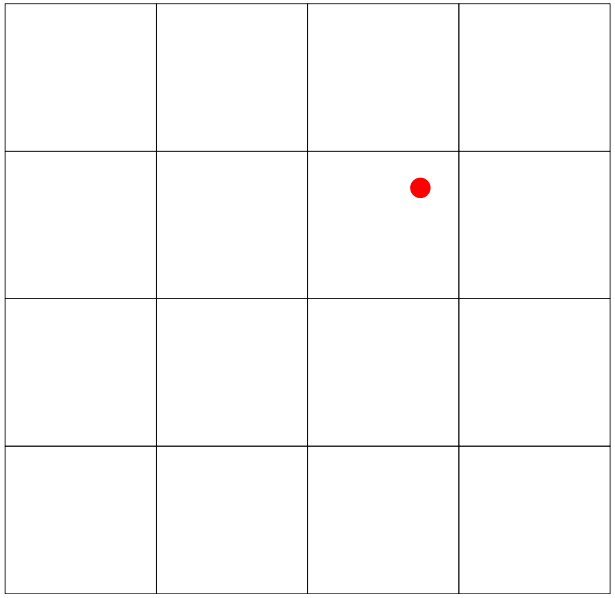


Textura

Texture filters

Determinen:

- Com es calcula el color al punt (s,t)
- Com s'avalua texture(sampler, texCoord)



Textura WxH texels

Magnification filters

- GL_NEAREST: nearest neighbor sampling

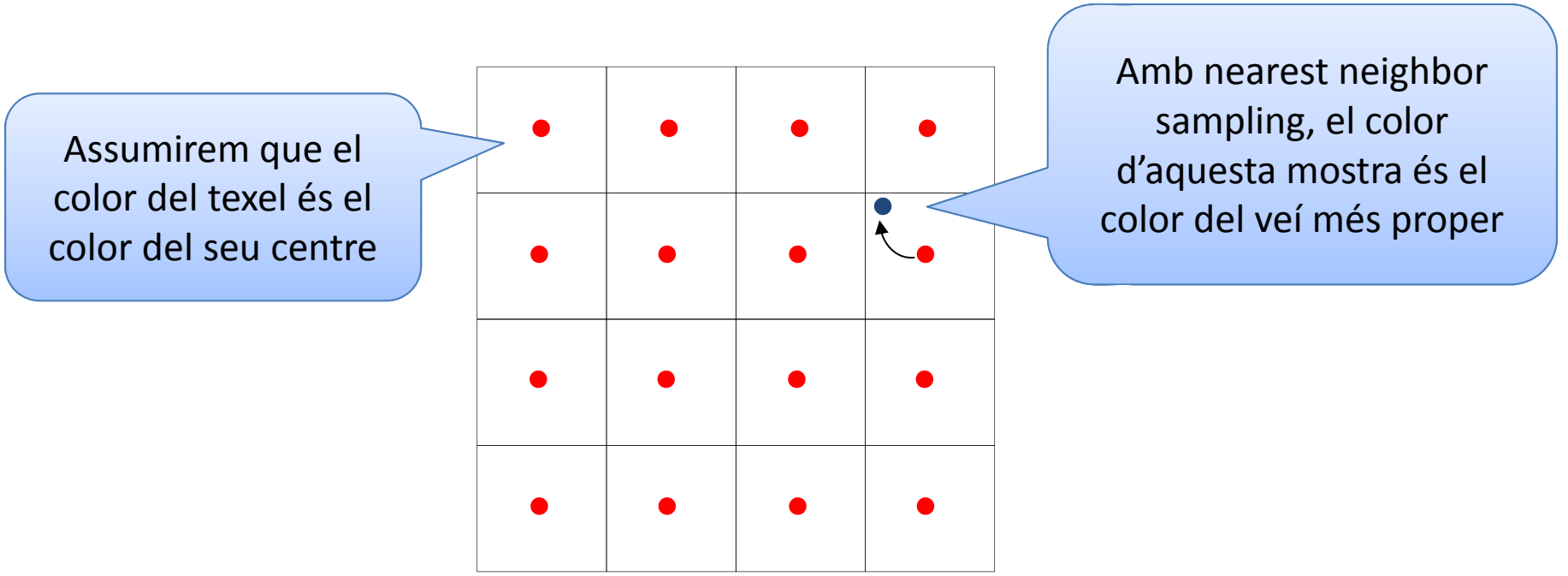
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_**MAG**_FILTER, **GL_NEAREST**);

- GL_LINEAR: interpolació bilineal

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_**MAG**_FILTER, **GL_LINEAR**);

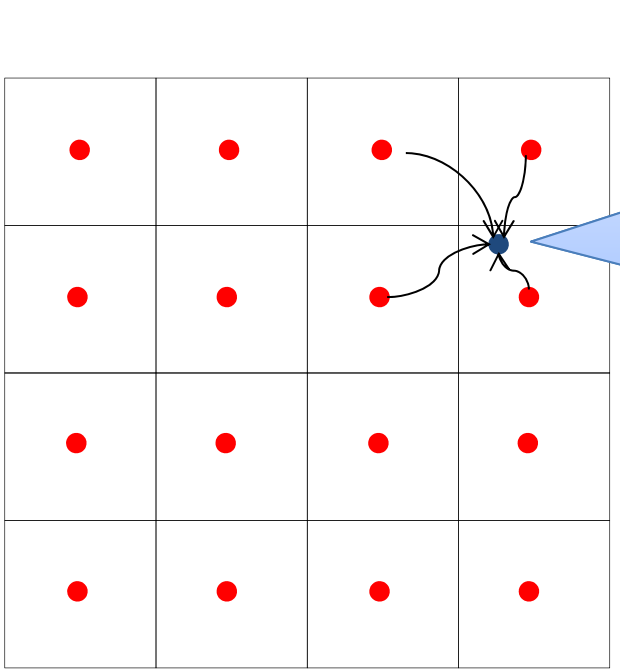
Magnification filters

GL_NEAREST: nearest neighbor sampling



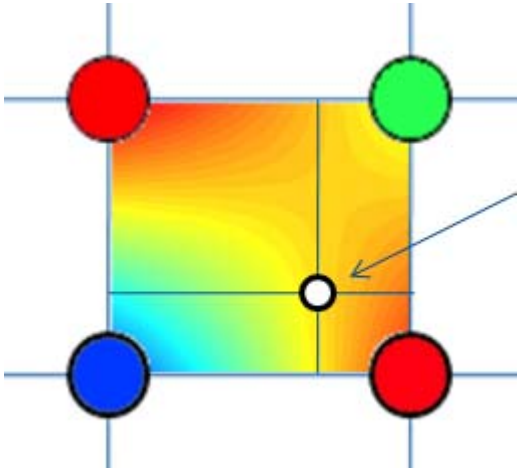
Magnification filters

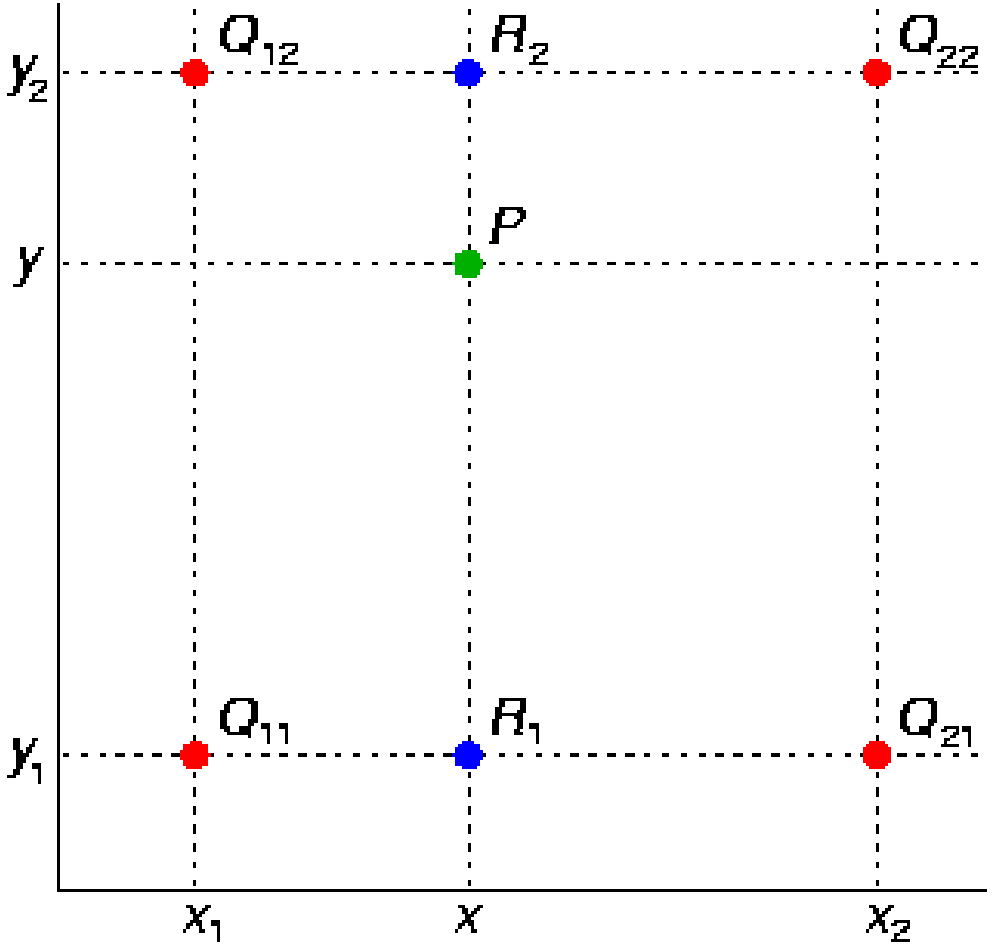
GL_LINEAR: bilinear interpolation



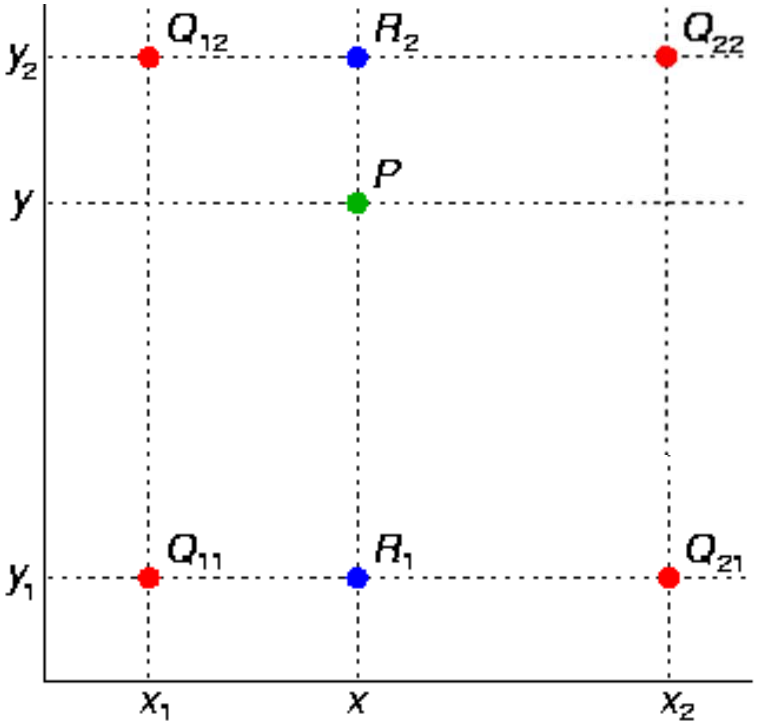
Amb interpolació bilineal, el color d'aquesta mostra és una mitjana ponderada dels colors dels quatre veïns més propers

Bilinear interpolation





Bilinear interpolation



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

Magnification filters

[Demo shadermaker]

Minification filters

- GL_NEAREST: nearest neighbor sampling

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```

- GL_LINEAR: interpolació bilineal

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

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!

Si aquest quad es projecta en 2x2 pixels, cada fragment correspon a una quarta part de la imatge!

Minification x16

Minification x8

Minification x4

Minification x2

Textura

Mipmapping

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

256x256 (LOD 0)

128x128 (LOD 1)

64x64 (LOD 2)

32x32 (LOD 3)

16x16 (LOD 4)

1x1 (LOD 8)

Un texel del LOD 3 “representa” 8x8 texels de la imatge original

Un texel del LOD 2 “representa” 4x4 texels de la imatge original

Un texel del LOD 1 “representa” 2x2 texels de la imatge original

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)

A l'hora de calcular el color d'una mostra, agafarem el LOD (o els dos LODs) més adient(s)

Mipmapping

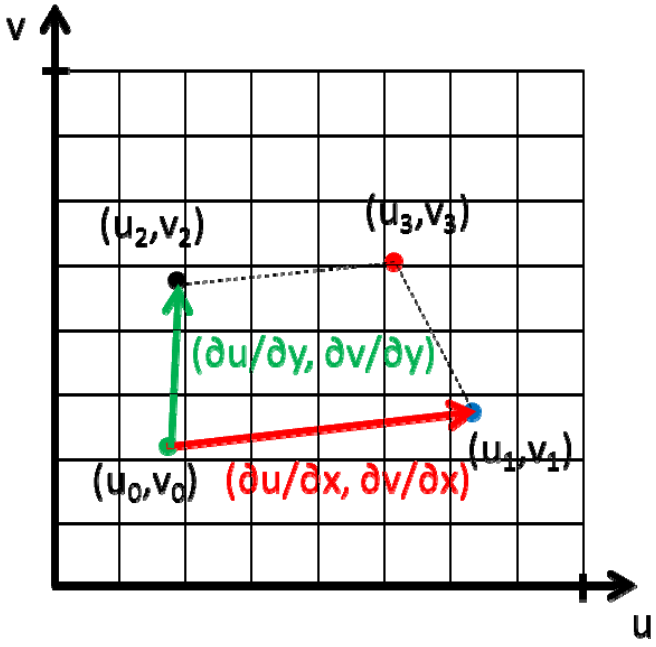
En alguns casos el LOD més adient λ és fàcil de calcular

	Minification	∂u/∂x, ∂v/∂y	LOD
	1x	1	0
	2x	2	1
	4x	4	2
	8x	8	3
	2^{λ} x	2^{λ}	λ

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

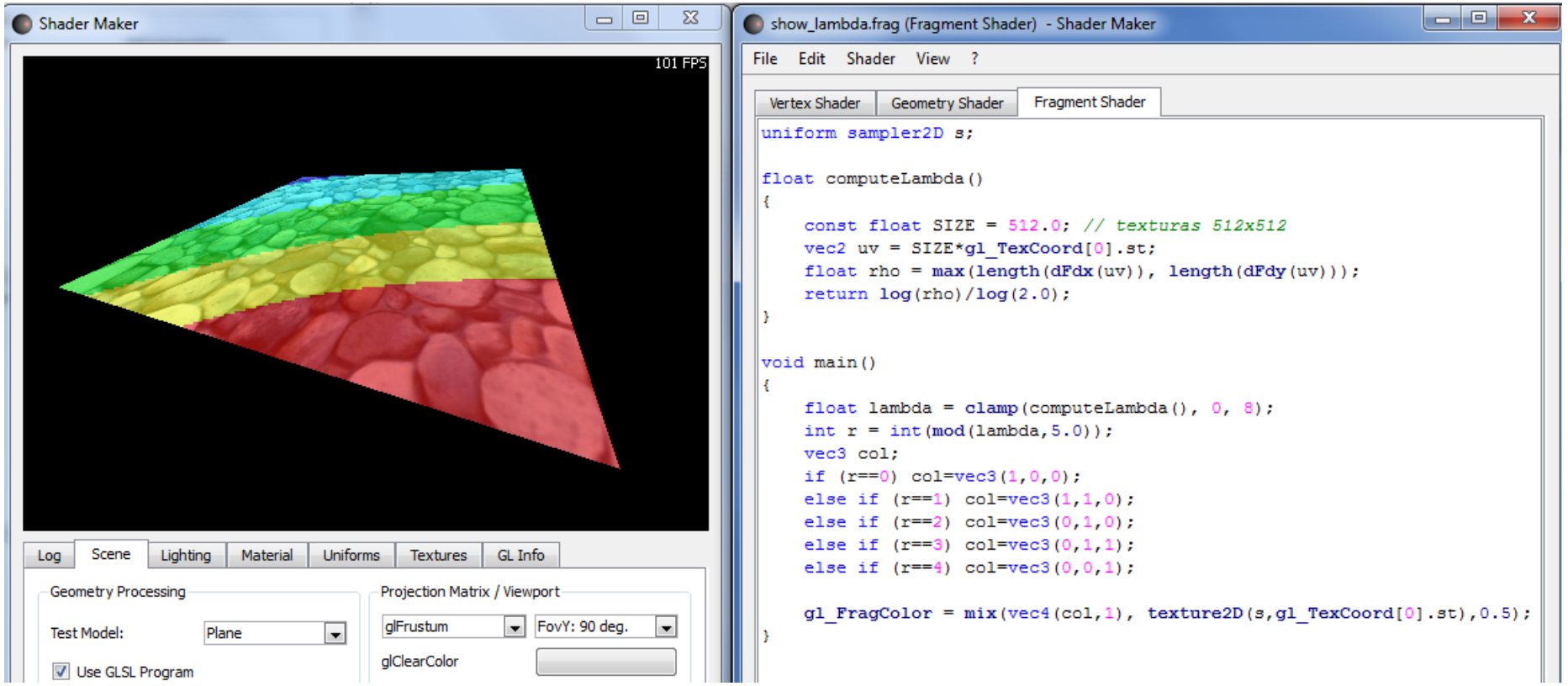
Mipmapping

- En general, però, ∂u/∂x != ∂v/∂x != ∂u/∂y != ∂v/∂y
- En aquests casos, es calcula un valor rho=f(∂u/∂x, ∂v/∂x, ∂u/∂y, ∂v/∂y) i després $\lambda=\log_2(rho)$



Mipmapping

- Demo: show_lambda.frag



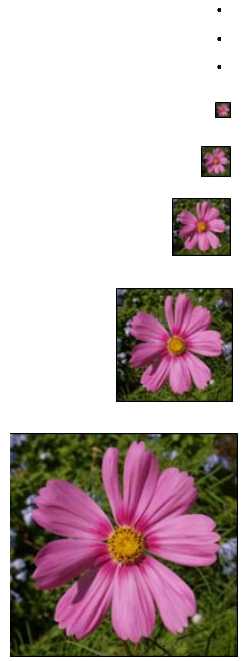
Minification filters (2)

Without mipmapping

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

With mipmapping

- GL_NEAREST_MIPMAP_NEAREST // Nearest neighbor sampling on LOD int(λ)
- GL_LINEAR_MIPMAP_NEAREST // Bilinear sampling on LOD int(λ)
- GL_NEAREST_MIPMAP_LINEAR
  // c_0 = nearest neighbor on LOD int(λ)
  // c_1 = nearest neighbor on LOD int(λ+1)
  // mix(c_0, c_1, fract(λ))
- GL_LINEAR_MIPMAP_LINEAR
  // c_0 = bilinear sampling on LOD int(λ)
  // c_1 = bilinear sampling on LOD int(λ+1)
  // mix(c_0, c_1, fract(λ))



GL_<samplingWithinTheImage>_MIPMAP_<oneOrTwoMipmaps>

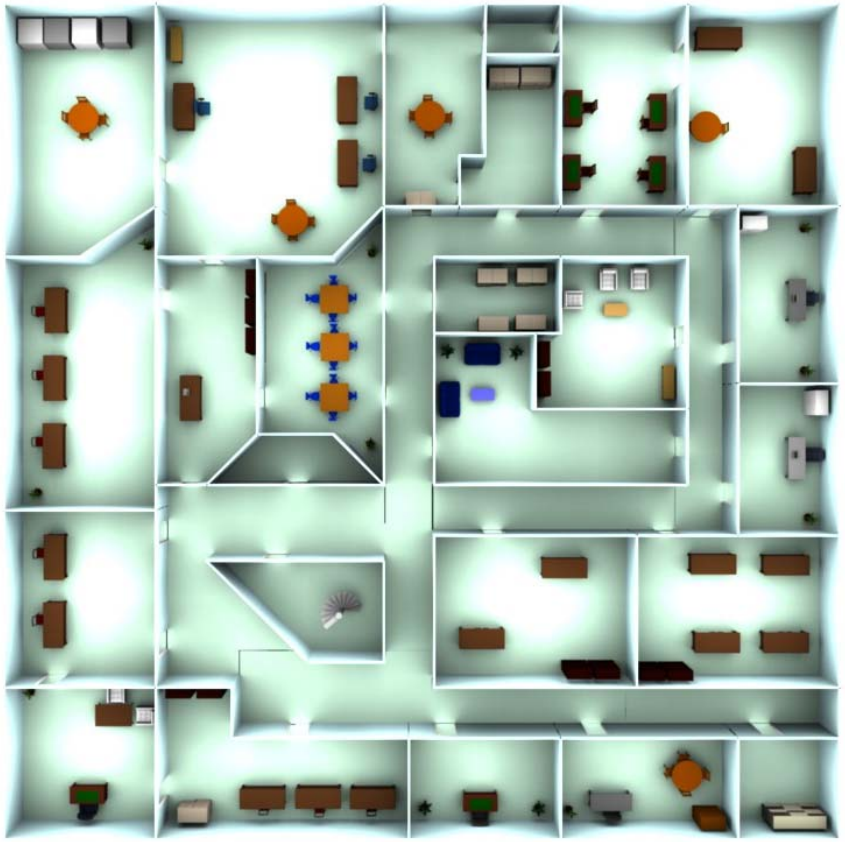
Minification filters (2)

[Demo mipmapping]

Texture combiners

COMBINACIÓ

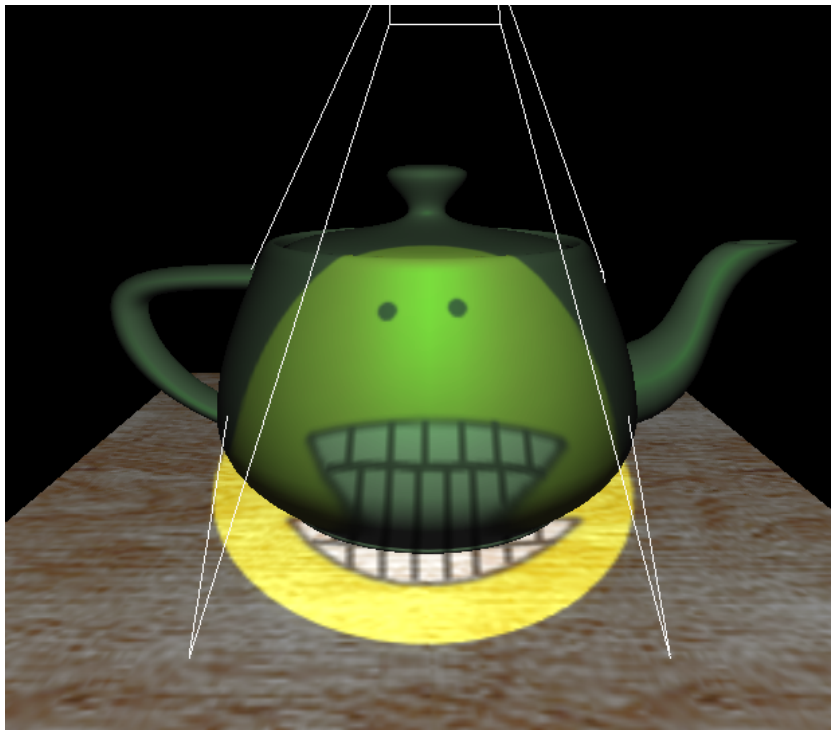
Light map/Diffuse color map

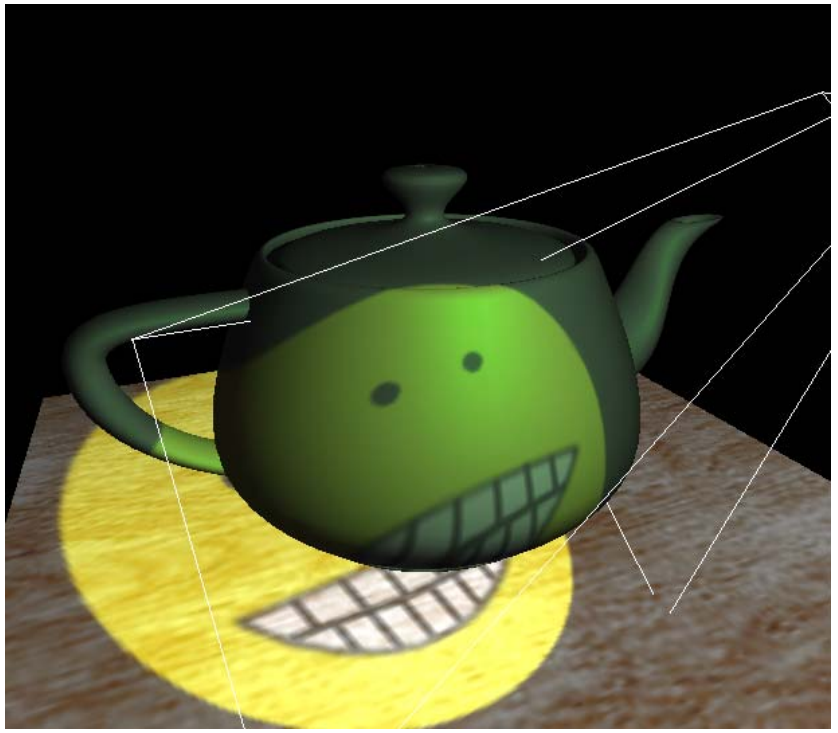


[Texture combiner demo]

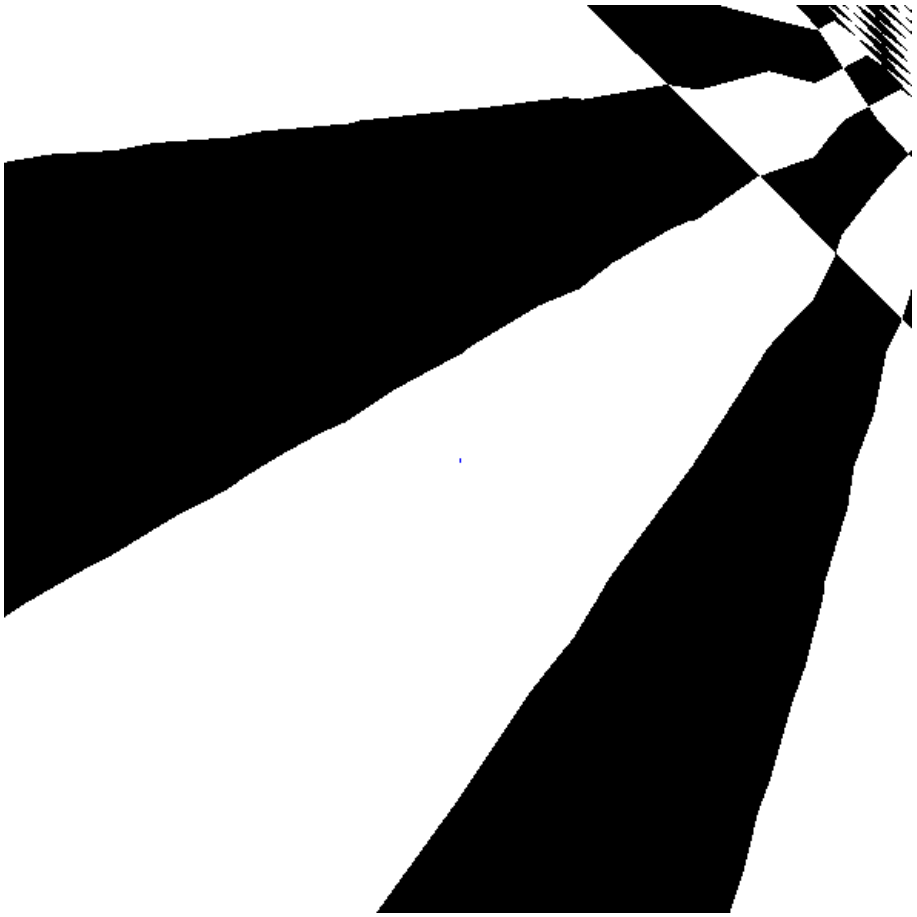
PROJECTIVE TEXTURE MAPPING

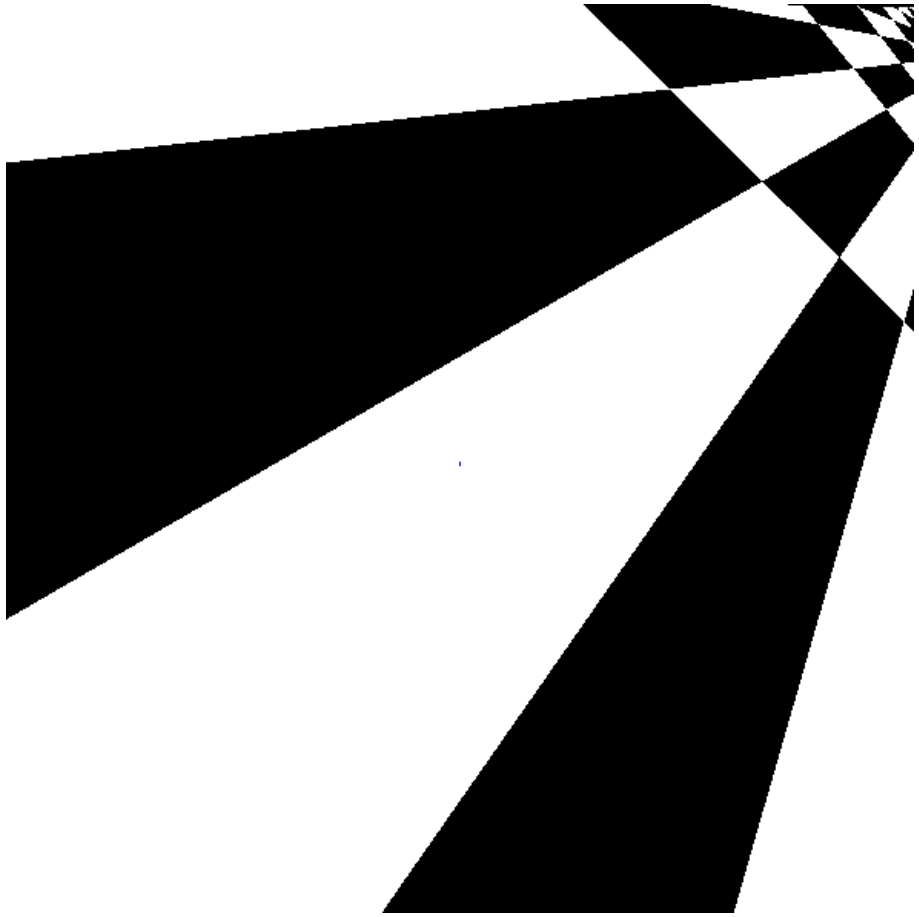
Projective texture mapping





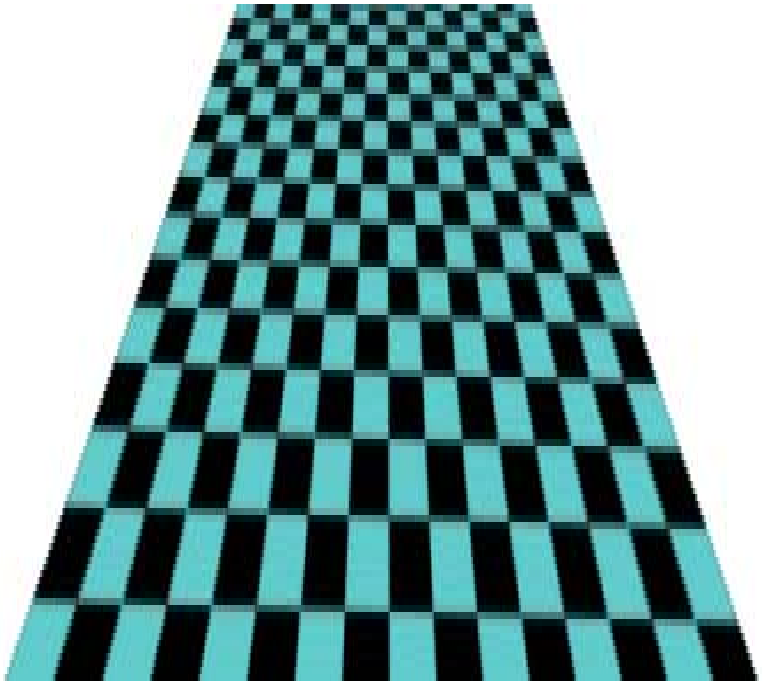
Projective texture mapping

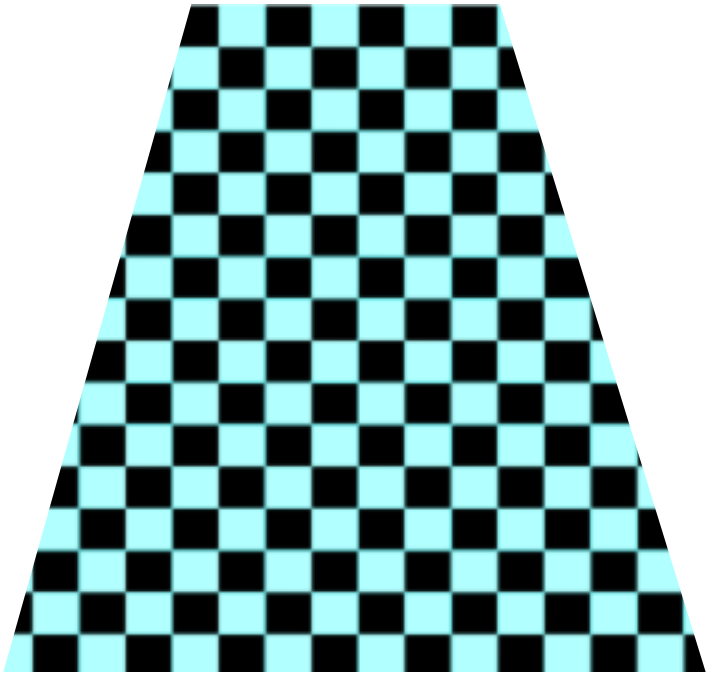




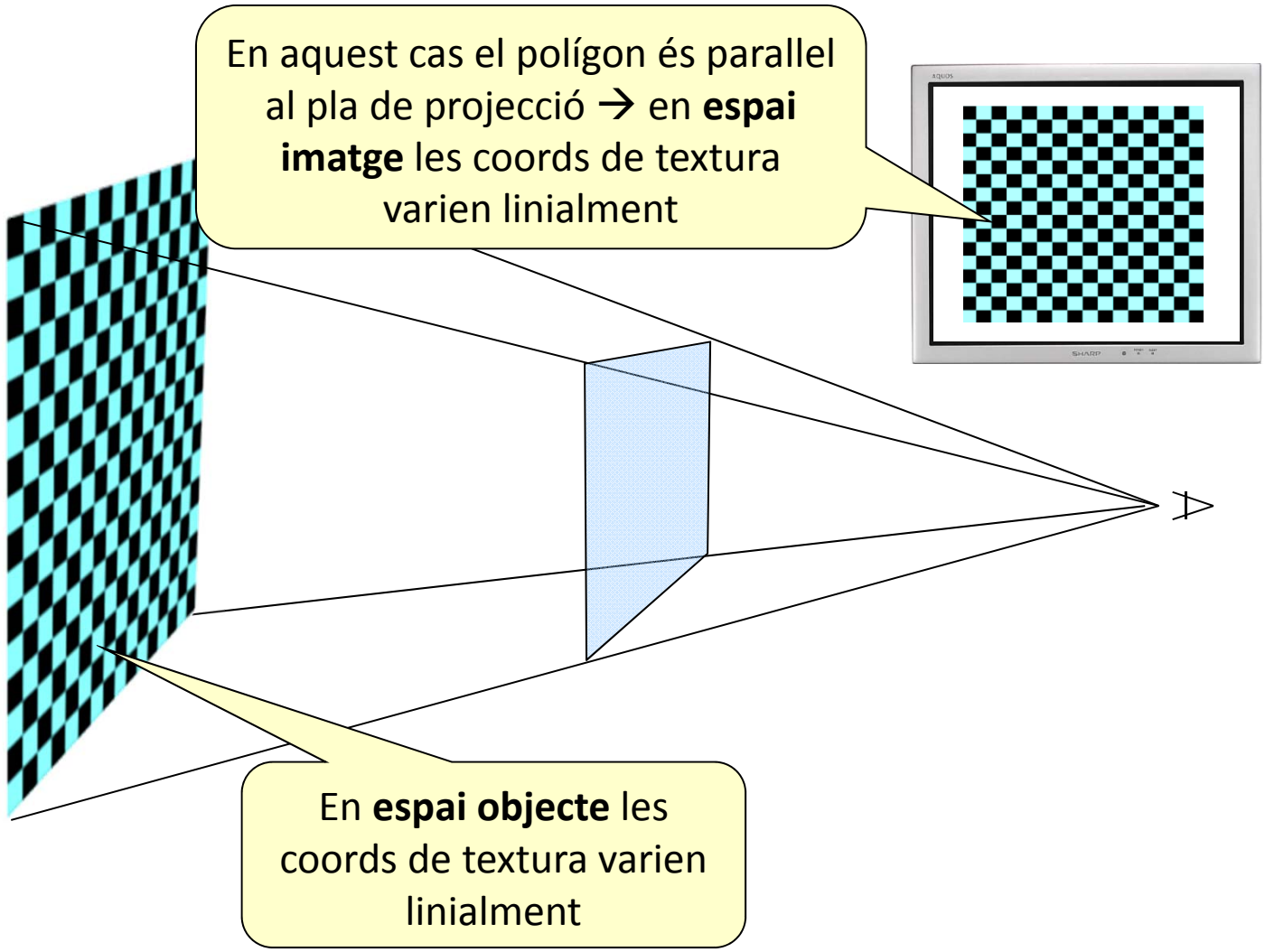
Perspective deformation

Perspective deformation

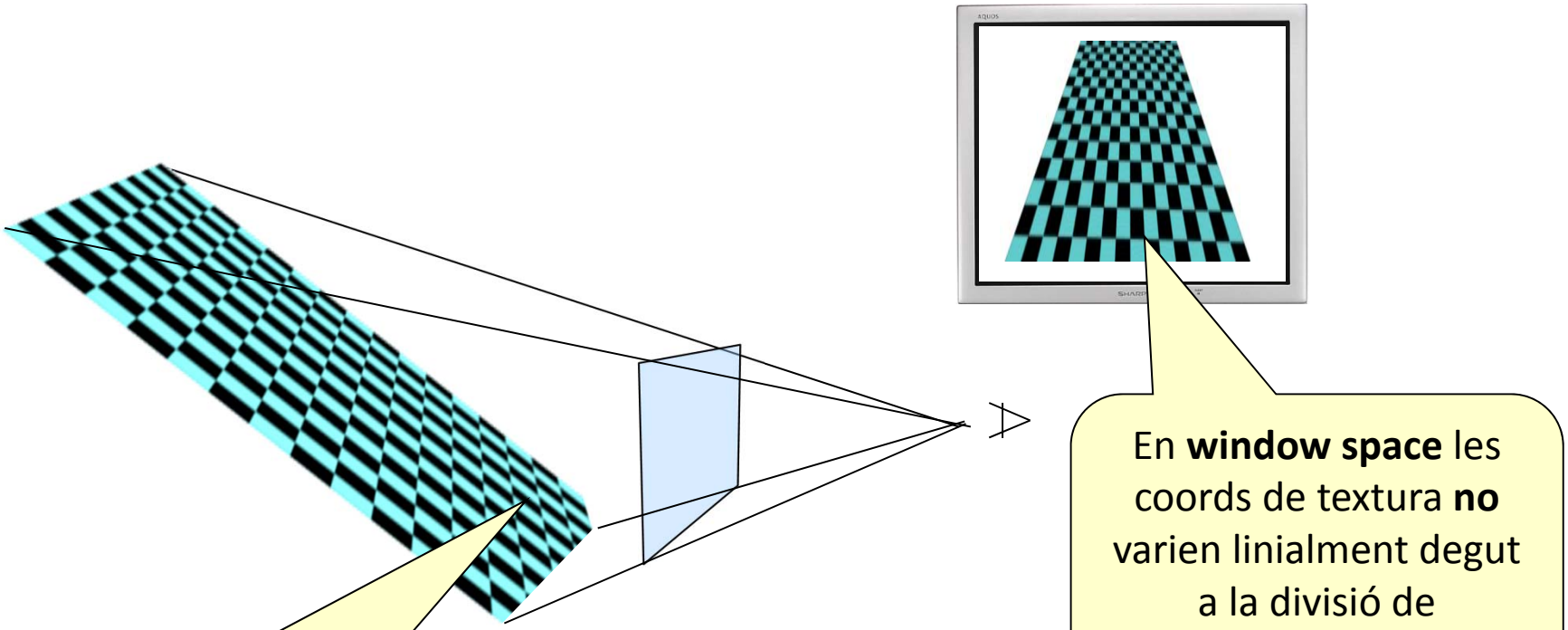




Perspective-correct interpolation



Perspective-correct interpolation

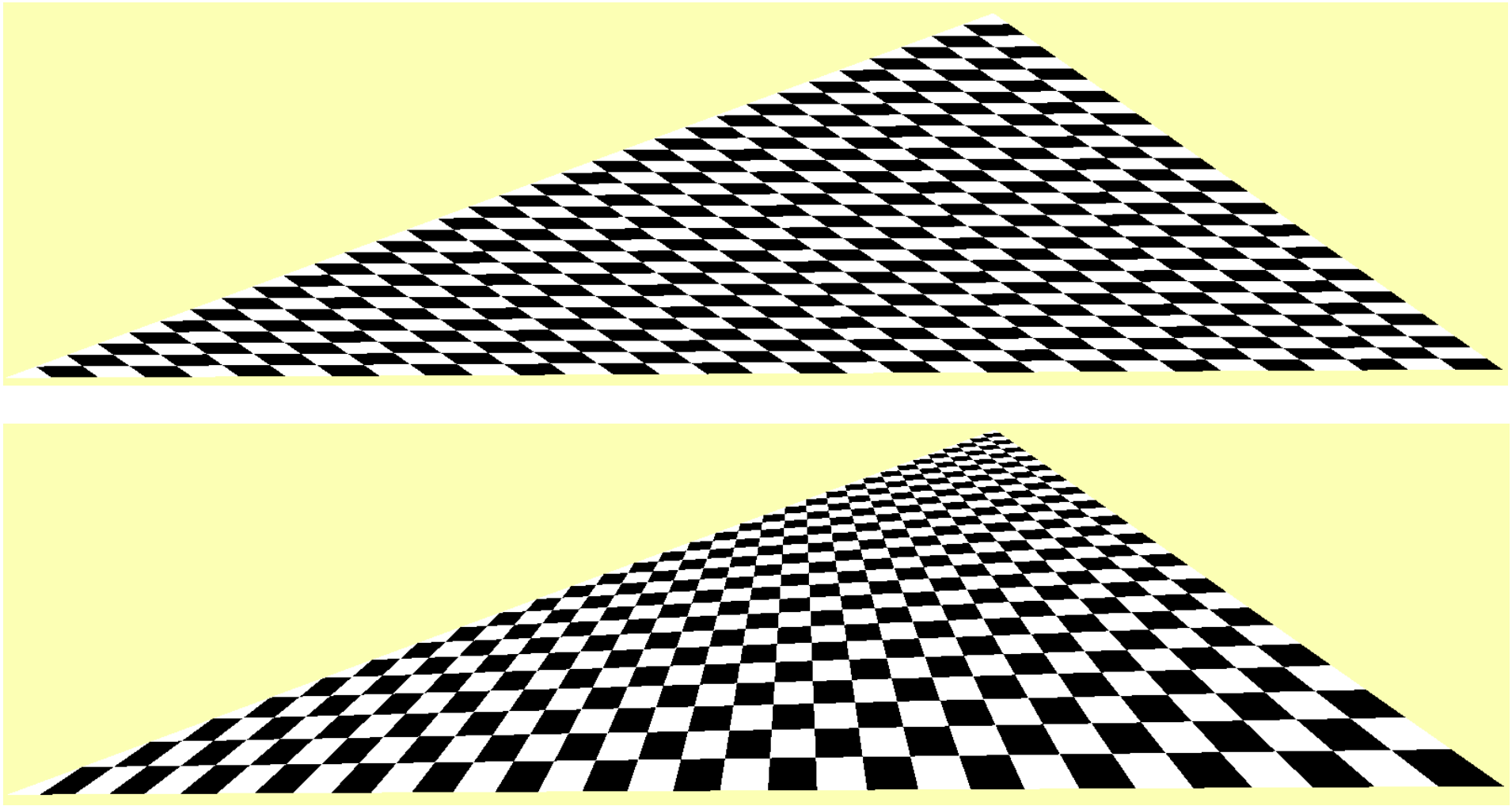


En **object space** les coords de textura varien linialment (totes les cel·les 4cm x 4cm)

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

Què passa si interpolem (linealment) en **window space**?

Perspective-correct interpolation



[Demo: pers.vert, pers.frag]

Perspective-correct interpolation

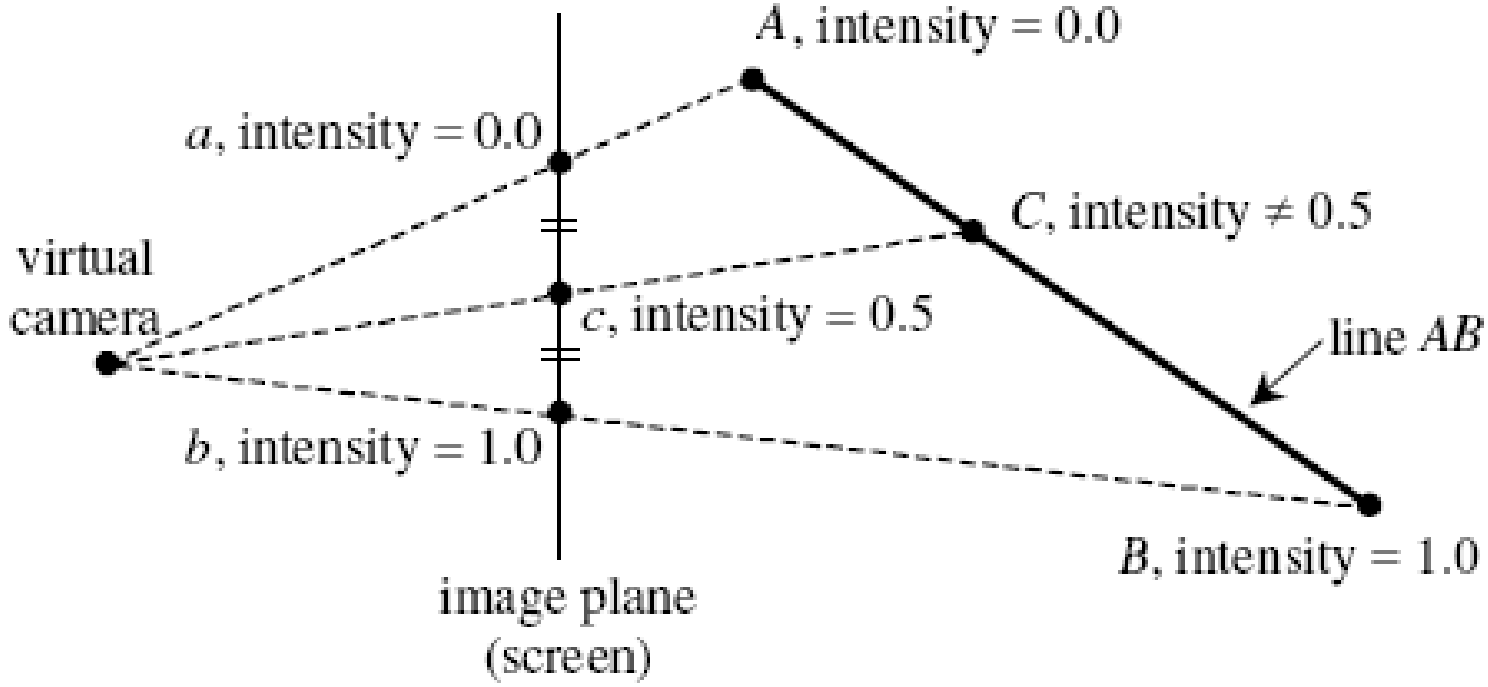


***Figure 1**: Straightforward linear interpolation of attribute values in the screen space (or in the image plane) does not always produce perspective-correct results.*

Perspective-correct interpolation

Solucions:

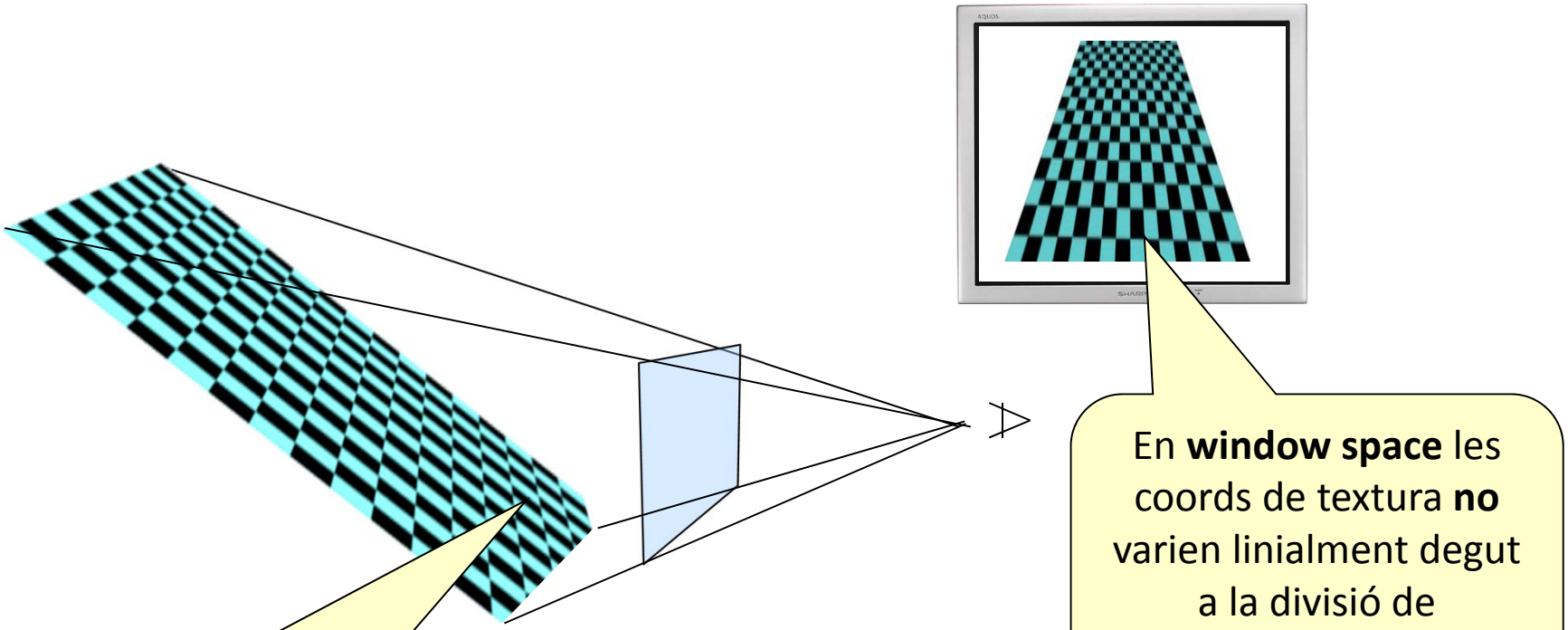
a) Interpolem (s,t) en **object space** (també valdria en **world space** o **eye space**, perquè són abans de la div. de perspectiva)

b) Interpolem (sw_d, tw_d, w_d) en **window space**, obtenint un texel **(s,t,q)**; accedirem a la textura amb **(s/q, t/q)**

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

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

Perspective-correct interpolation

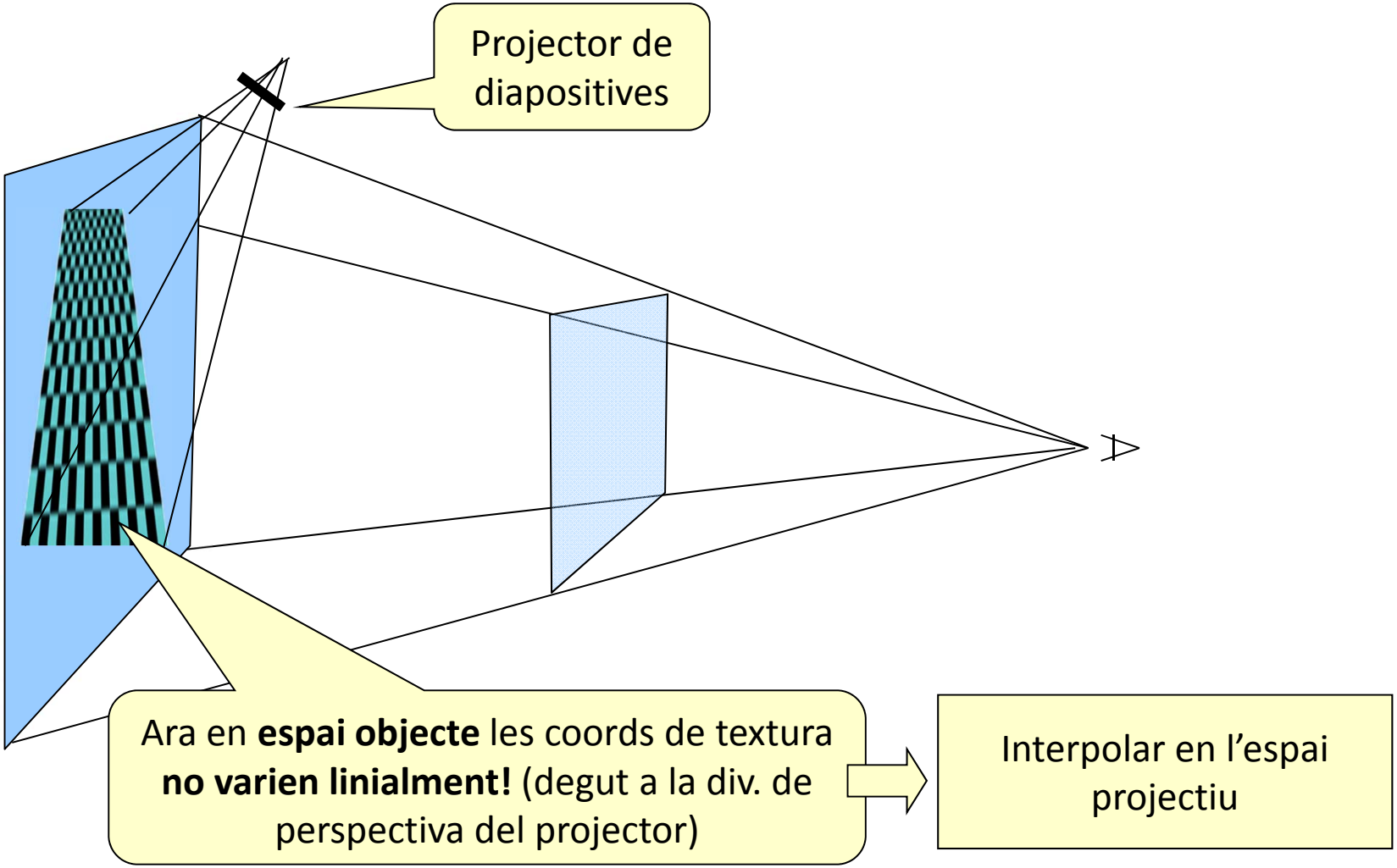


En **object space** les coords de textura varien linialment (totes les cel·les 4cm x 4cm)

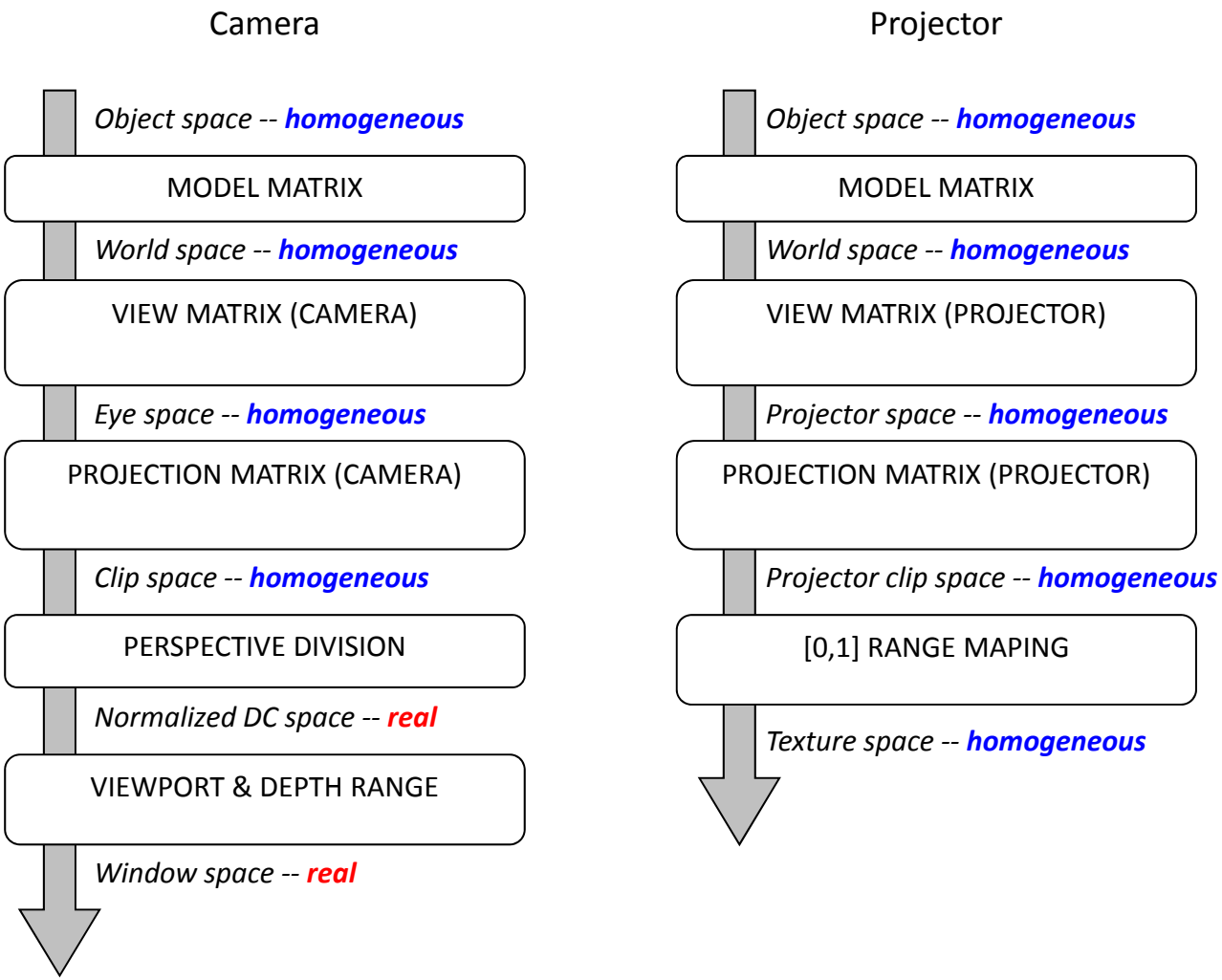
En **window space** les coords de textura **no** varien linialment degut a la divisió de perspectiva

Cal interpolar (sw_d, tw_d, w_d)

Projective-space interpolation



Projective texture mapping



Projective texture mapping (1/2)

```
// Pas 1: Modificar la TEXTURE MATRIX
glMatrixMode(GL_TEXTURE);
glLoadIdentity();
glTranslated(0.5, 0.5,0.5); // T (translació)
glScaled(0.5, 0.5, 0.5);       // S (escalat)
gluPerspective(…);             // Pp (projection matrix del projector)
gluLookAt(…);                  // Vp (view matrix del projector, no de càmera!
glMultMatrixf(Model);          // M (tranformació de modelat)
glMatrixMode(GL_MODELVIEW);
../..
// Pas 2:  Dibuixar amb coords de textura = coords del vèrtex
…
glTexCoord3f(x,y,z);
glVertex3f(x,y,z);
…
```

Projective texture mapping (2/2)

```
// Pas 2: Opció (a) – Dibuixar amb coords de textura = coords del vèrtex
glTexCoord3f(x,y,z);
glVertex3f(x,y,z);
                …
// Pas 2: Opció (b) – Utilitzar glTexGen per aconseguir el mateix efecte
glTexGenf(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenf(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenf(GL_R, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenf(GL_Q, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
float s[4] ={1.0f, 0.0f, 0.0f, 0.0f);
float t[4] ={0.0f, 1.0f, 0.0f, 0.0f);
float r[4] ={0.0f, 0.0f, 1.0f, 0.0f);
float q[4] ={0.0f, 0.0f, 0.0f, 1.0f);
glTexGenf(GL_S, GL_OBJECT_PLANE, s);
glTexGenf(GL_T, GL_OBJECT_PLANE, t);
glTexGenf(GL_R, GL_OBJECT_PLANE, r);
glTexGenf(GL_Q, GL_OBJECT_PLANE, q);
dibuixarEscena()
```

Projective texture mapping

- També es pot utilitzar glTexGen() amb GL_EYE_LINEAR, amb alguns avantatges addicionals.
- Consulteu:

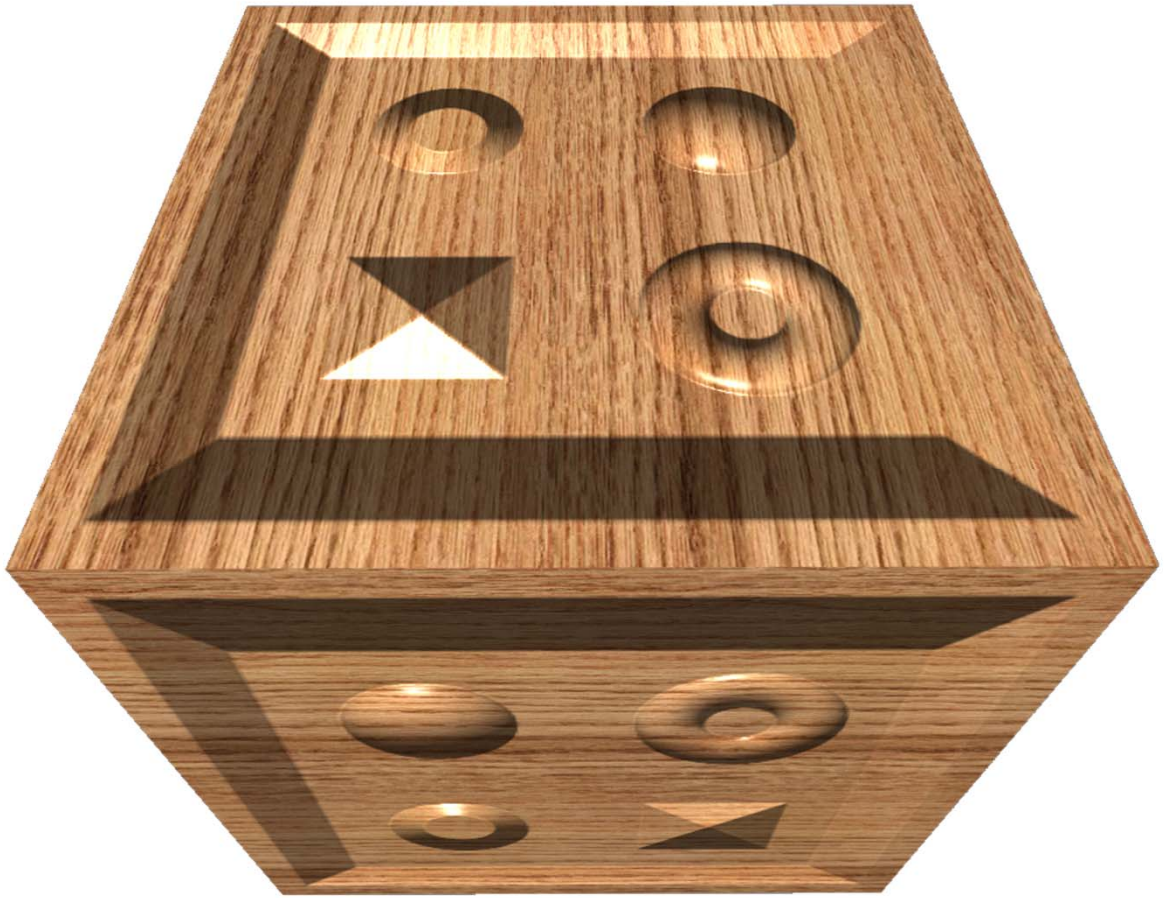
Cass Everitt, Projective Texture Mapping. Nvidia,

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

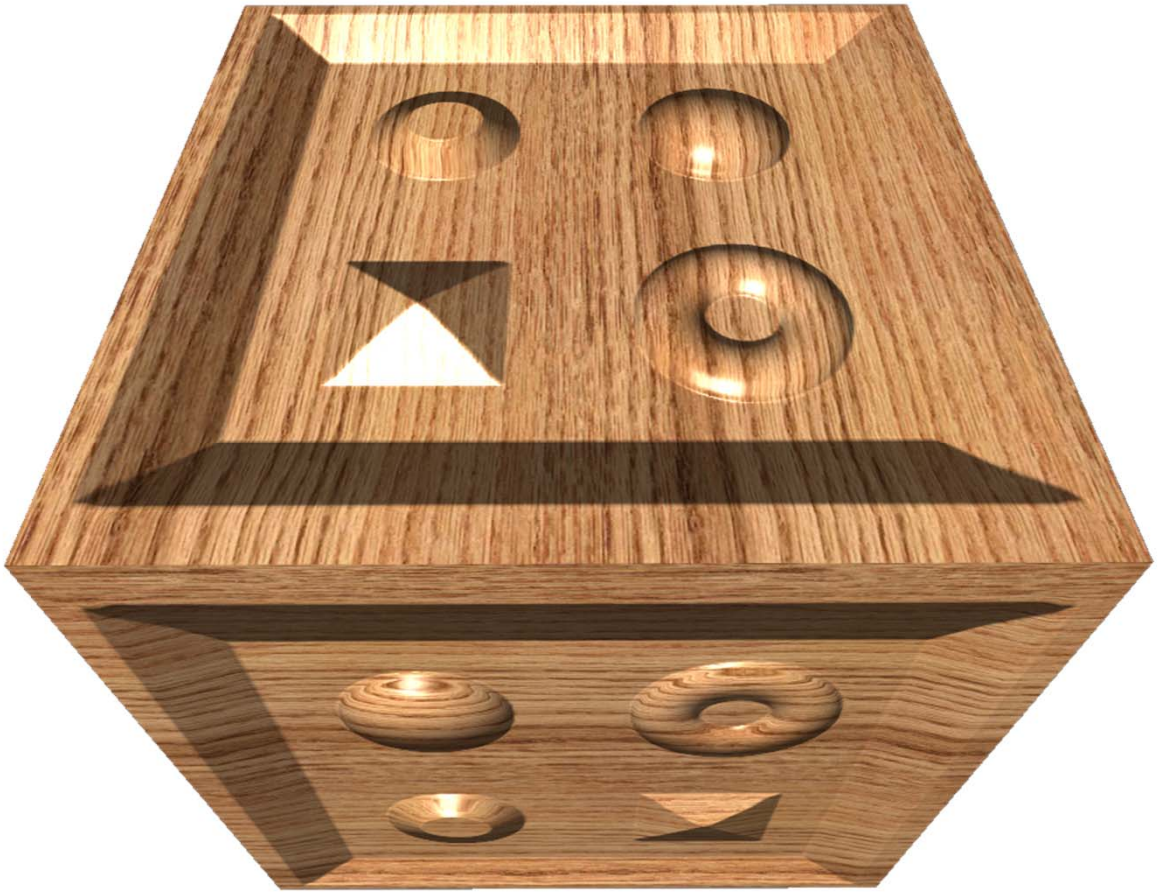
Color, bump, parallax, relief i displacement mapping

APLICACIONS

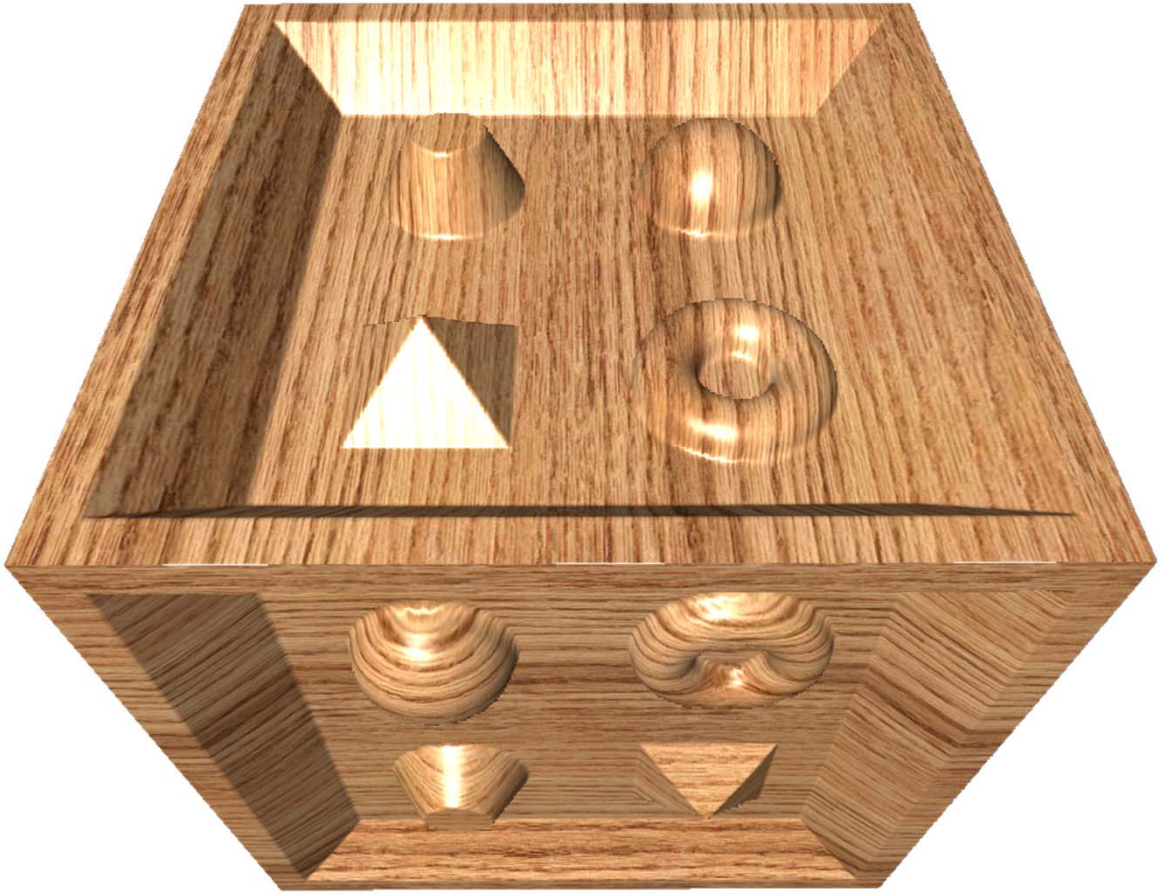
Bump mapping/Normal mapping



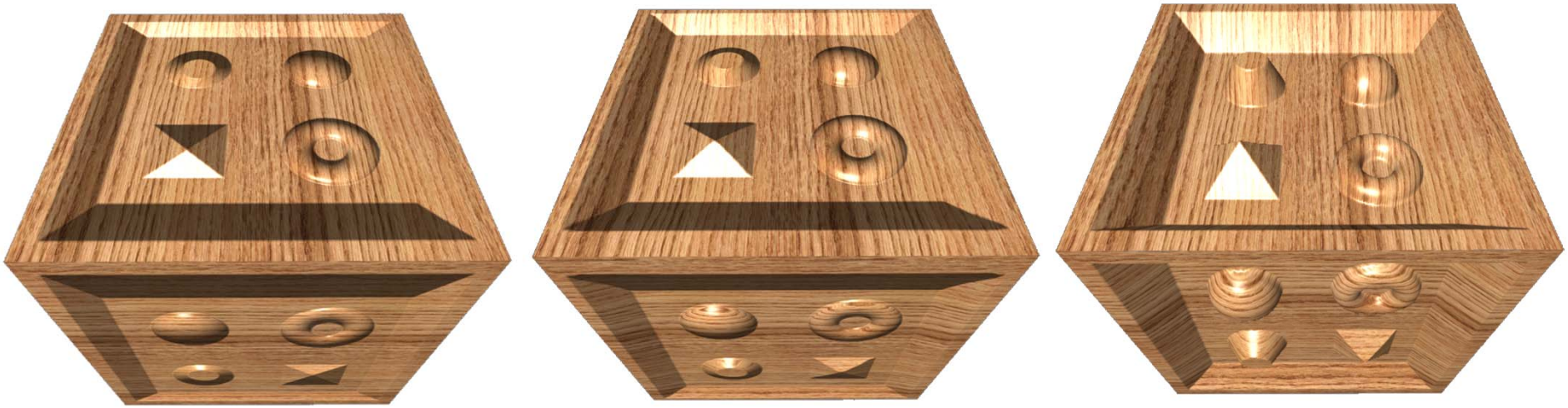
Parallax mapping



Relief mapping



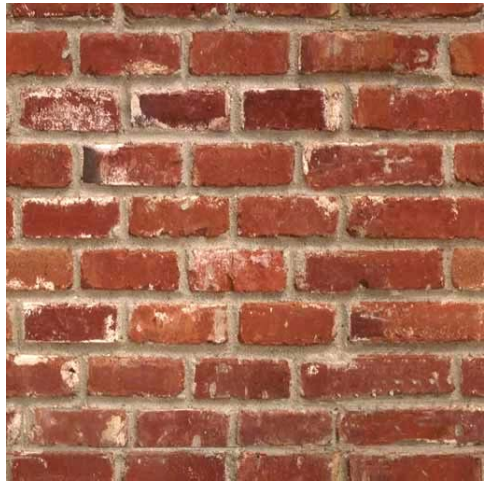
Comparació

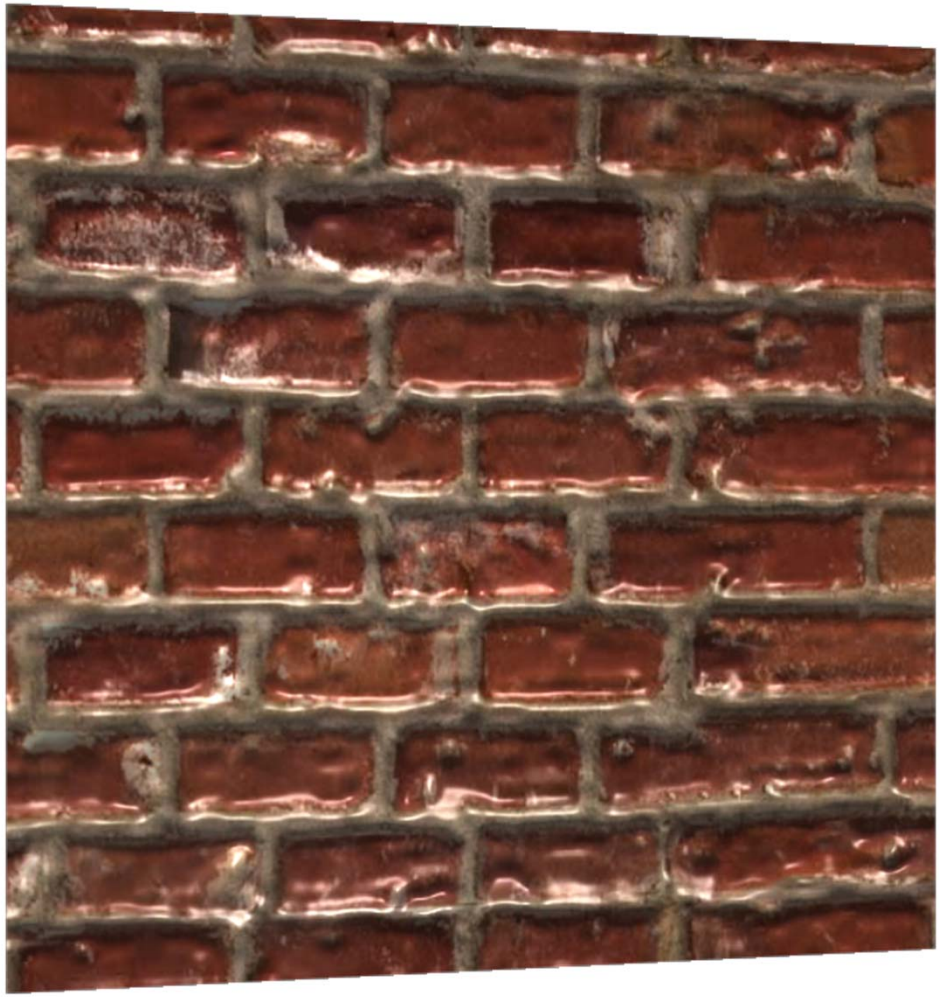


Mark Kilgard. A Practical and Robust Bump-mapping Technique for Today's GPUs. GDC 2000

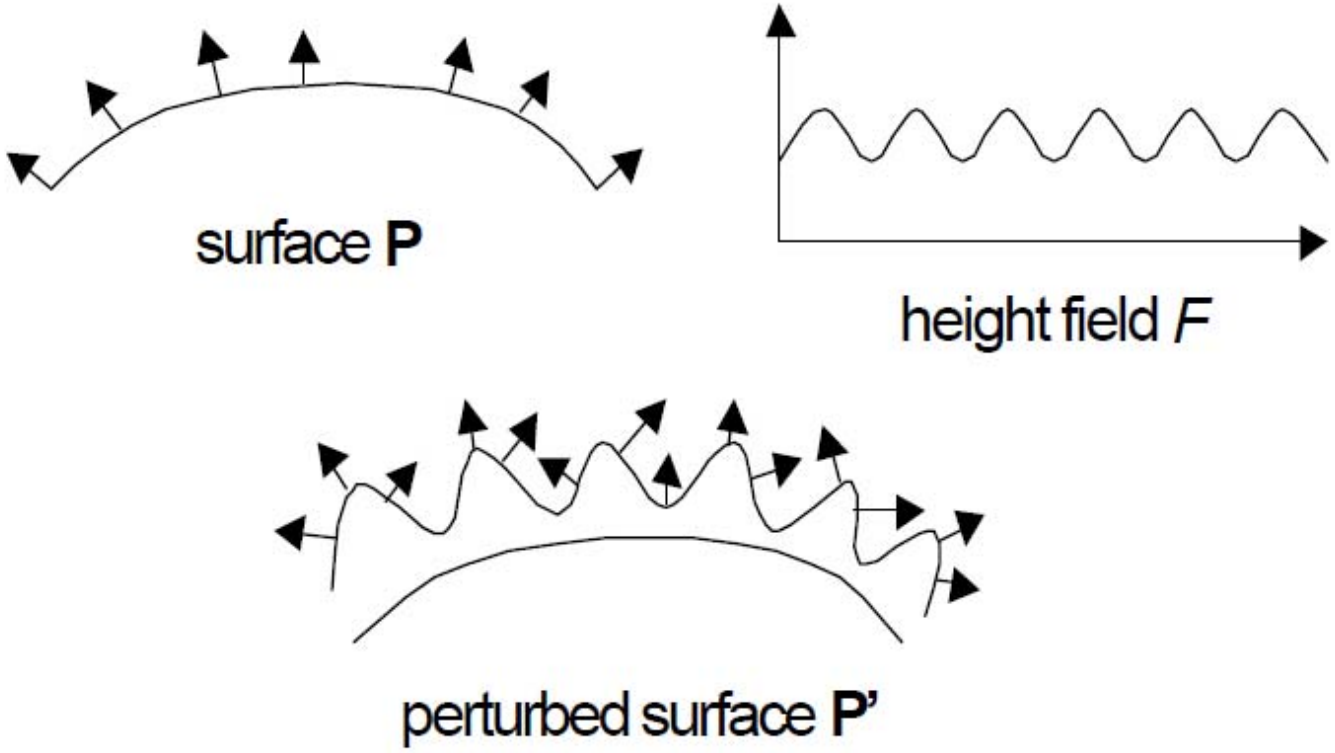
BUMP MAPPING

Bump mapping

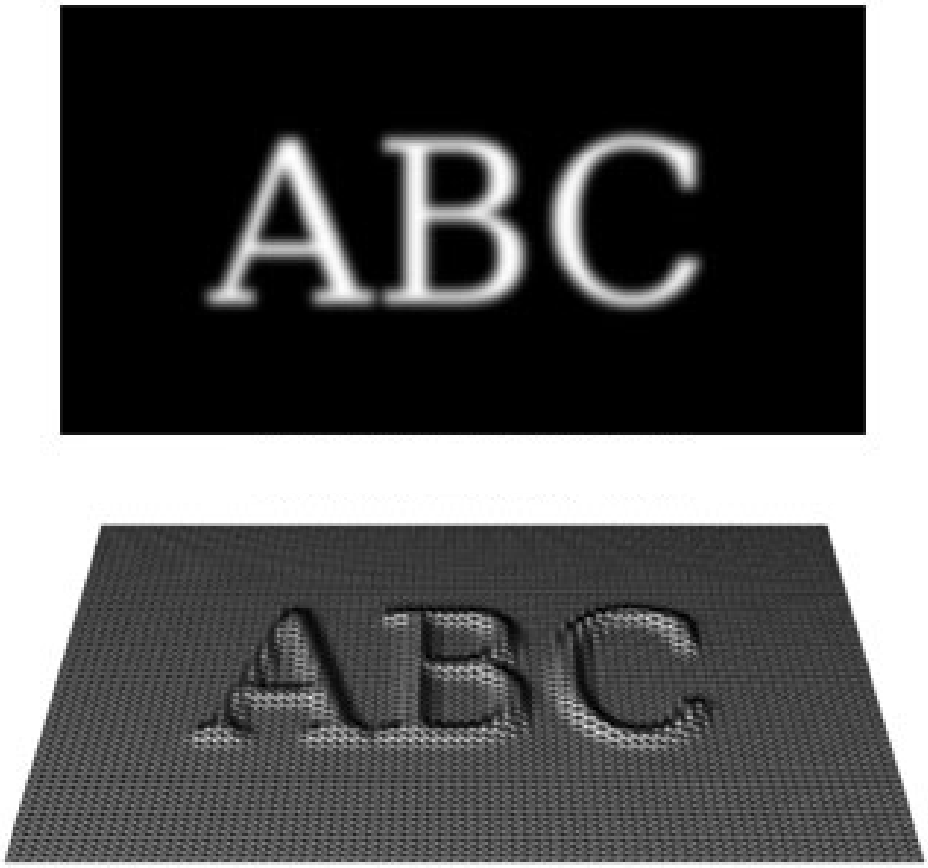




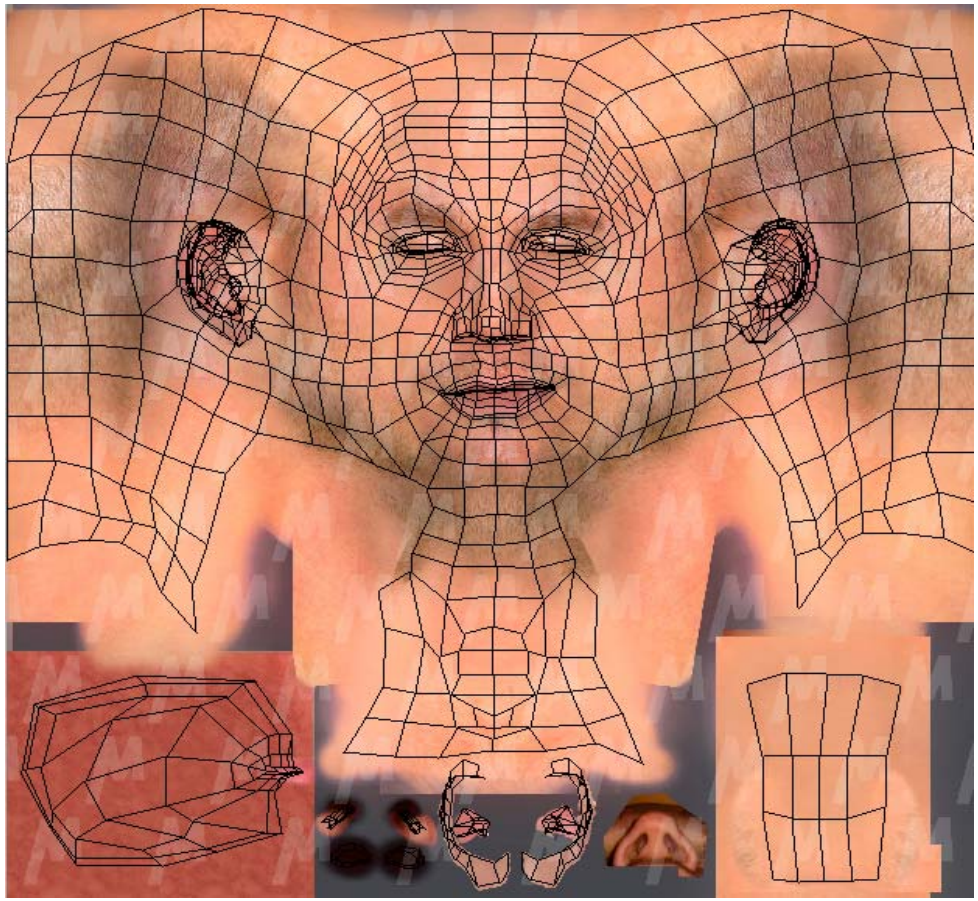
Elements bàsics

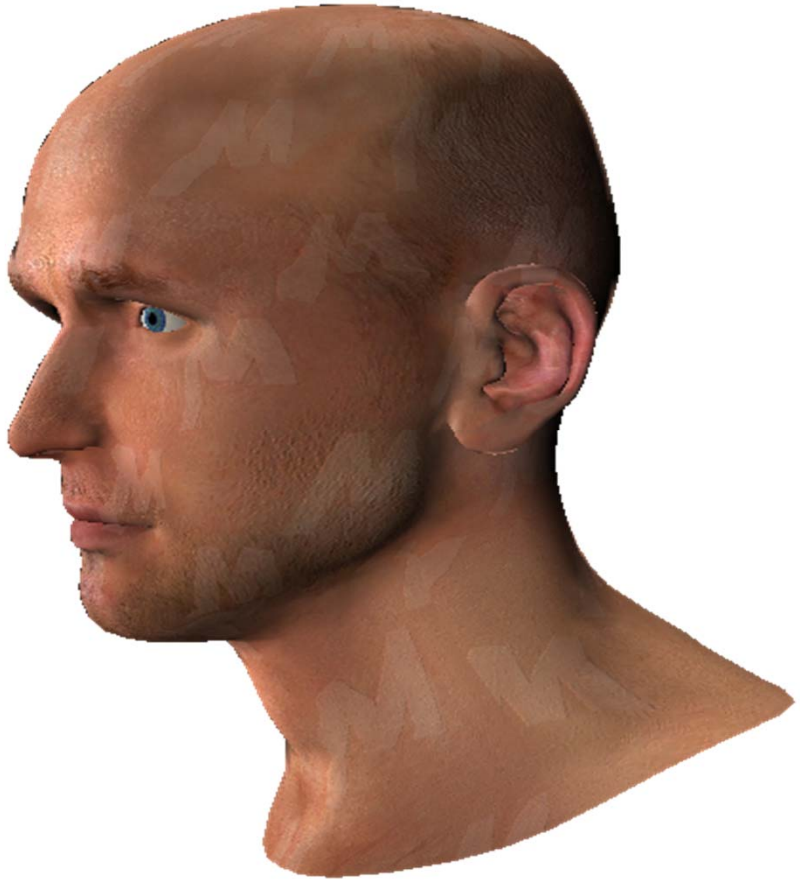


Height field

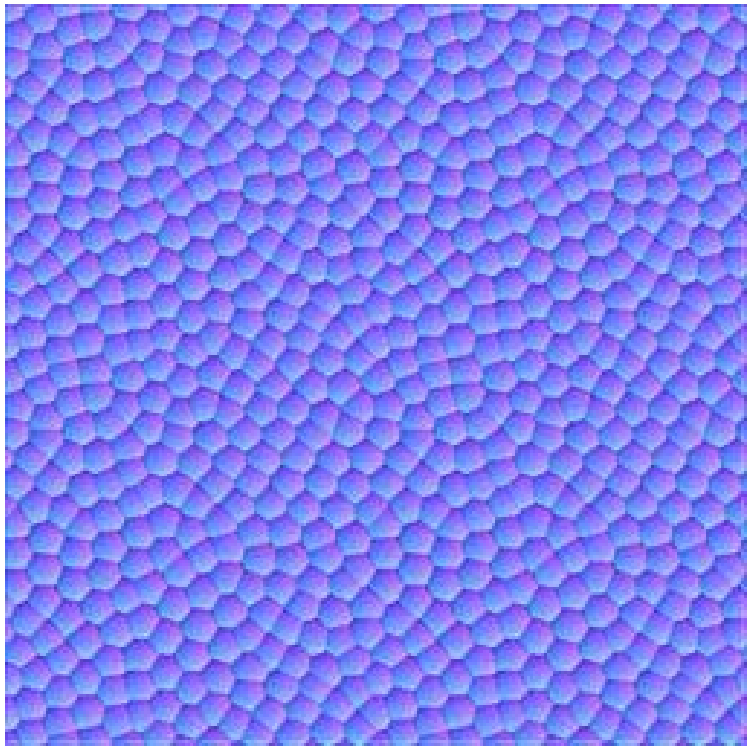


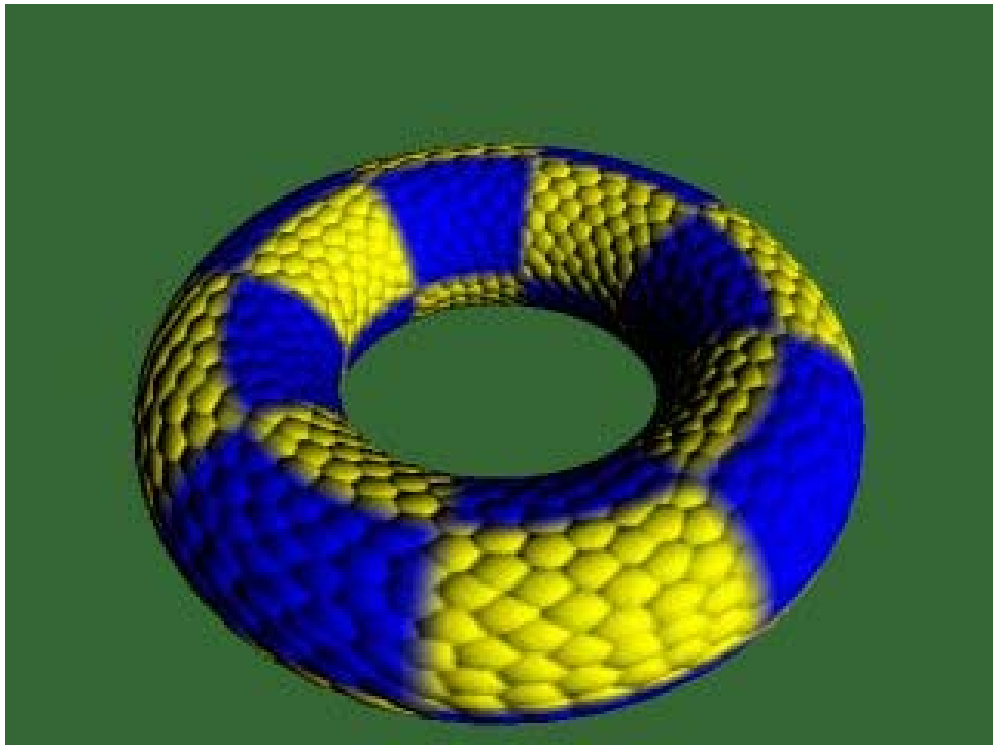
Malla amb coordenades de textura





Elements bàsics





Eqüacions

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

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

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

Eqüacions

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

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

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

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

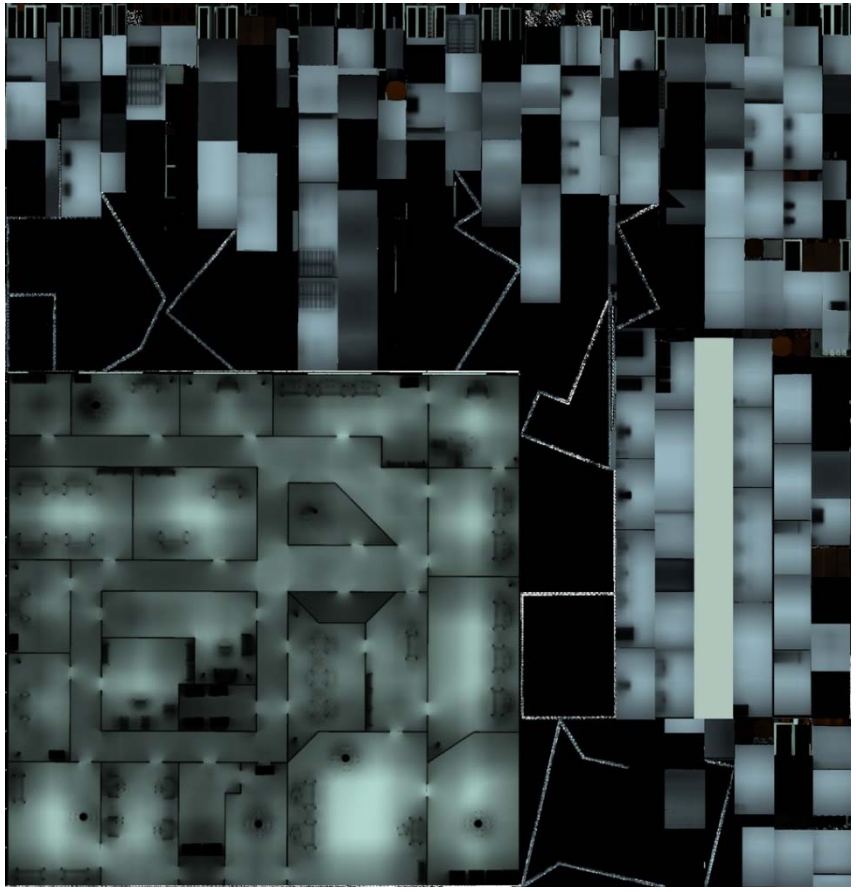
Eqüacions - resum

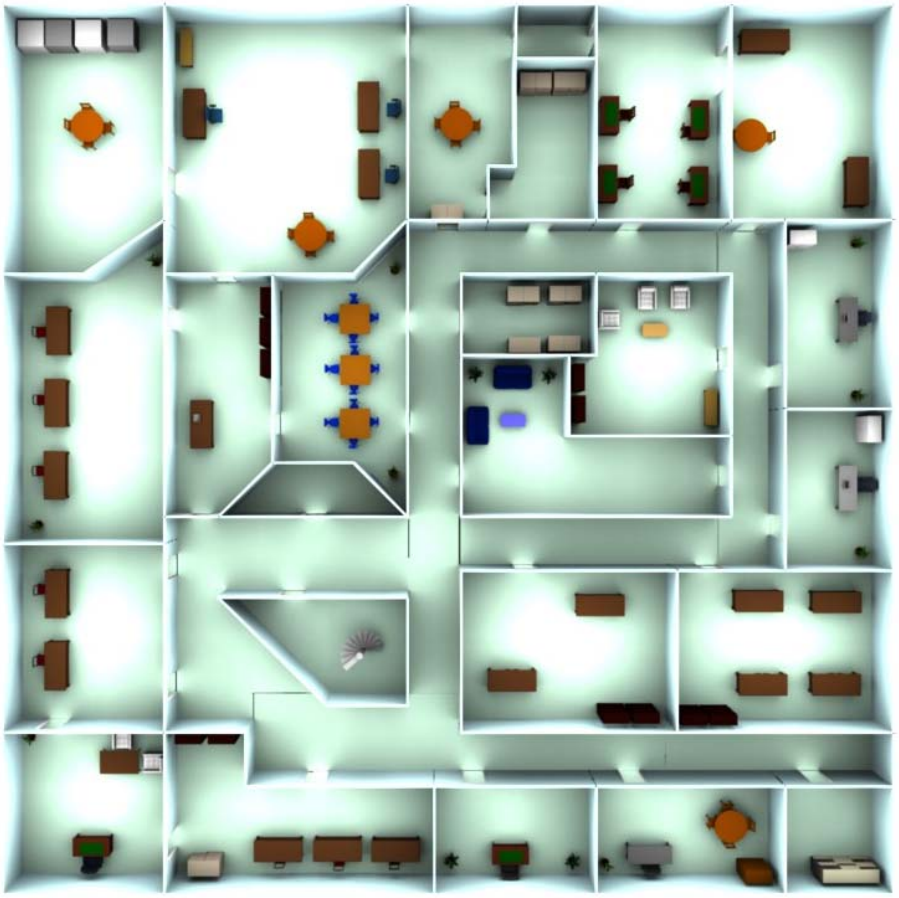
$$\mathbf{N}'(u,v) = \frac{\partial \mathbf{P}'(u,v)}{\partial u} \times \frac{\partial \mathbf{P}'(u,v)}{\partial v}$$

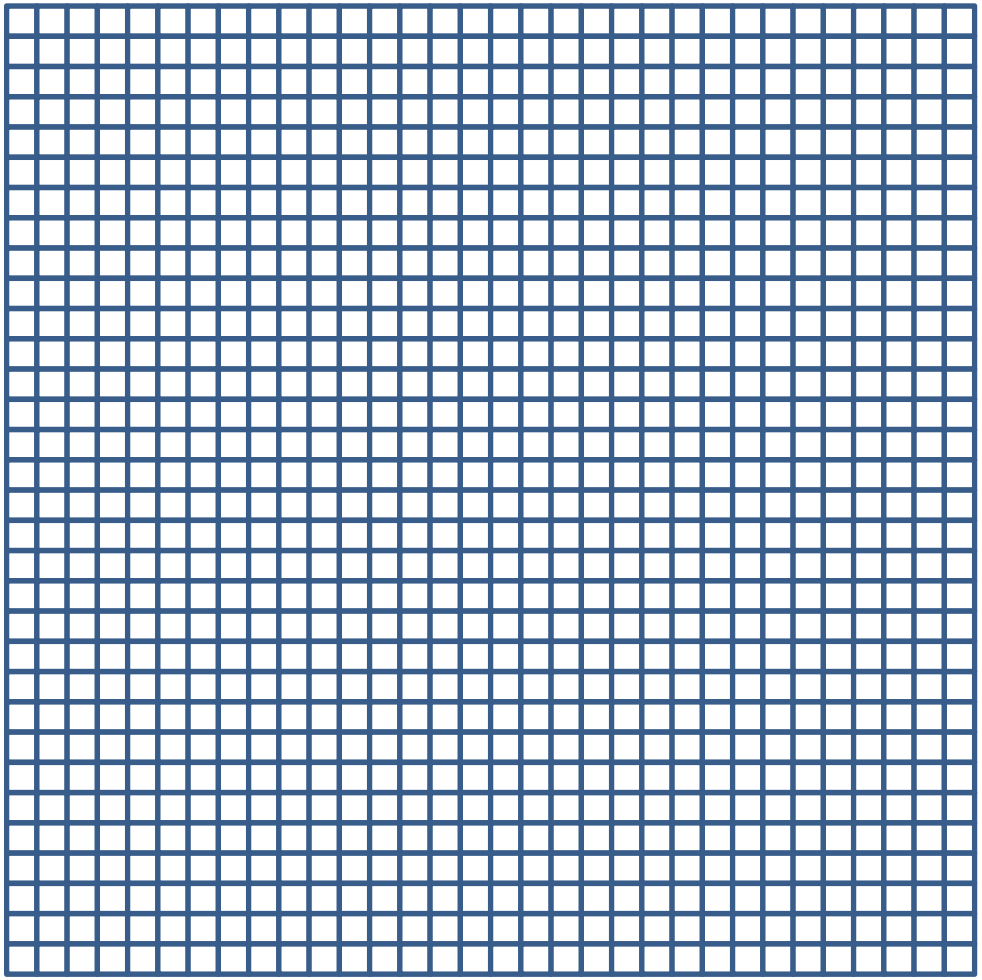
$$\mathbf{P}'(u,v) = \mathbf{P}(u,v) + F(u,v)\frac{\mathbf{N}(u,v)}{\left|\mathbf{N}(u,v)\right|}$$

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

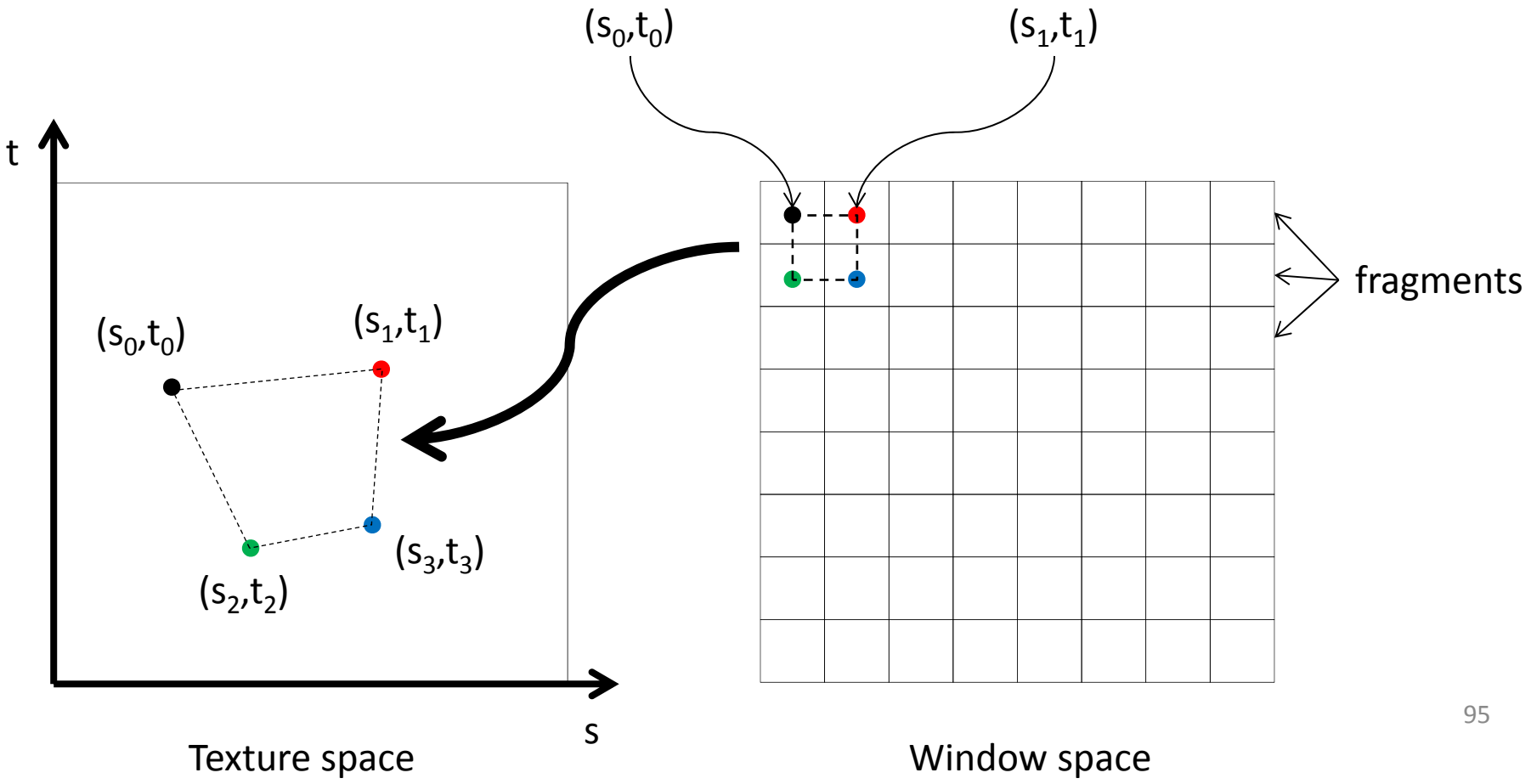
Color mapping





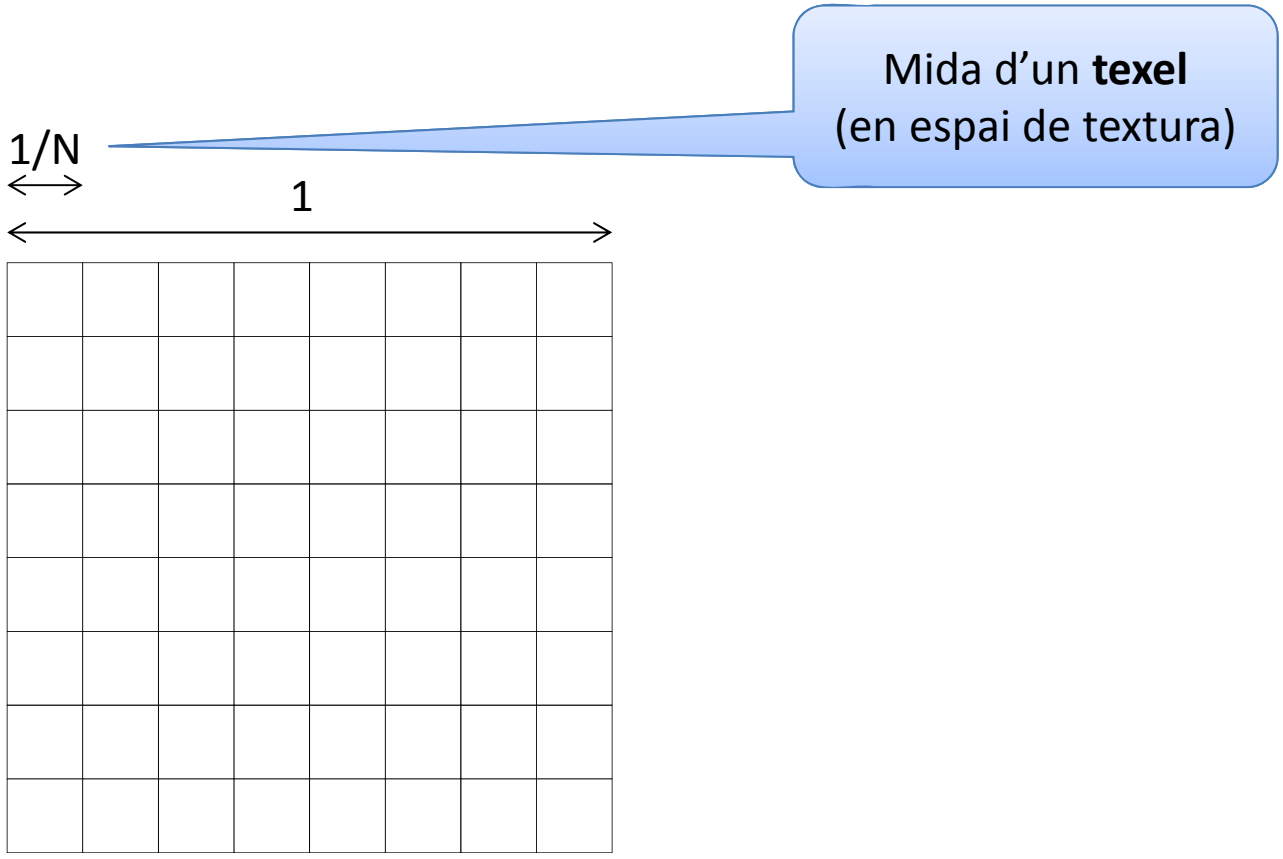


Coordenades (s,t) de fragments veïns



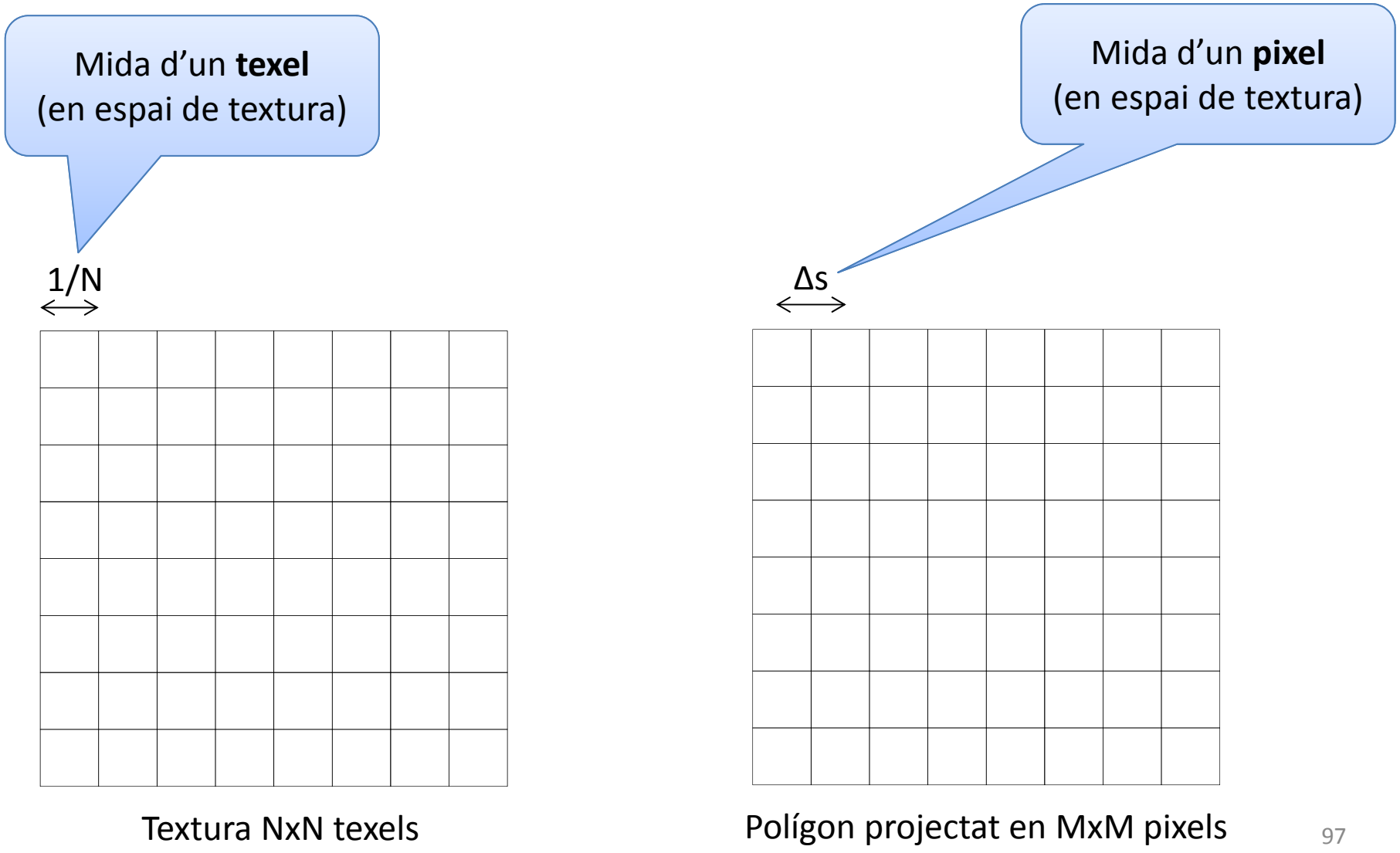
Mida d'un texel

Si una textura té NxN texels, cada texel té mida = 1/N



Textura NxN texels

Mida d'un pixel



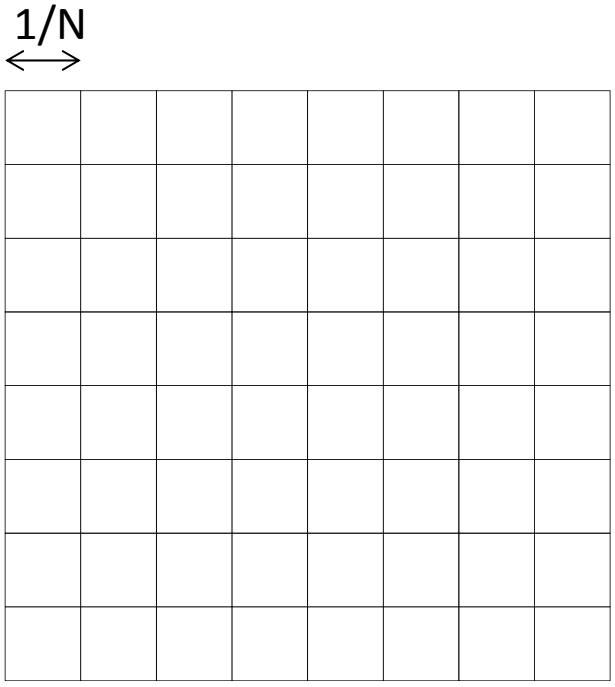
Situació ideal

Mida d'un **texel**
(en espai de textura)

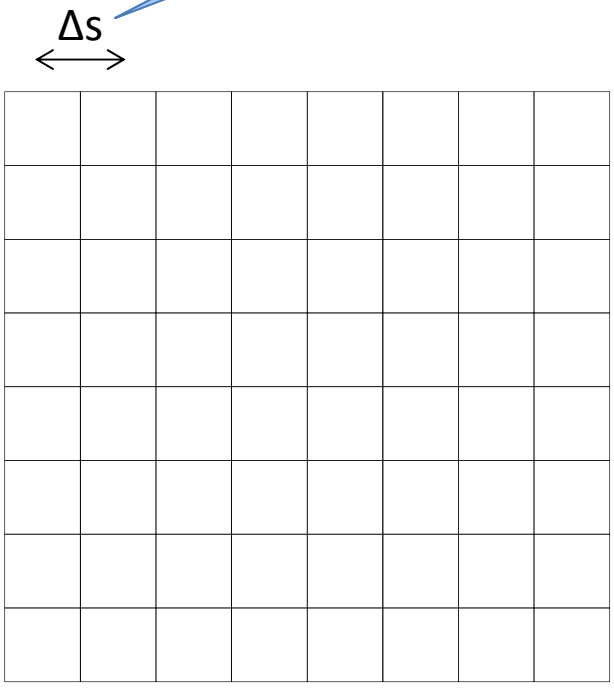
N=M

Δs = 1/N

Mida d'un **pixel**
(en espai de textura)



Textura NxN texels



Polígon projectat en MxM pixels

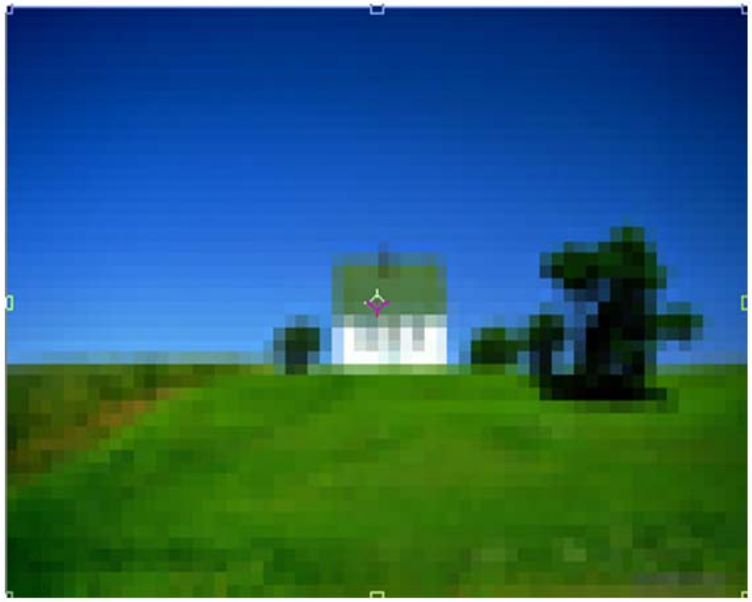
Magnification/Minification

Textura

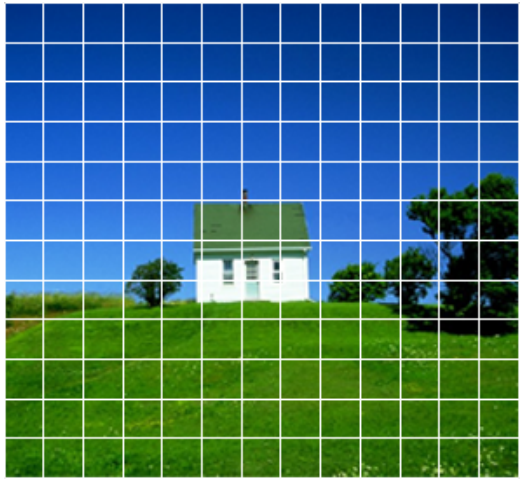
Δs < 1/N

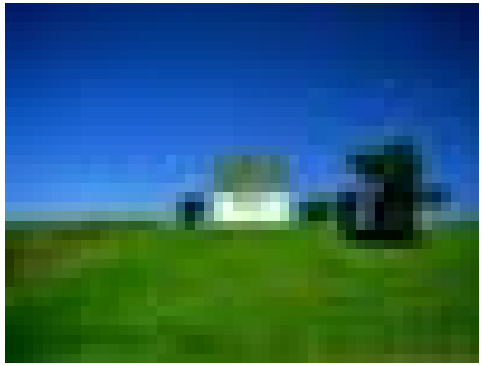
Δs > 1/N

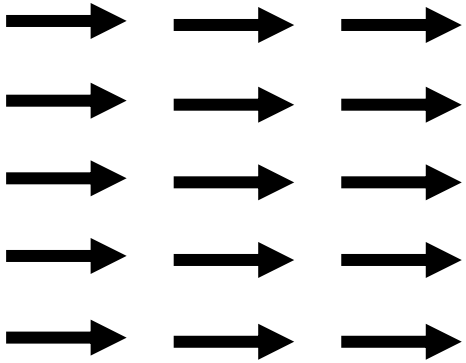
Magnification: NEAREST vs LINEAR



Minification (NEAREST / LINEAR)







```
uniform vec2 size;
varying vec3 normal;
uniform sampler2D sampler;
float computeMipLevel(vec2 uv)
{
        vec2 dx = dFdx(uv);
        vec2 dy = dFdy(uv);
        float rho = max( dot(dx, dx), dot( dy, dy ) );
        return max( 0.5 * log2(rho), 0.0 );
}
void main()
{
        float level = computeMipLevel(gl_TexCoord[0]*size);
        vec3 normalE = gl_NormalMatrix * normal;
        gl_FragColor = vec4(level/10.0)*texture2D(sampler,gl_TexCoorc
}
```

```
uniform sampler2D sampler;
void main()
{
  gl_FragColor = texture2D(sampler, gl_TexCoord[0].st);
}
```

```
// Torus + fieldstone
// Plane + tree.png
uniform sampler2D colormap;
uniform int mode; // 0..4
void main()
{
vec4 cF = vec4(gl_Color.rgb, 0.0);
vec4 cT = texture2D(colormap, gl_TexCoord[0].st);
switch(mode)
{
case 0: gl_FragColor = cF;
break;
case 1: gl_FragColor = cT;
break;
```