



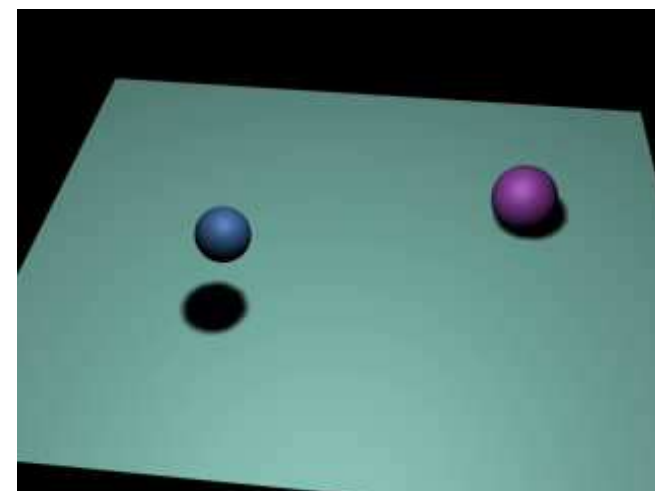
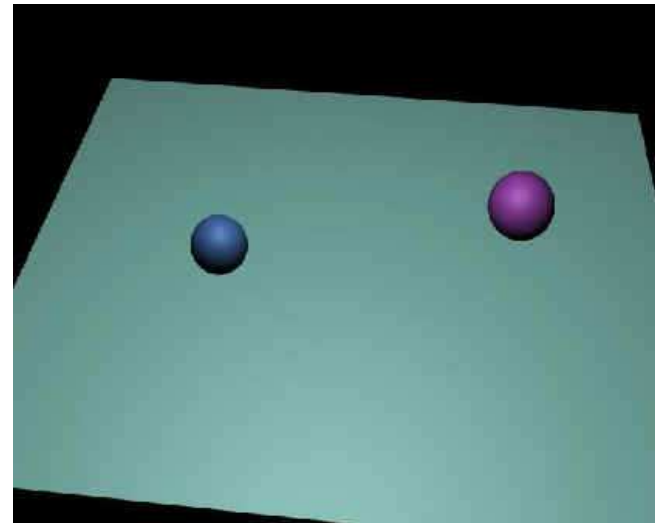
Ombres

Carlos Andújar

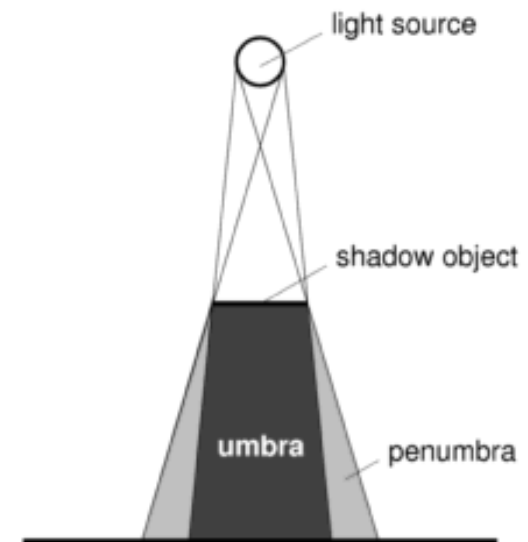
Març 2012

Avantatges de simular ombres

- Més realisme
- Indicació visual de profunditat



Umbra i penumbra

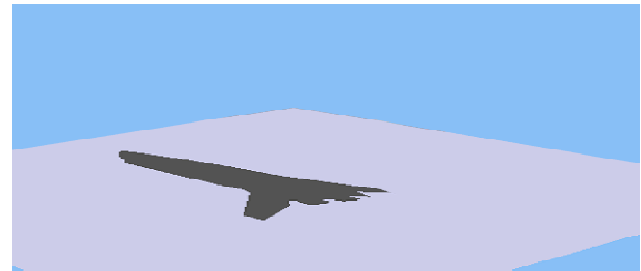
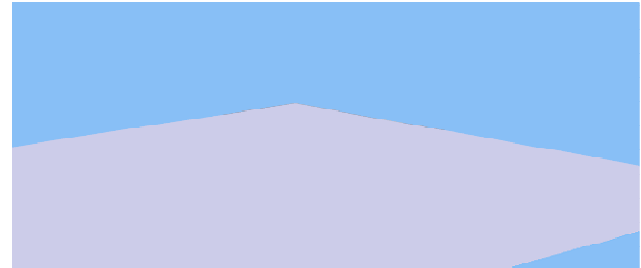


Ombres per projecció (un pla)

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

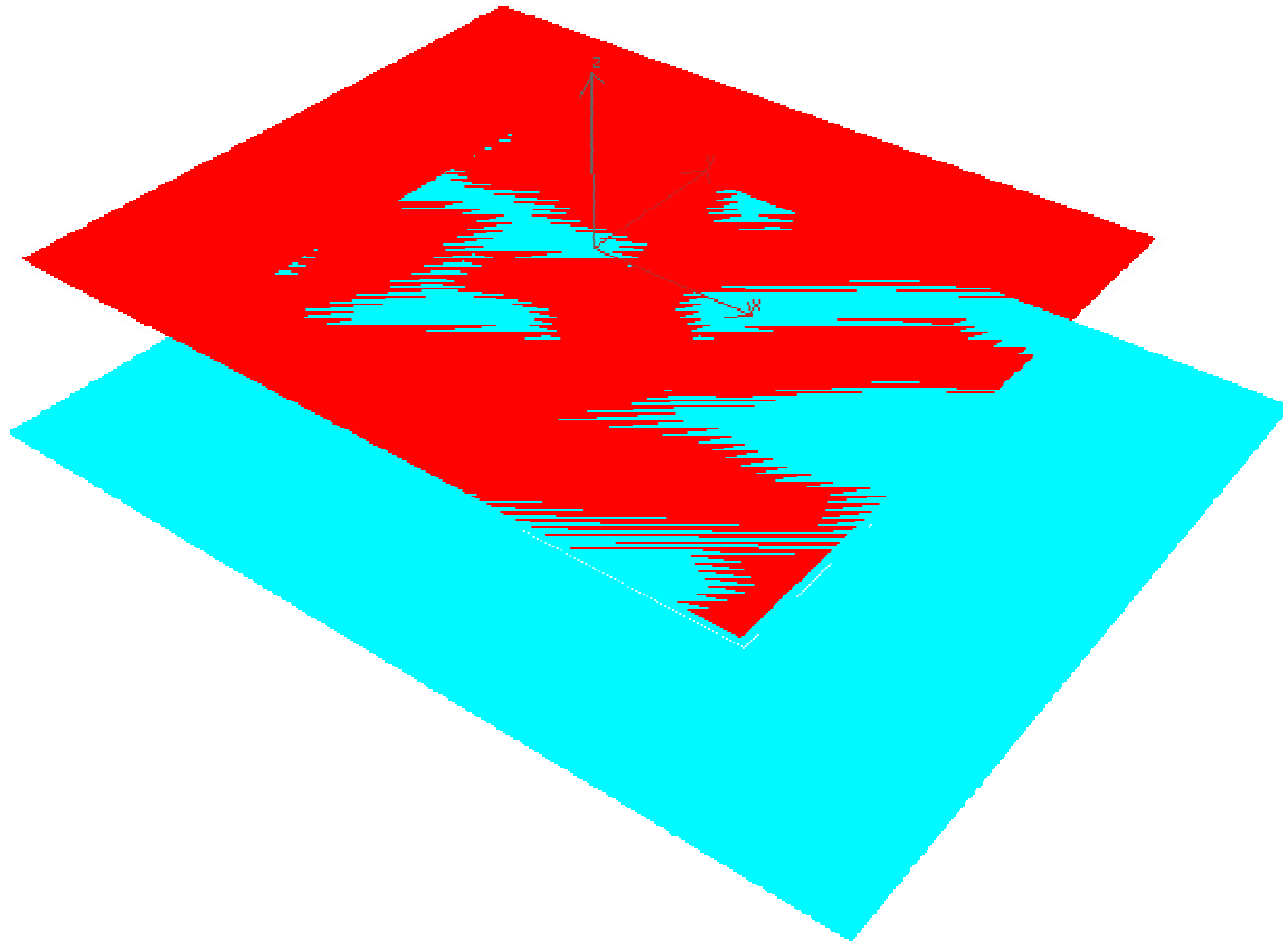
```
// 2. Dibuixar l'emissor projectat (ombra)  
glDisable (GL_LIGHTING);  
glDisable (GL_DEPTH_TEST);  
glMatrixMode (GL_MODELVIEW);  
glPushMatrix ();  
glMultMatrixf (MatriuProjeccio);  
dibuixa (oclusor);  
glPopMatrix ();
```

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





Z-fighting





Evitar problemas 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

$$z' = z + dz \cdot factor + r \cdot units$$

on

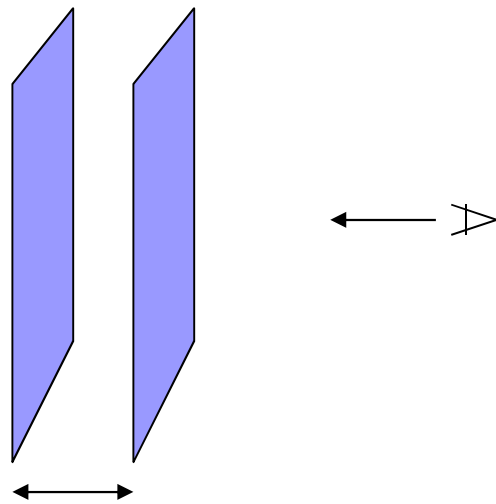
$$dz = \max(dz/dx, dz/dy)$$

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

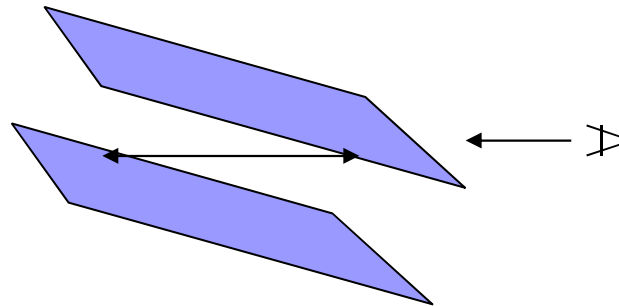
- El paràmetre *factor* permet introduir un **offset variable** (depen de la inclinació del polígon)
- El paràmetre *units* permet introduir un **offset constant**

Evitar problemas de z-fighting

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



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



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

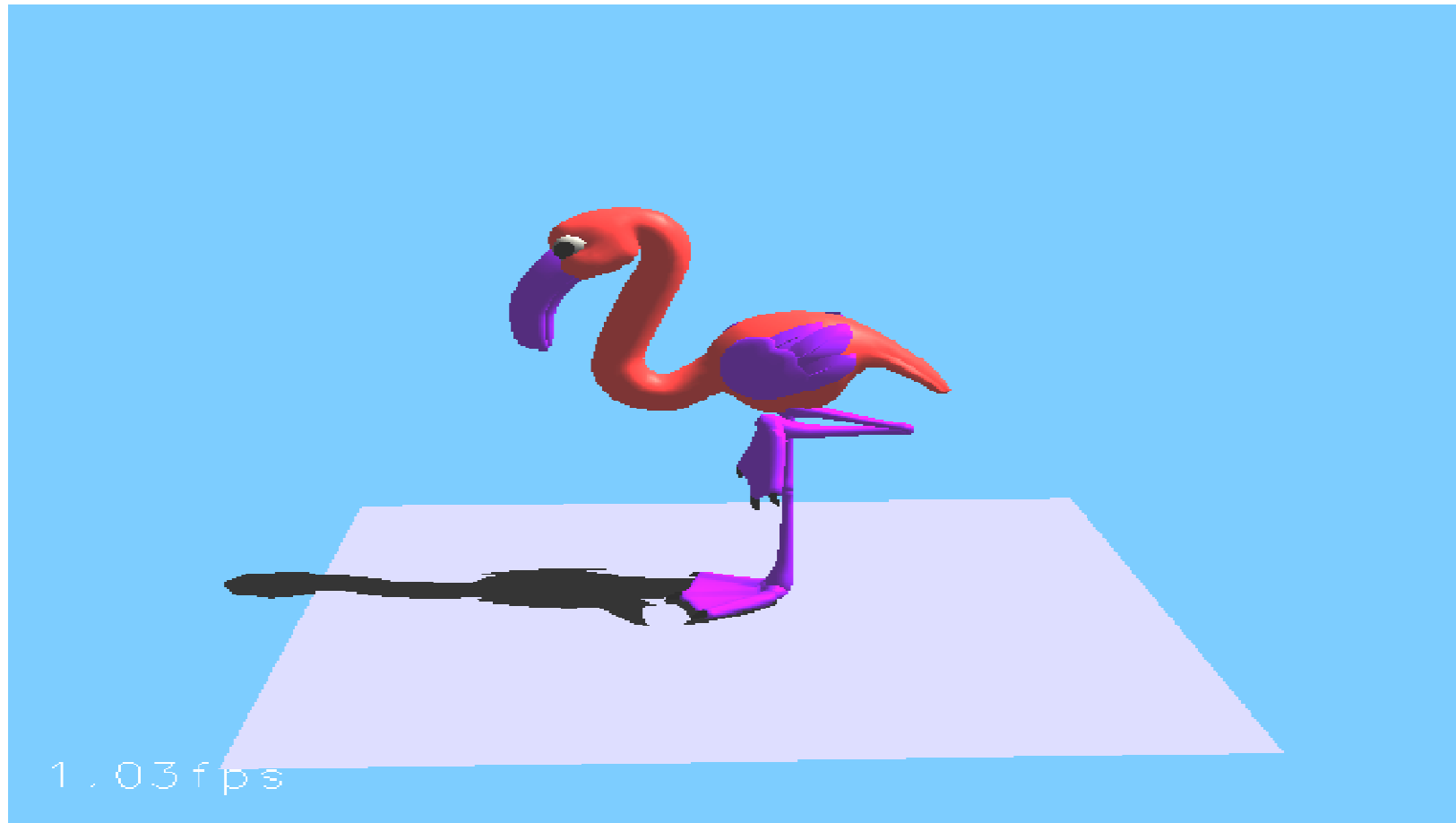


Evitar problemas de z-fighting

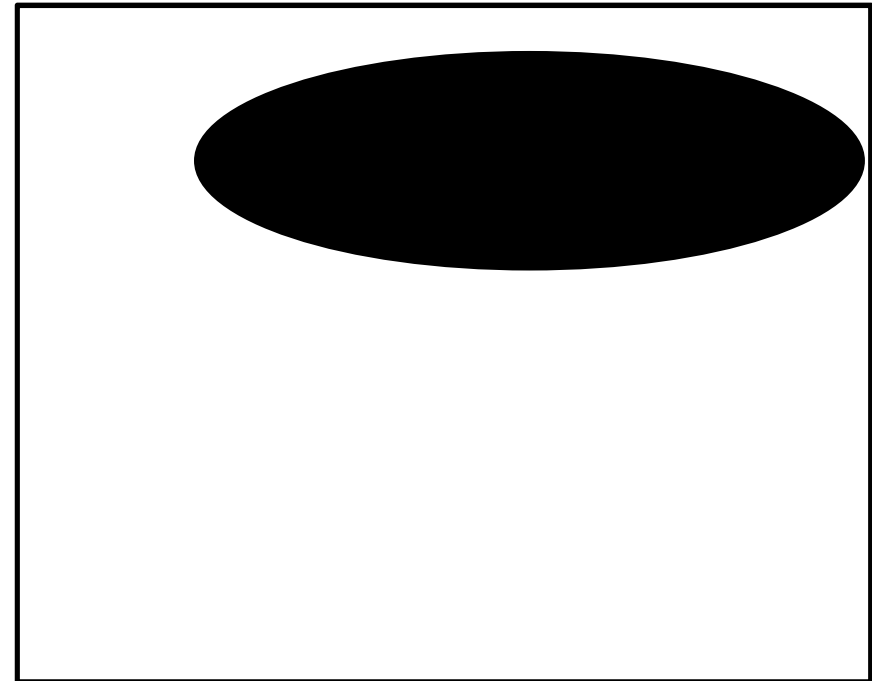
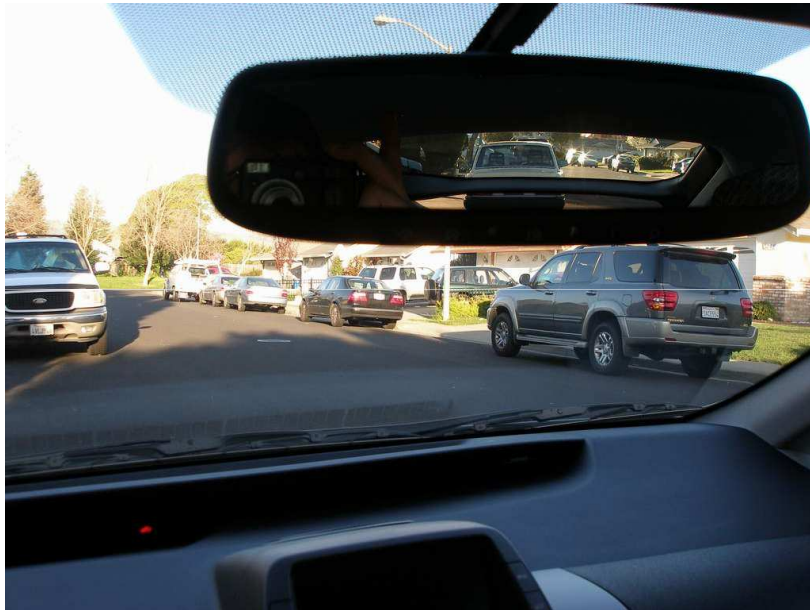
Valors típics: `glPolygonOffset(1, 1);`

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

Ombres per projecció (un pla)



Stencil buffer





Stencil buffer

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

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



Stencil buffer

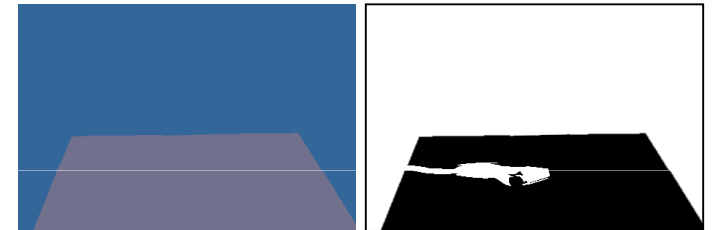
- Establir el test de comparació:
 - `glEnable(GL_STENCIL_TEST);`
 - **`glStencilFunc`**(comparació, valorRef, mask)
 - Comparació pot ser: `GL_NEVER`, `GL_ALWAYS`, `GL_LESS`...
 - Ex: `GL_LESS`: $(\text{valorRef} \& \text{mask}) < (\text{valorStencil} \& \text{mask})$
- Operacions a fer a stencil buffer 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 refèrència)

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);
```



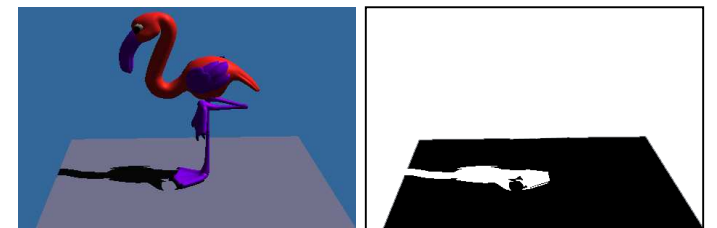
```
// 2. Dibuixa oclusor per netejar l'stencil a les zones a l'ombra  
glDisable(GL_DEPTH_TEST);  
glColorMask(GL_FALSE, ... GL_FALSE);  
glStencilFunc(GL_EQUAL, 1, 1);  
glStencilOp(GL_KEEP, GL_KEEP, GL_ZERO);  
glPushMatrix(); glMultMatrixf(MatriuProjeccio);  
dibuixa(oclusor);  
glPopMatrix();
```



```
// 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);
```

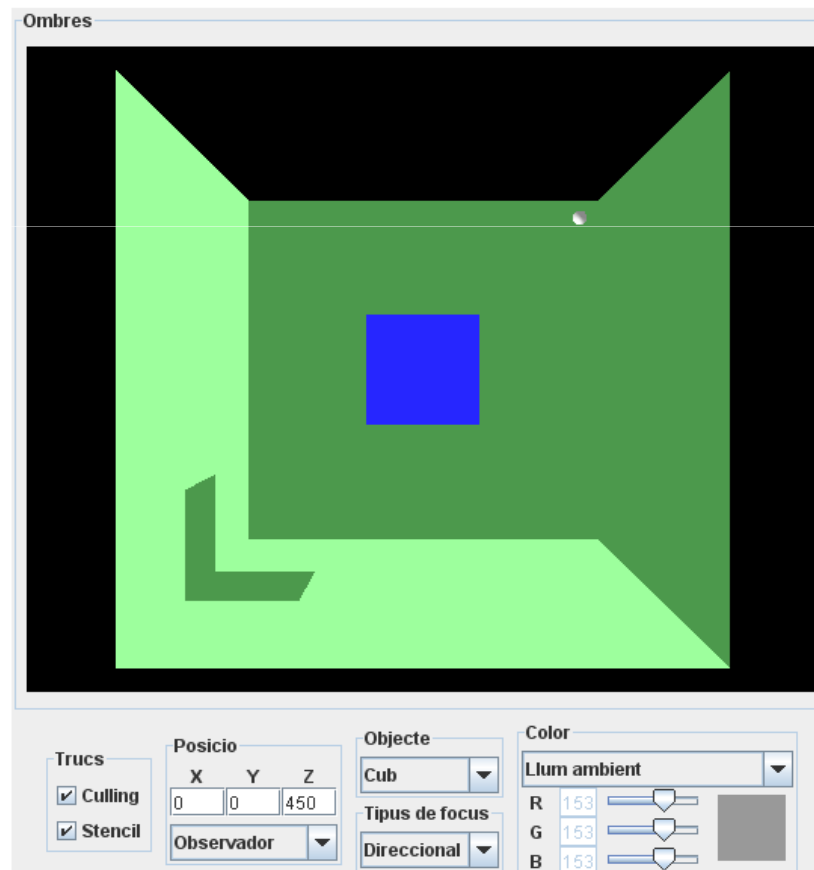


```
// 4. Dibuixa l'oclusor  
glEnable(GL_LIGHTING);  
glDepthFunc(GL_LESS);  
glDisable(GL_STENCIL_TEST);  
Dibuixa(oclusor);
```



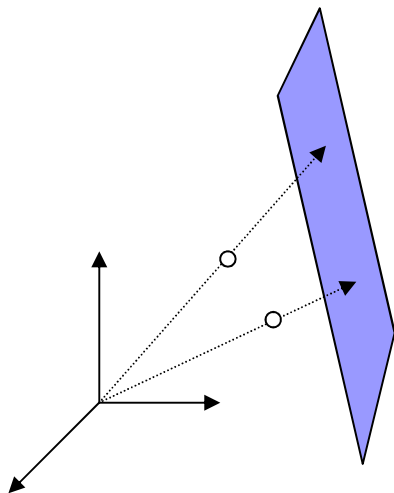
Ombres per projecció (amb stencil)

Experimenteu amb l'applet d'ombres del CD-ROM:



Projecció respecte l'origen

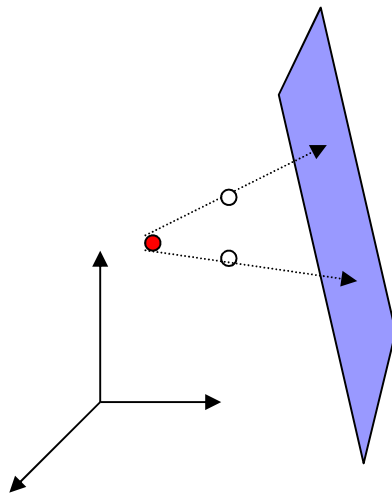
- Donats els coeficients (a,b,c,d) d'un pla, la matriu de projecció respecte l'origen és:



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

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:

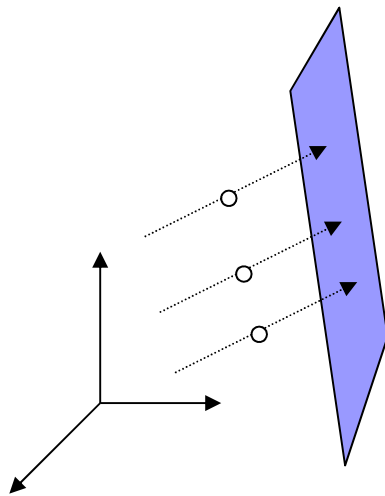


$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -(d+ax+by+cz) & 0 & 0 & 0 \\ 0 & -(d+ax+by+cz) & 0 & 0 \\ 0 & 0 & -(d+ax+by+cz) & 0 \\ a & b & c & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 &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

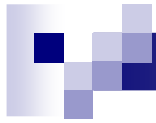
$-d - by - cz$	xb	xc	xd
ya	$-d - ax - cz$	yc	yd
za	zb	$-d - ax - by$	zd
a	b	c	$-ax - by - cz$

Projecció en la direcció (x,y,z)

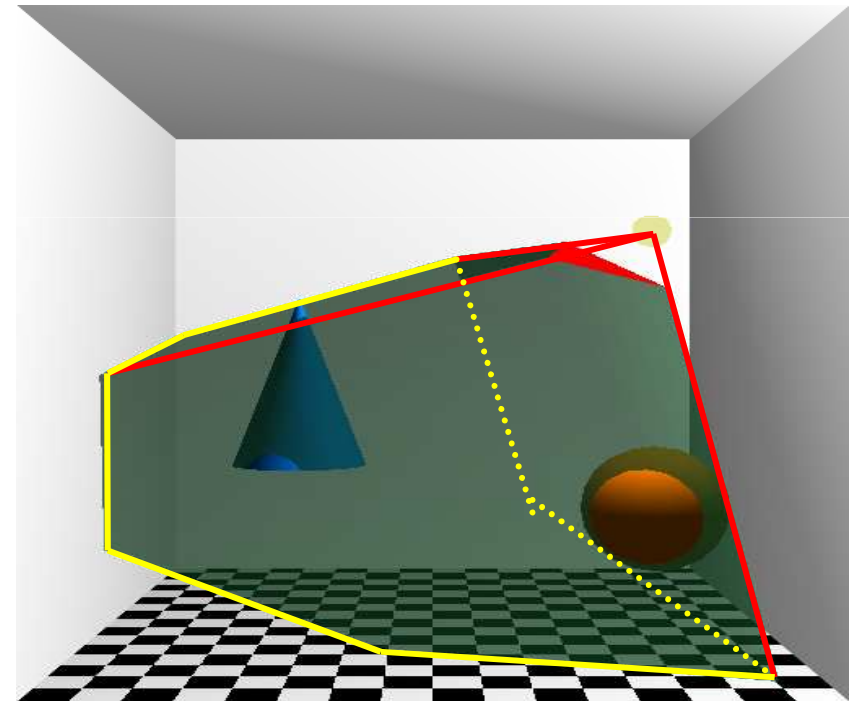
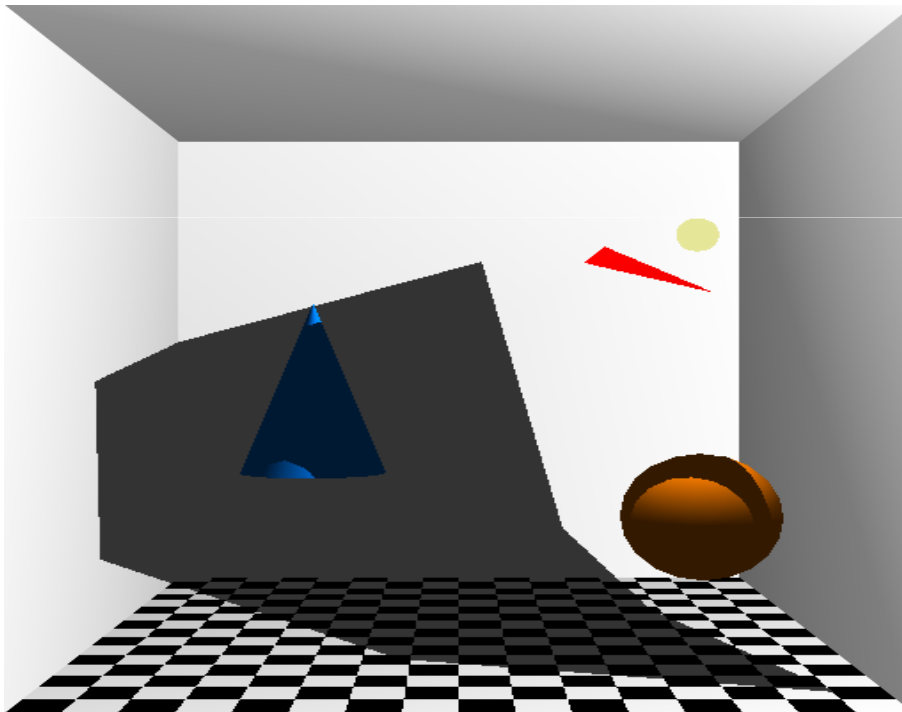
- Donats els coeficients (a,b,c,d) d'un pla, la matriu de projecció en la direcció del vector (x,y,z) és:



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



Volums d'ombra



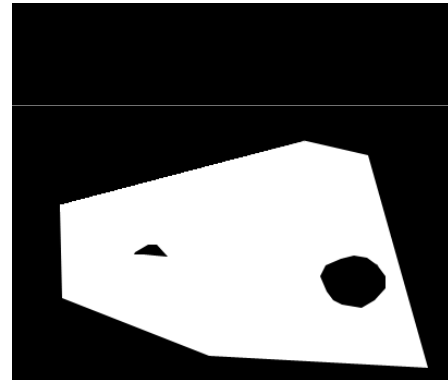
Volums d'ombra (1/2)

// 1. Dibuixa l'**escena** al z-buffer

```
glColorMask(GL_FALSE, ..., GL_FALSE);  
dibuixa(escena);
```

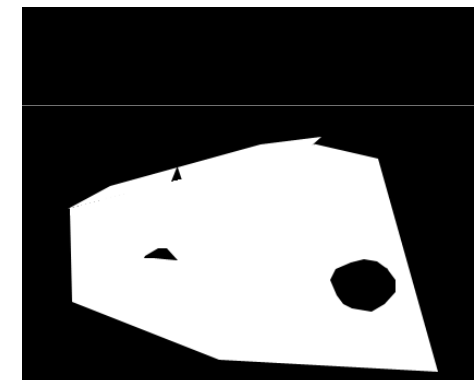
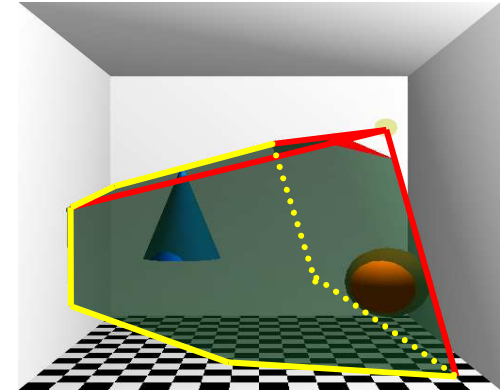
// 2. Dibuixa al stencil les **cares frontals del volum**

```
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

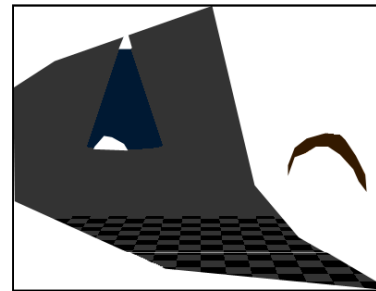
```
glDepthMask(GL_TRUE);  
glColorMask(GL_TRUE, ... , GL_TRUE);  
glCullFace(GL_BACK);  
glDepthFunc(GL_LEQUAL);  
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);  
glStencilFunc(GL_EQUAL, 1, 1);  
glDisable(GL_LIGHTING);  
dibuixa(escena);
```

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

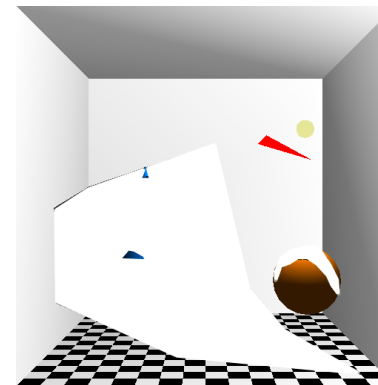
```
glStencilFunc(GL_EQUAL, 0, 1);  
glEnable(GL_LIGHTING);  
dibuixa(escena);
```

// 6. Restaura l'estat inicial

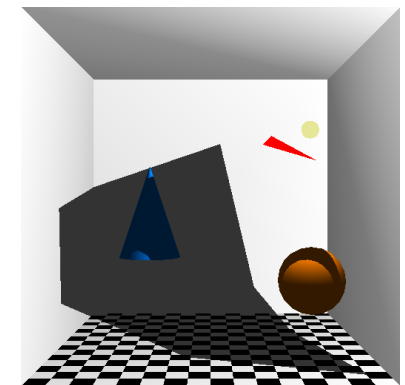
```
glDepthFunc(GL_LESS);  
glDisable(GL_STENCIL_TEST);
```



+

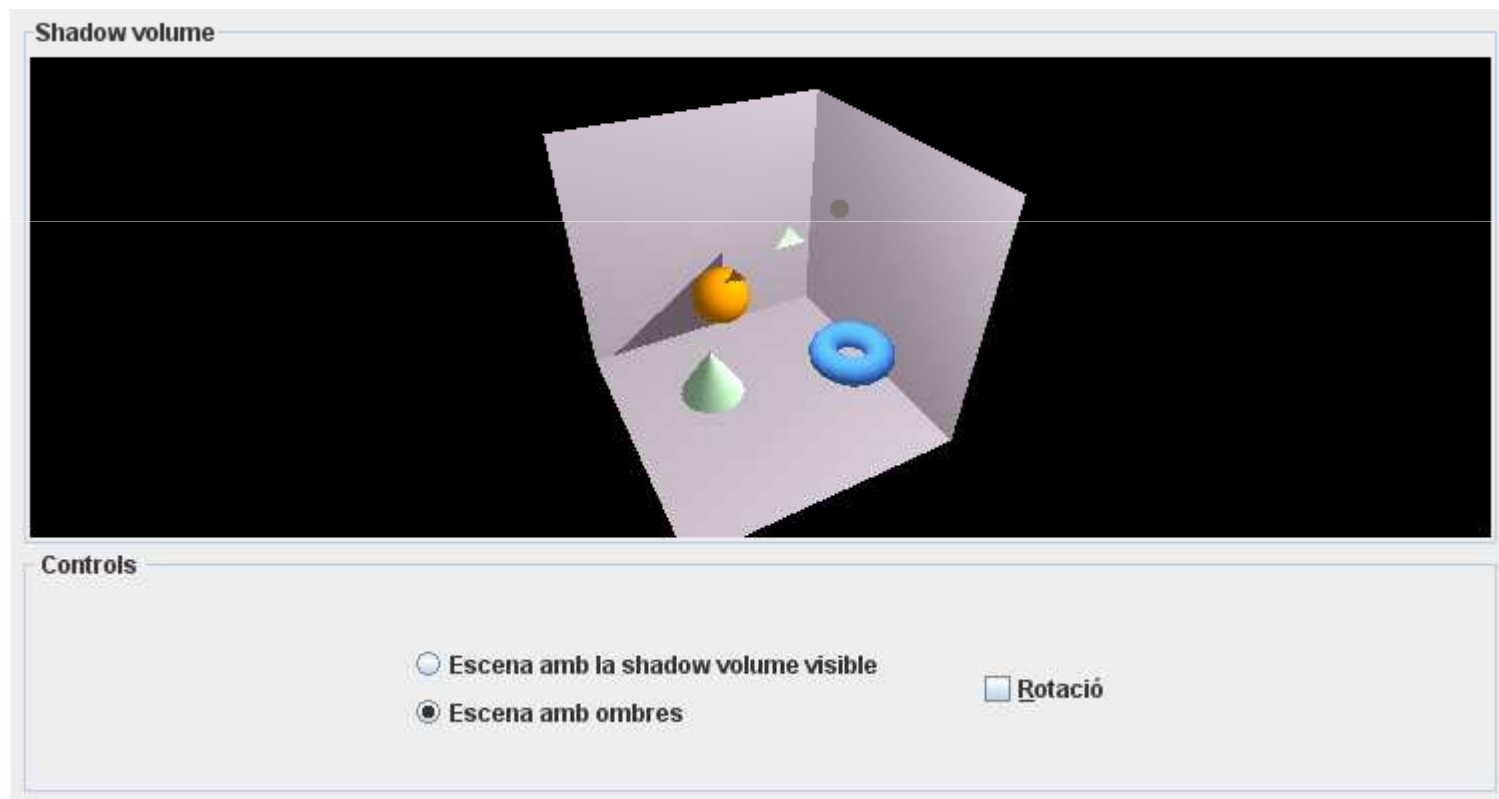


=



Volums d'ombra

- Proveu l'applet del CD-ROM:





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);
```

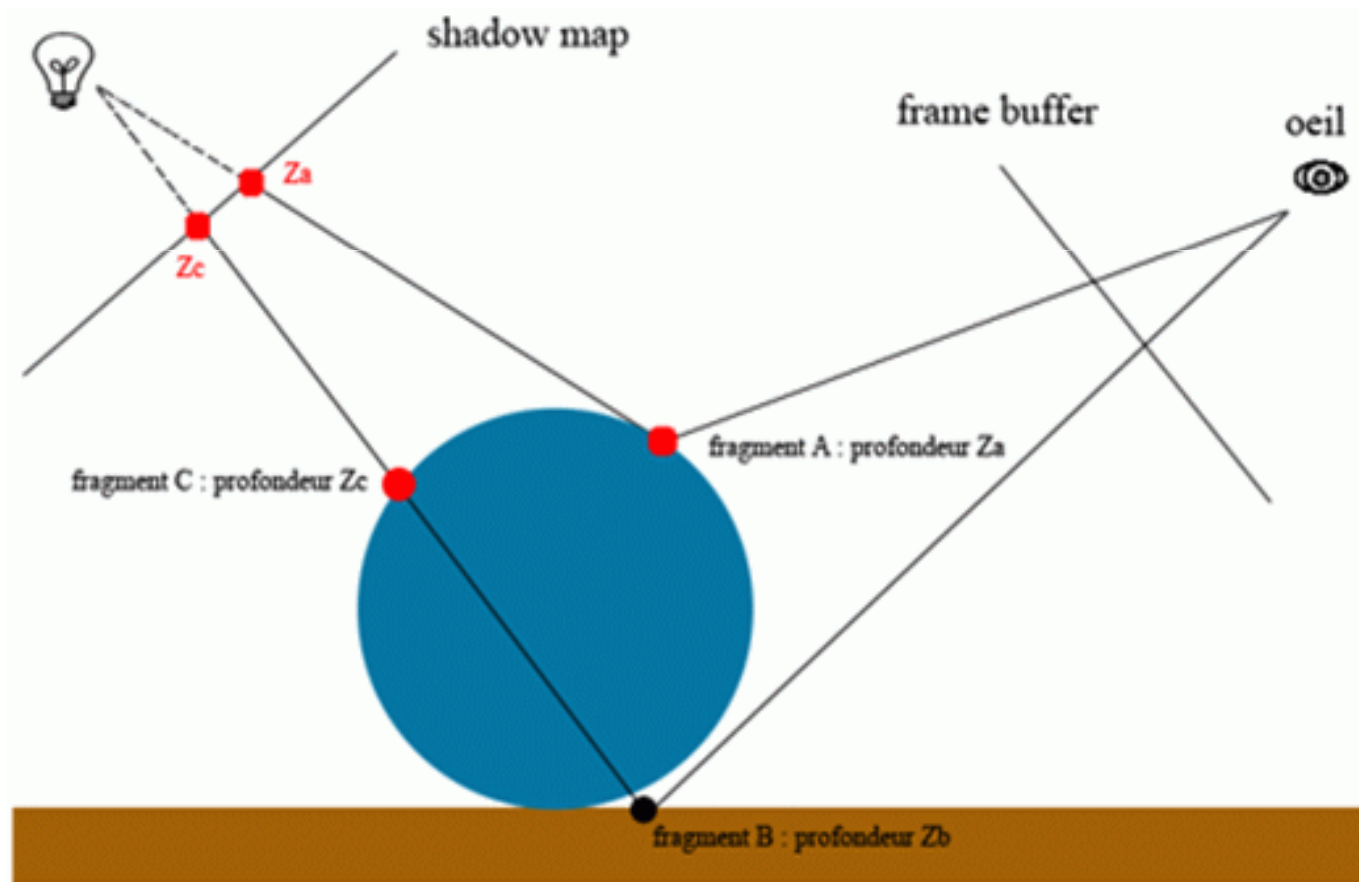
```
glTexGenf(GL_T, GL_OBJECT_PLANE, t);
```

```
glTexGenf(GL_R, GL_OBJECT_PLANE, r);
```

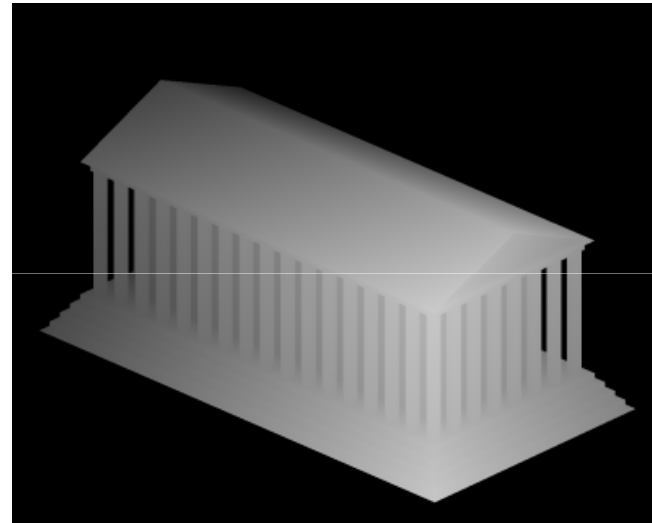
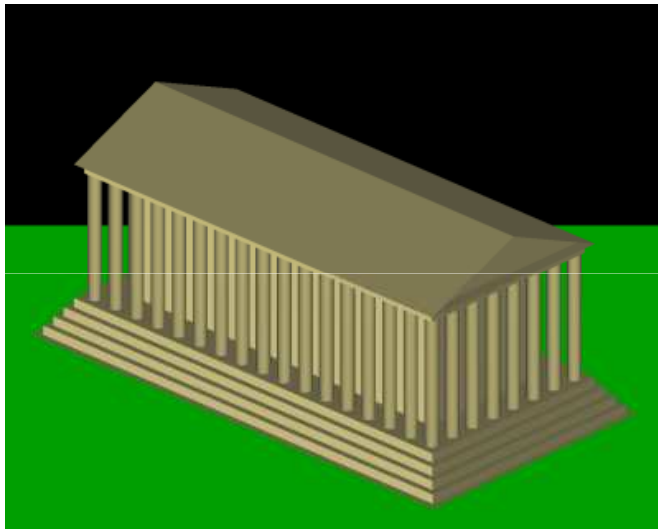
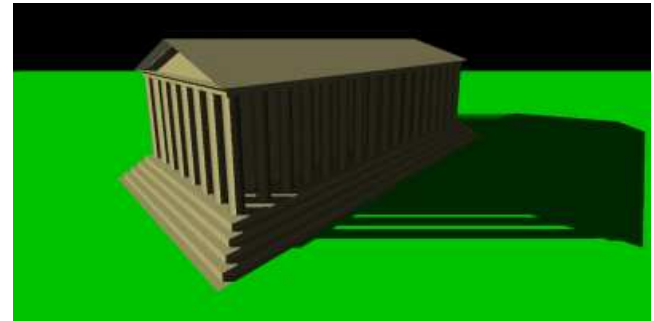
```
glTexGenf(GL_Q, GL_OBJECT_PLANE, q);
```

```
dibuixarEscena()
```

Shadow mapping



Shadow mapping





Shadow mapping (1/3)

// Creació 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 );  
drawScene();
```

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

```
glCopyTexImage2D( GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, 0, 0,  
                  SHADOW_MAP_WIDTH, SHADOW_MAP_HEIGHT, 0 );
```

// Restaurar càmera i viewport



Shadow mapping (2/3)

// 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...);
```



Shadow mapping (3/3)

// Dibuix de l'escena

// Activar filtratge linial

```
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );  
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
```

// Activar el test amb el shadow map

```
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL );  
glTexParameteri( GL_TEXTURE_2D, GL_DEPTH_TEXTURE_MODE, GL_LUMINANCE );  
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE,  
                  GL_COMPARE_R_TO_TEXTURE );
```

// Dibuixar l'escena

```
drawObjects();
```

<http://www.cs.ualberta.ca/~ghali/courses/texts/redbook/html/09-10-shadowmap.c.html>