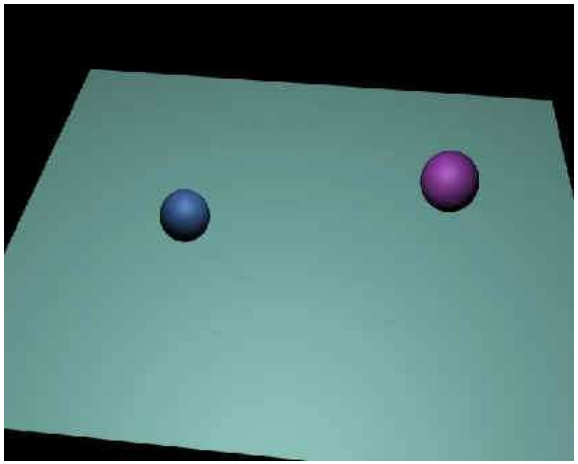
Simulació d'ombres

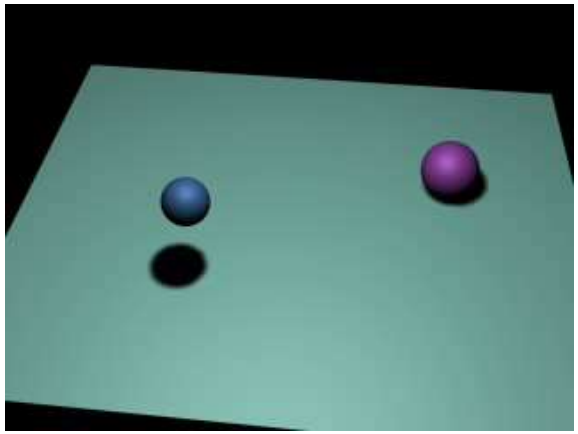
Carlos Andújar

Maig 2022

Avantatges de simular ombres

- Més realisme
- Indicació visual de profunditat

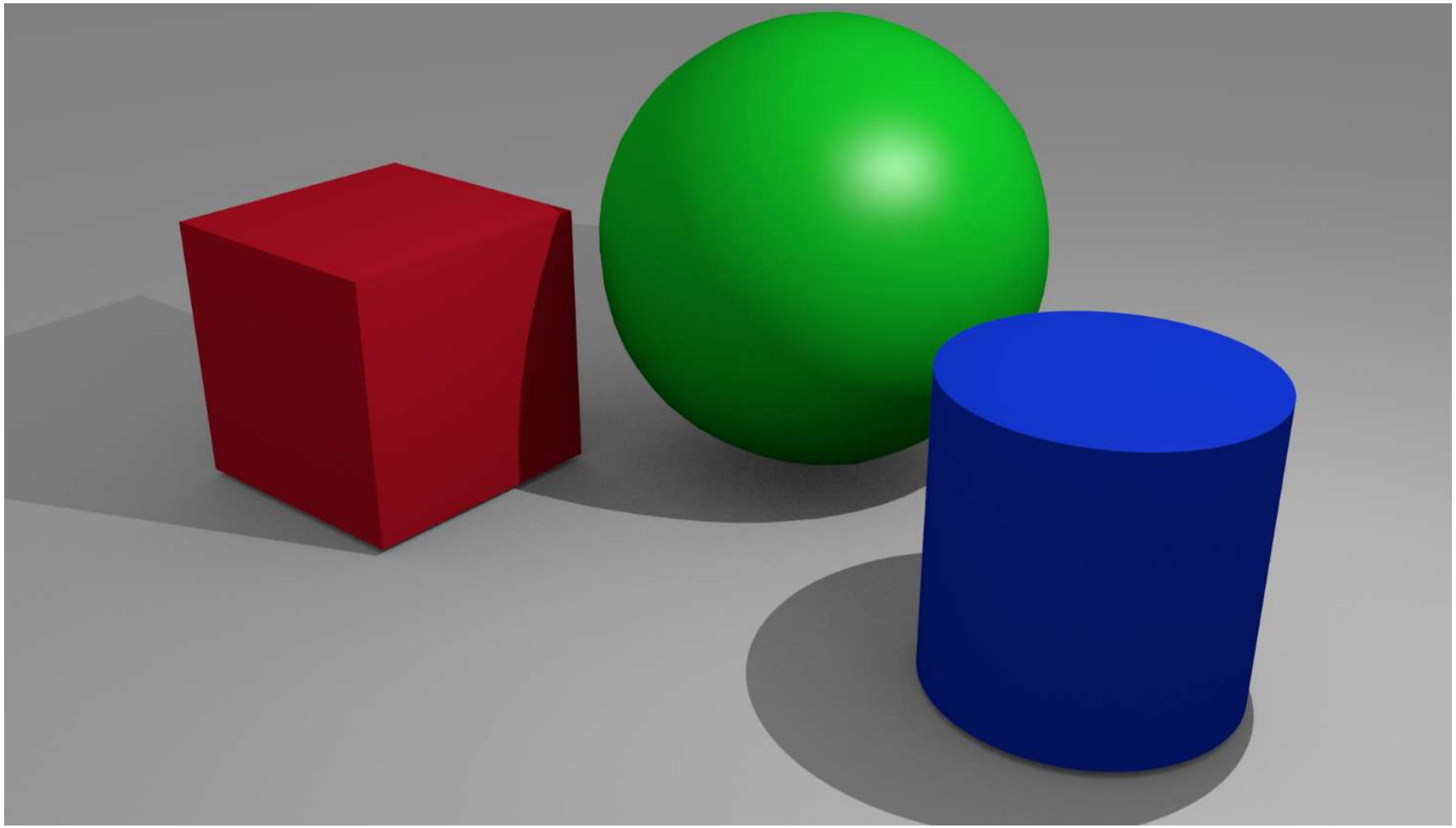




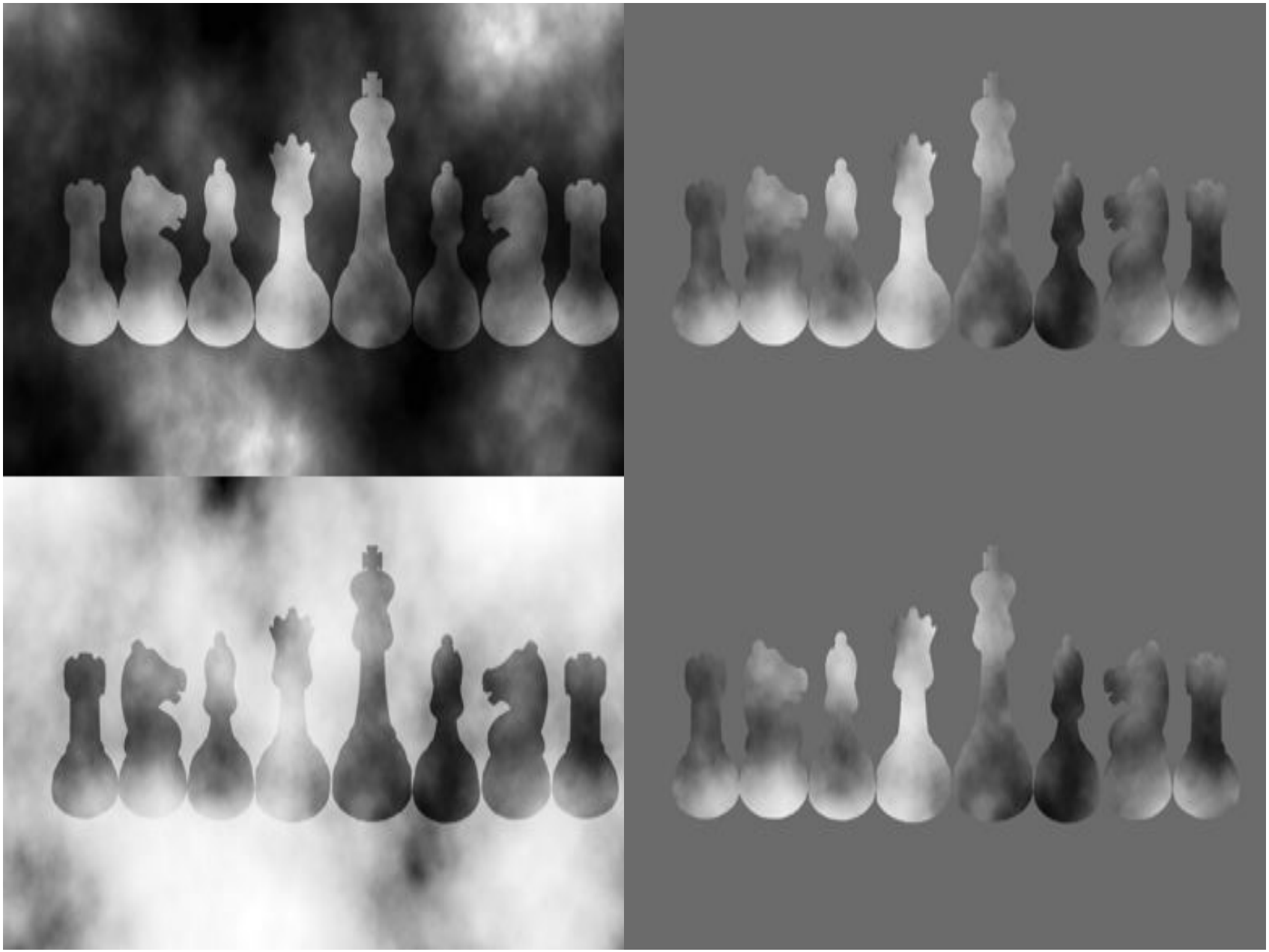
Avantatges de simular ombres

- Informació adicional

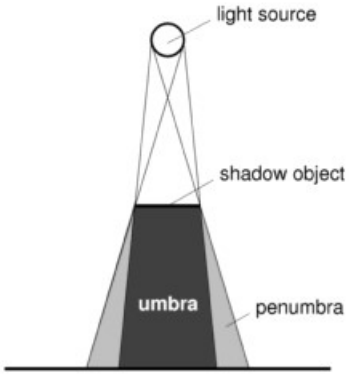
Percepció d’ombres



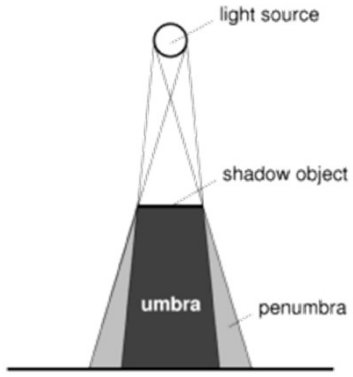

ECVP
2005
ECVP
2005



Umbra i penumbra



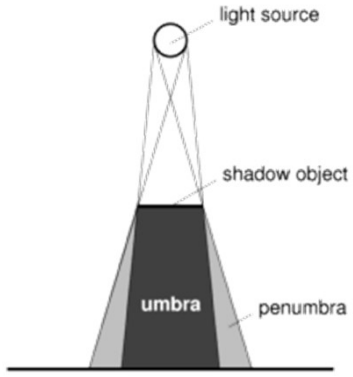
Propietats



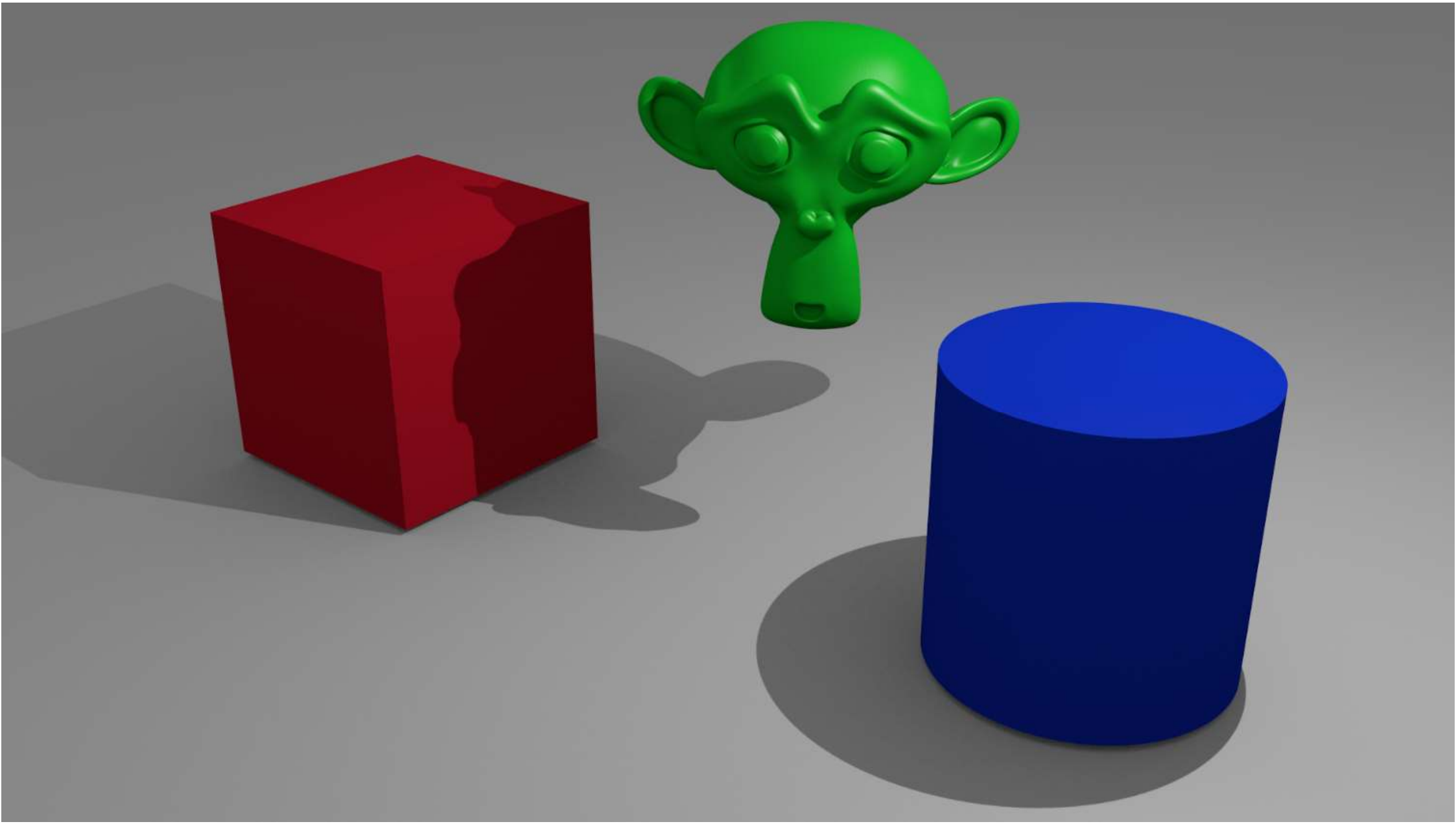
- Si la font de llum és puntual →
- Si augmenta la mida de la font de llum...

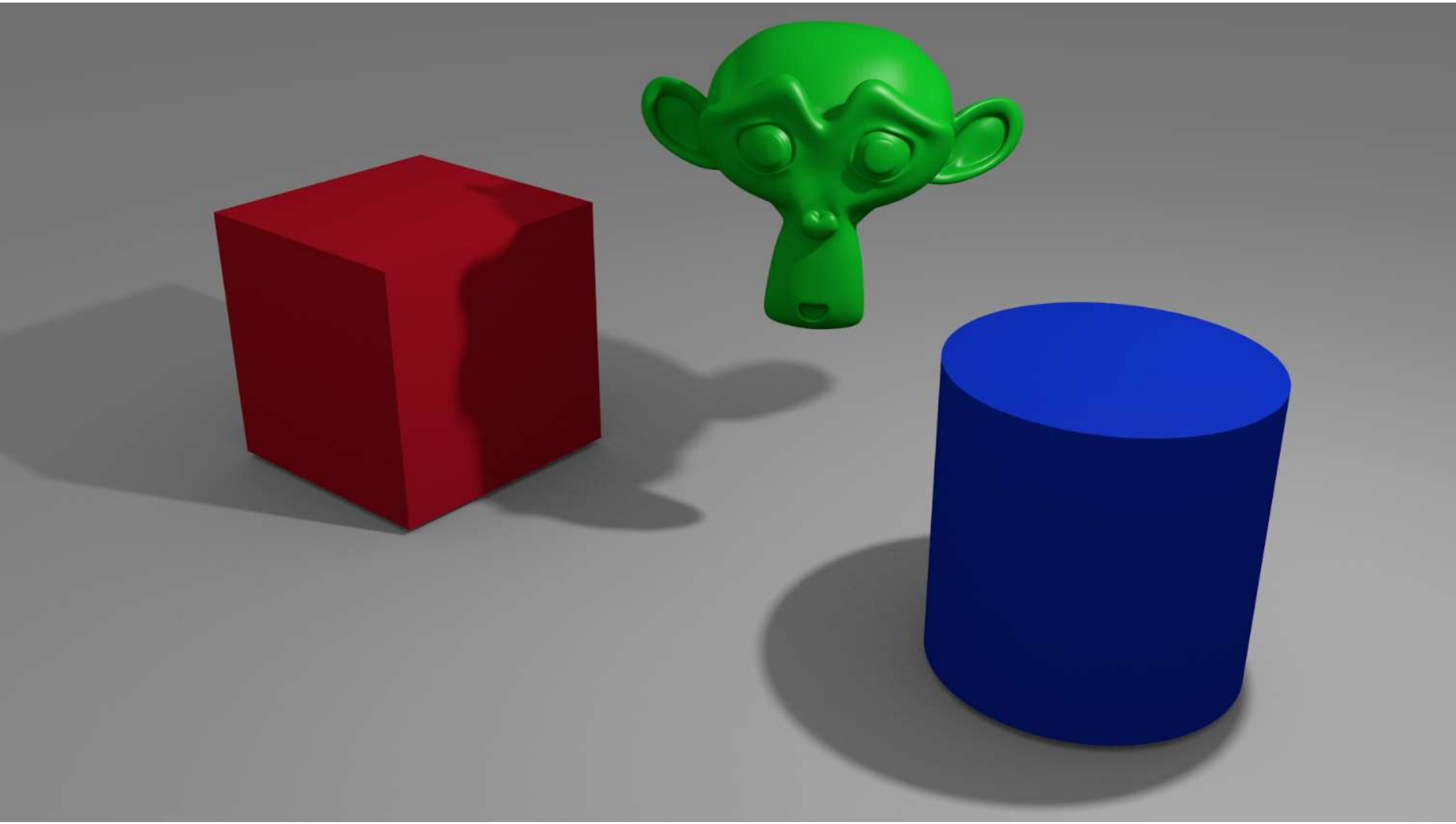
- Si apropem oclusor i receptor...

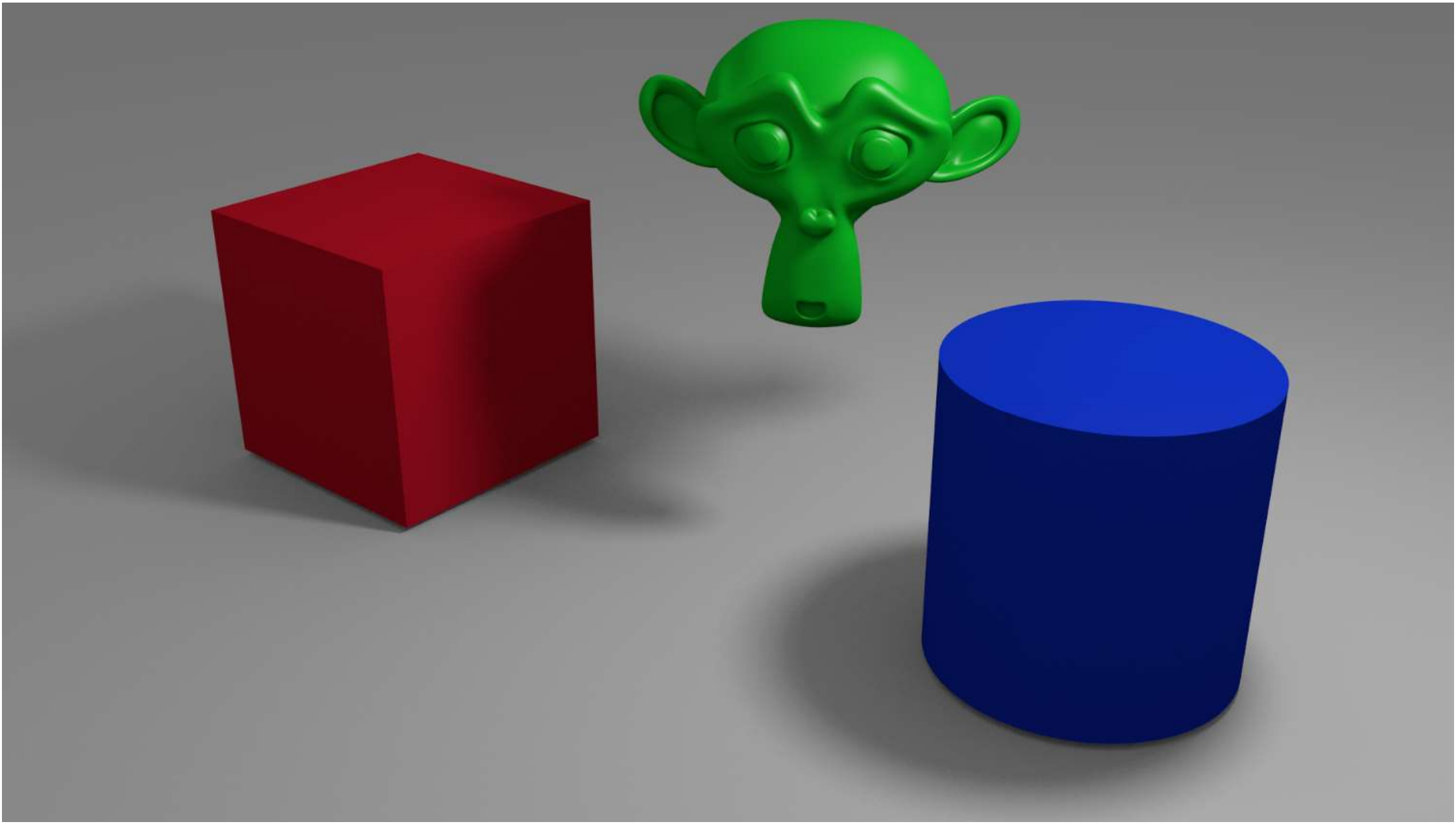
Propietats



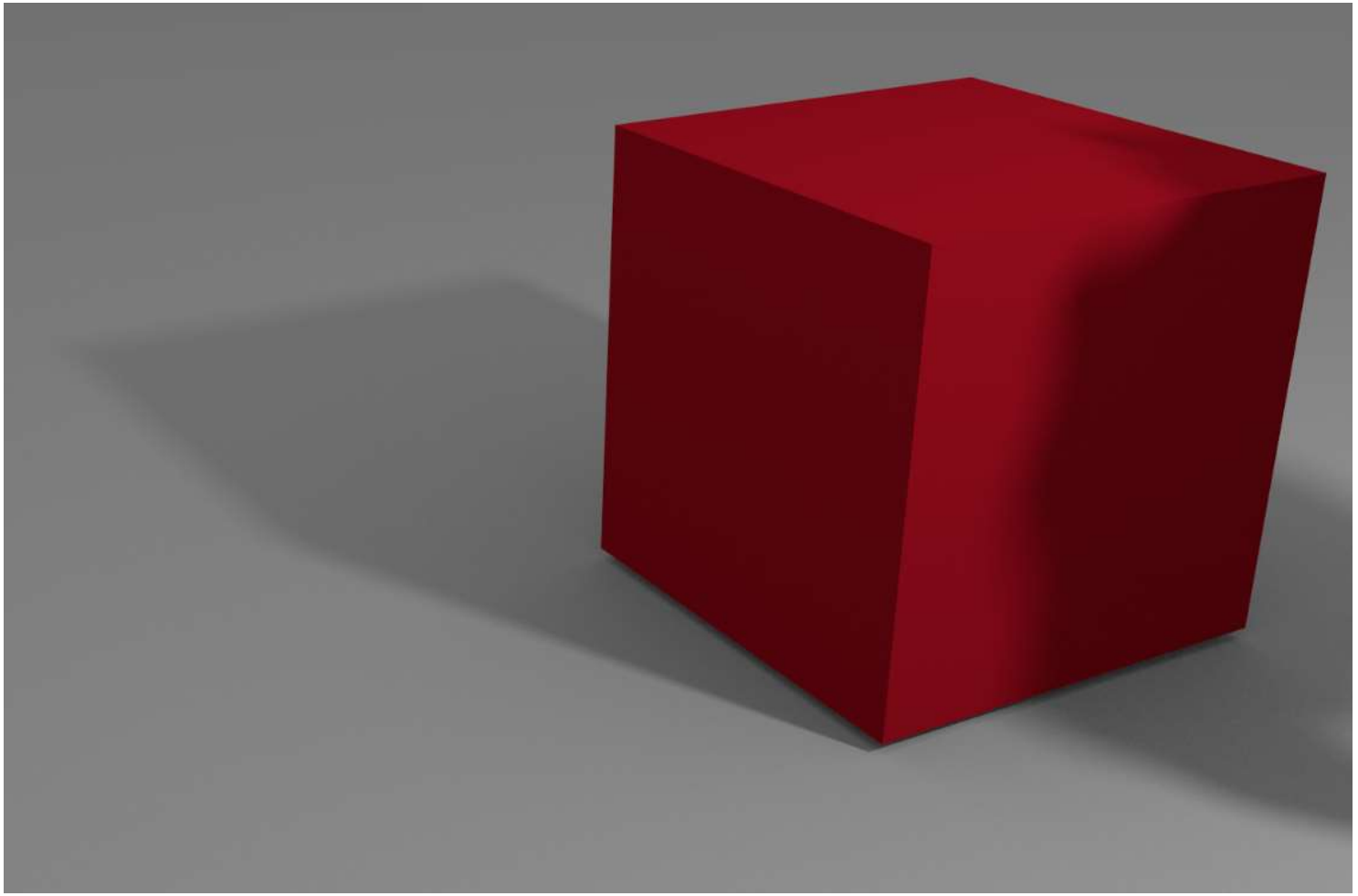
- Si la font de llum és puntual → no hi ha penombra
- Si augmenta la mida de la font de llum...
  - Augmenta la penombra
  - Disminueix la umbra (pot ser nul·la)
- Si apropem oclusor i receptor...
  - Disminueix la penombra





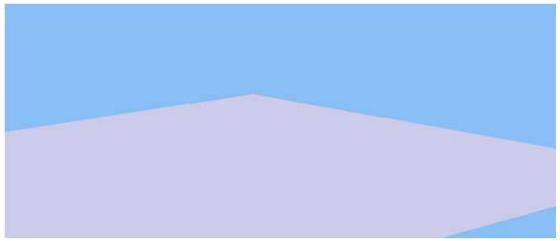


Propietats

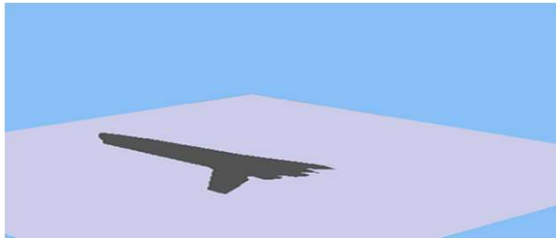


Ombres per projecció (un pla)

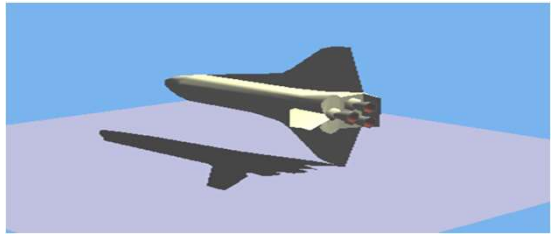
// 1. Dibuixar **receptor**



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



// 3. Dibuixar **oclusor**

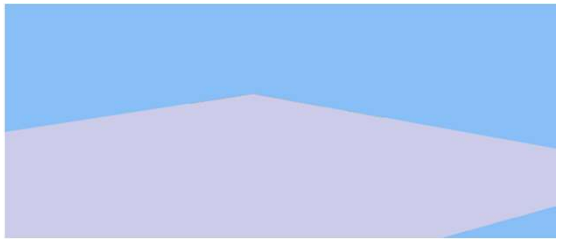


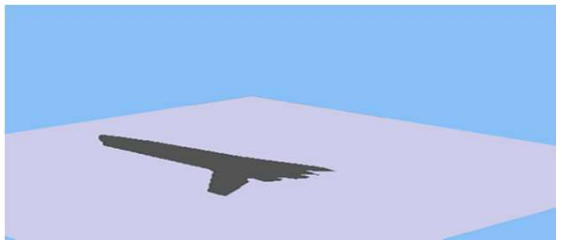
Ombres per projecció (un pla)

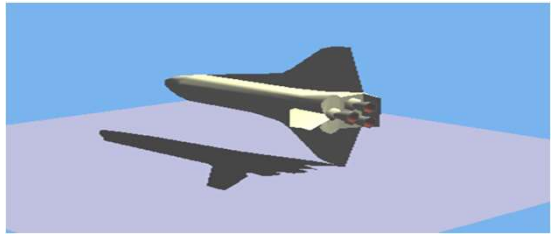
// 1. Dibuixar **receptor**
dibuixa(receptor)

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

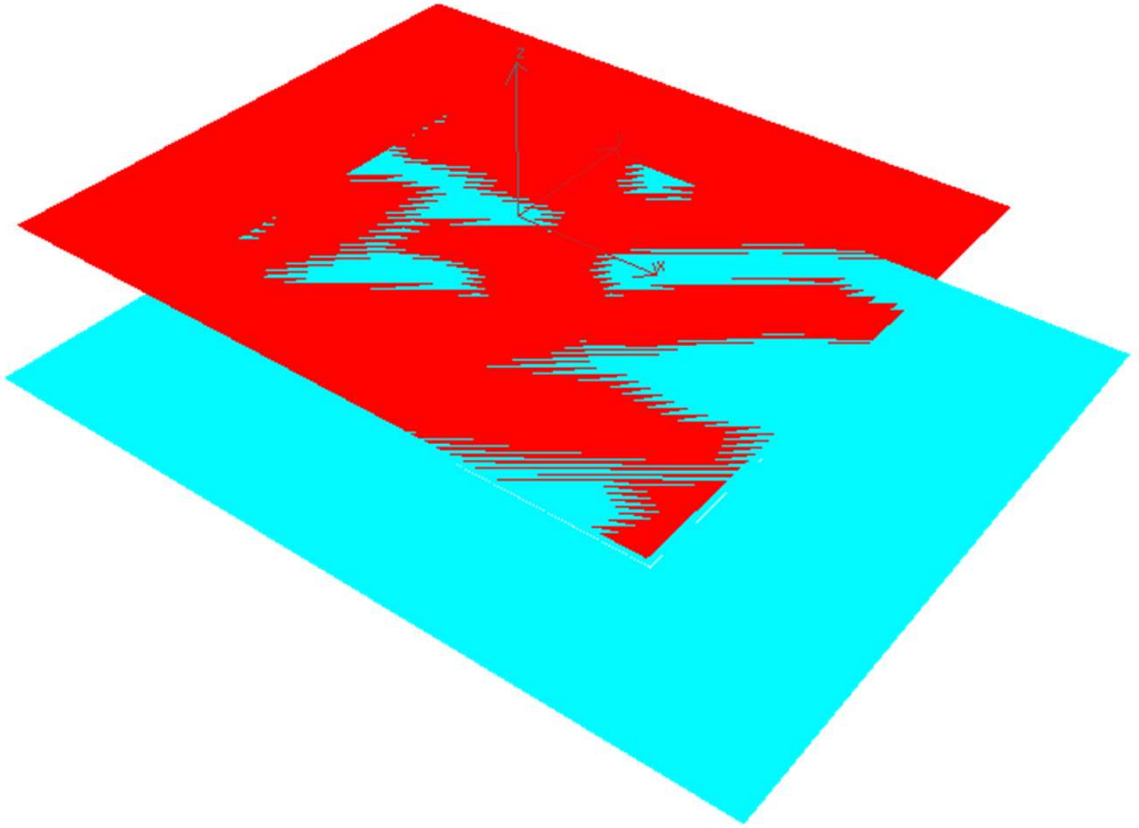
// 3. Dibuixar **oclusor**
glEnable (GL_LIGHTING);
glEnable (GL_DEPTH_TEST);
dibuixa (emissor);







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

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

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

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

on

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

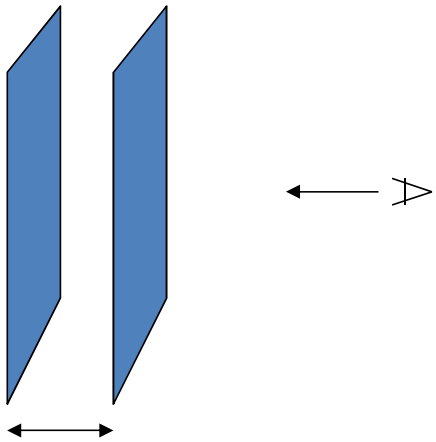
r = valor més petit tal que garantitza un offset > 0

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

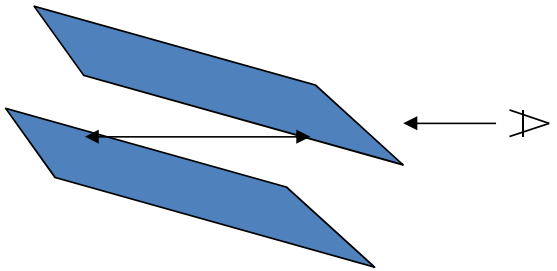
→ El paràmetre *units* permet introduir un **offset constant**

Evitar problemes de z-fighting

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



Offset = r·units (dz=0)



Offset = 3 · factor + r·units (dz=3)

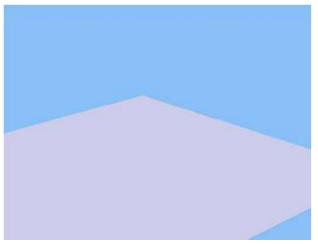
Evitar problemes de z-fighting

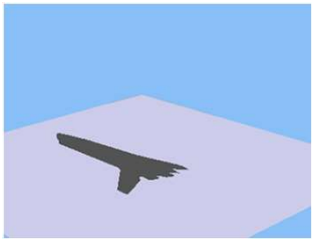
Valors típics: glPolygonOffset(*1*, *1*);

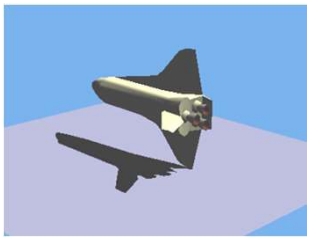
Offset positiu → increment de la z (en window coordinates) → es calcula la z com si estigués més lluny

Ombres per projecció (un pla)

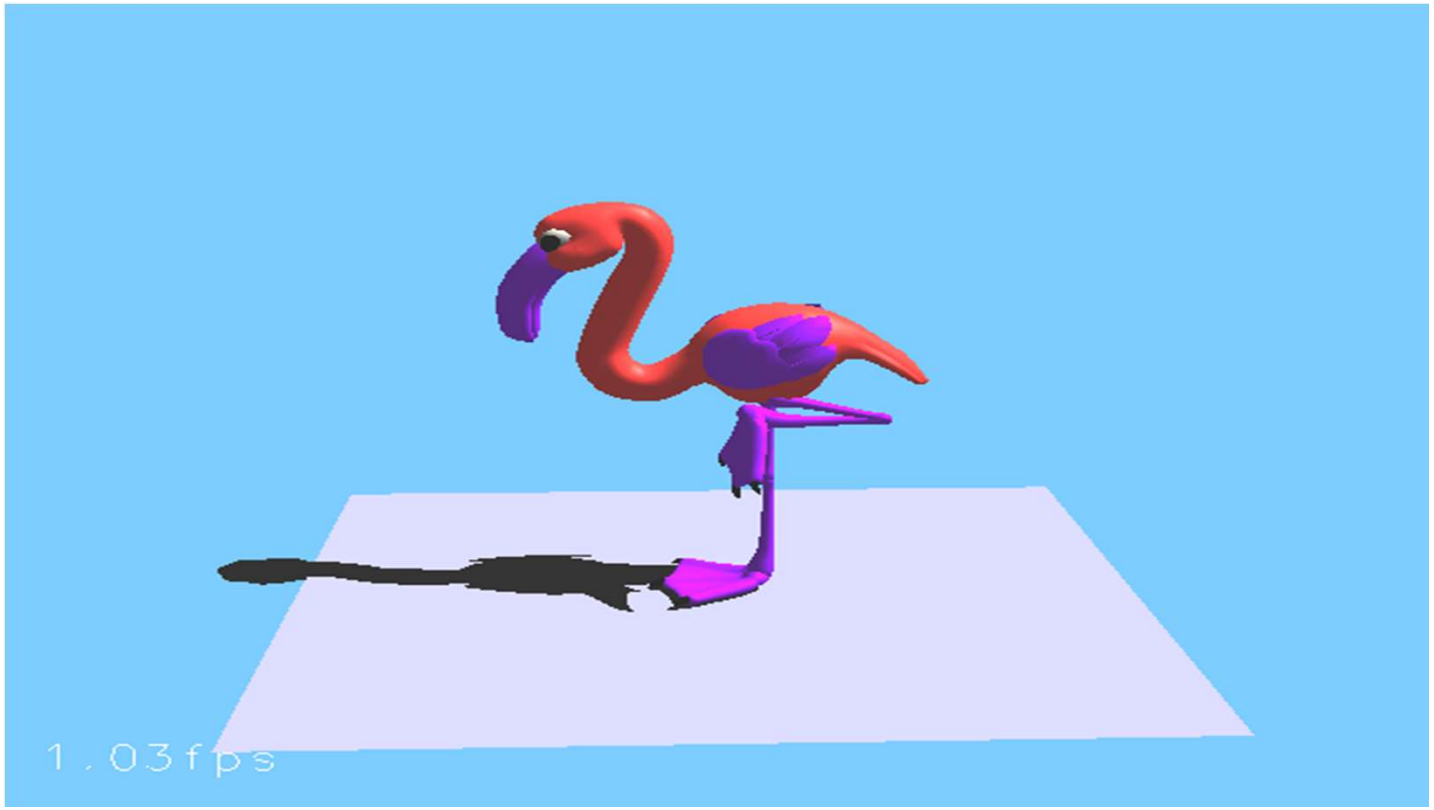
```
void CastShadows::paintGL(…) {
  // 1. Dibuixar receptor
  phongShader->bind();
  drawReceiver();

// 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();
}
```



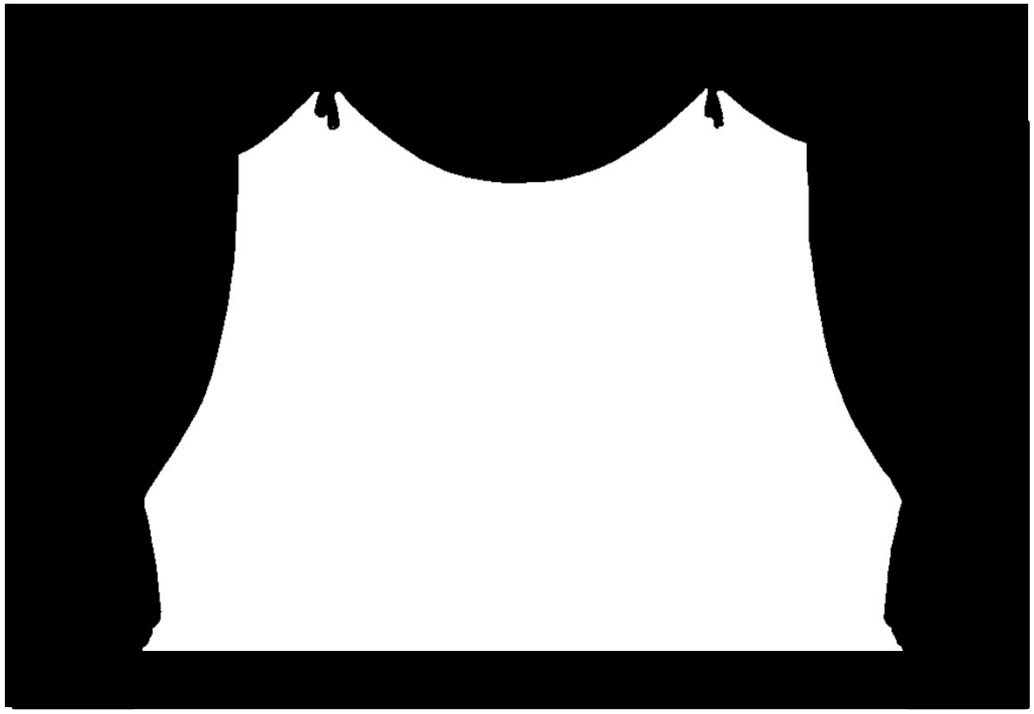




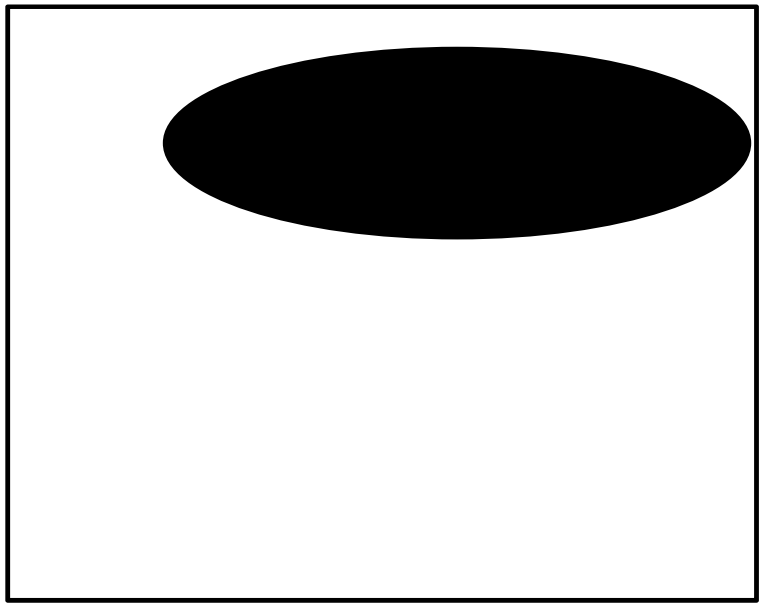
Ombres per projecció (un pla)



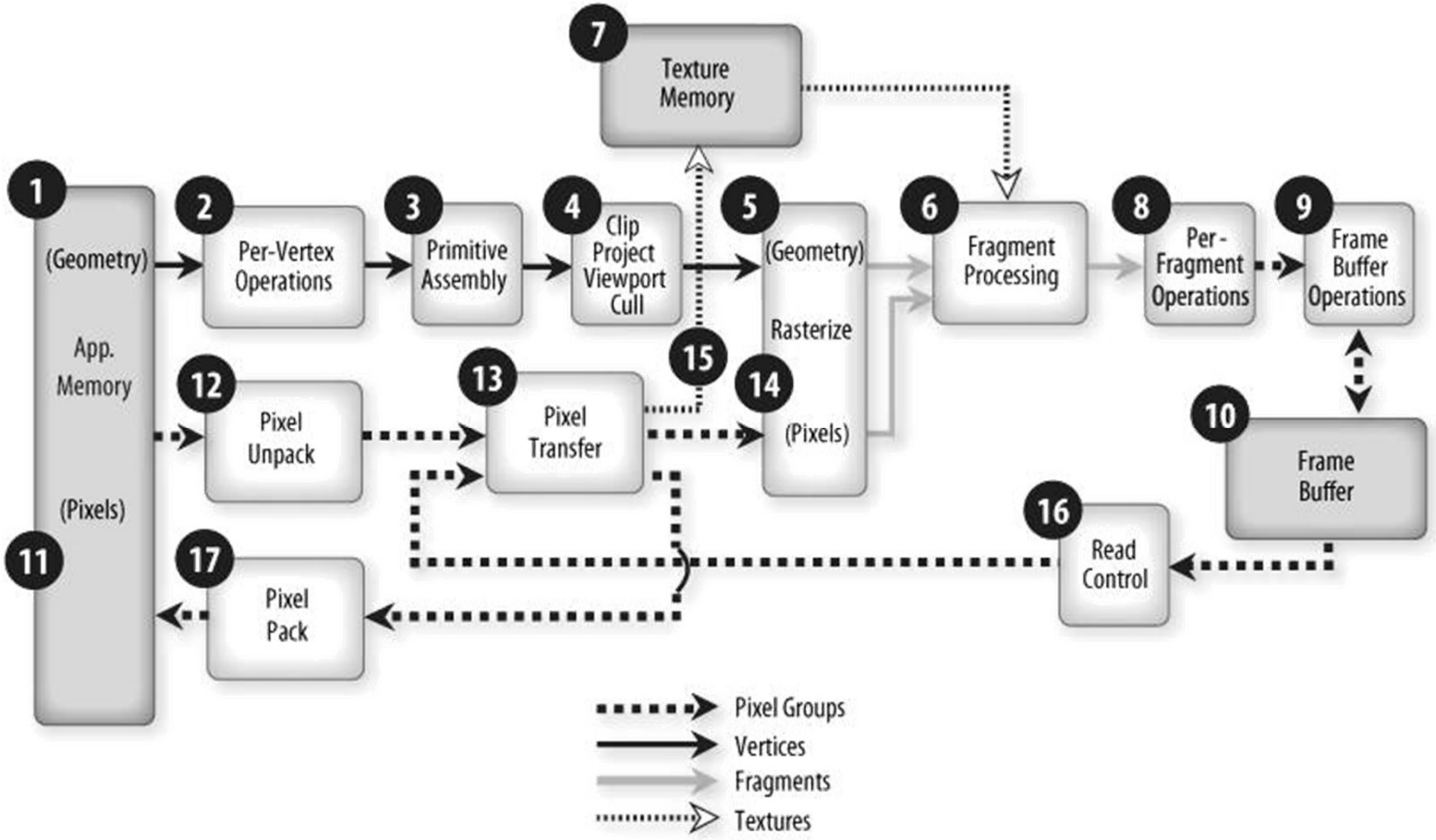
Stencil buffer



Stencil buffer



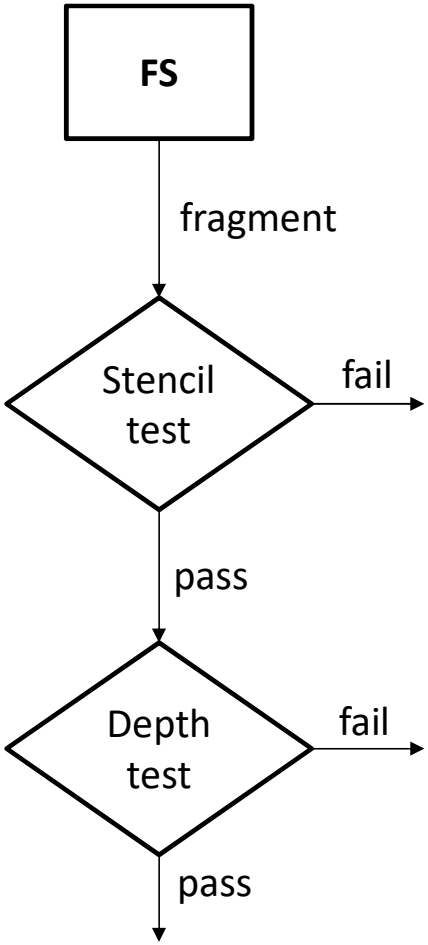
Pipeline OpenGL



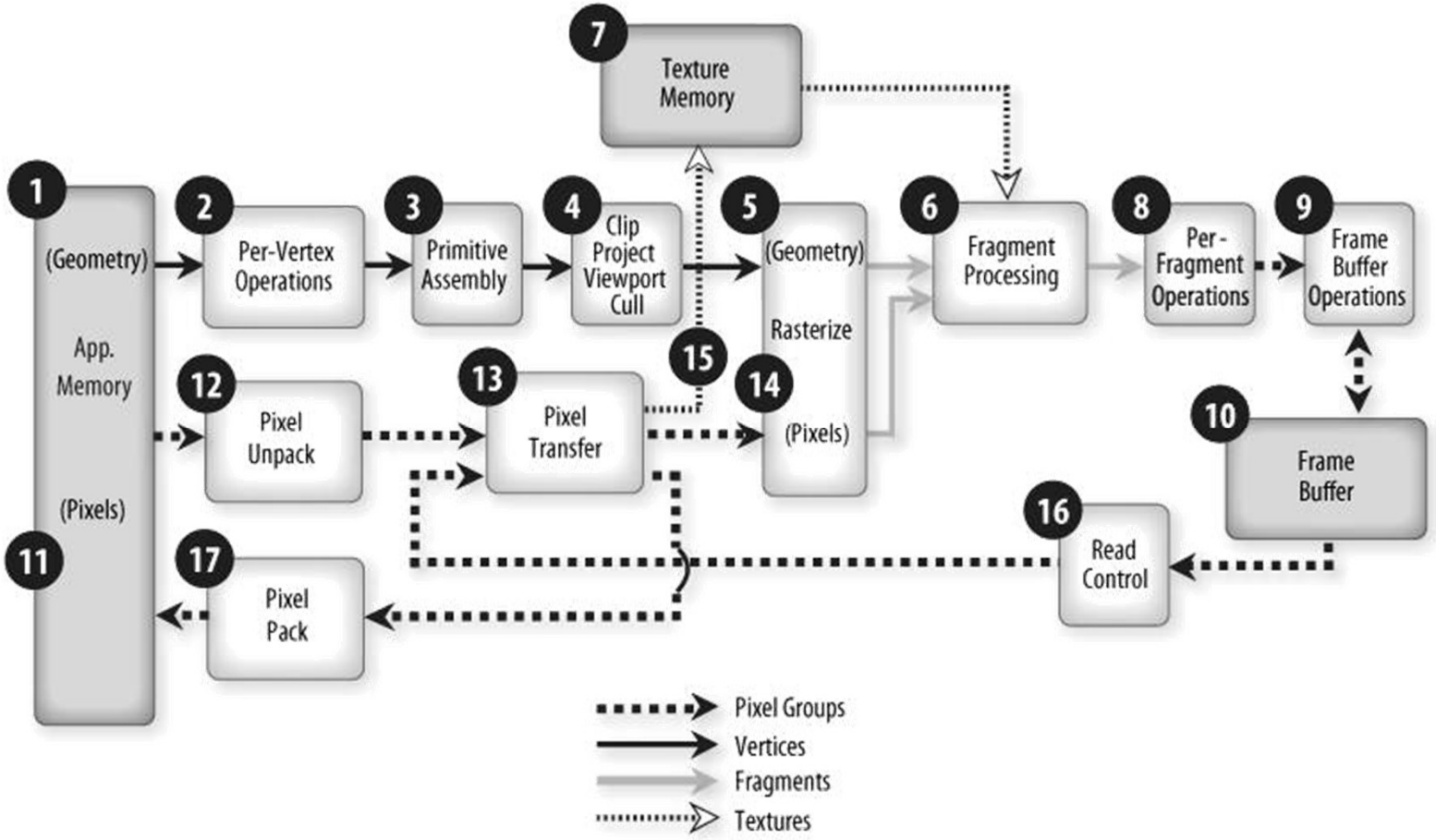
Pipeline OpenGL

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

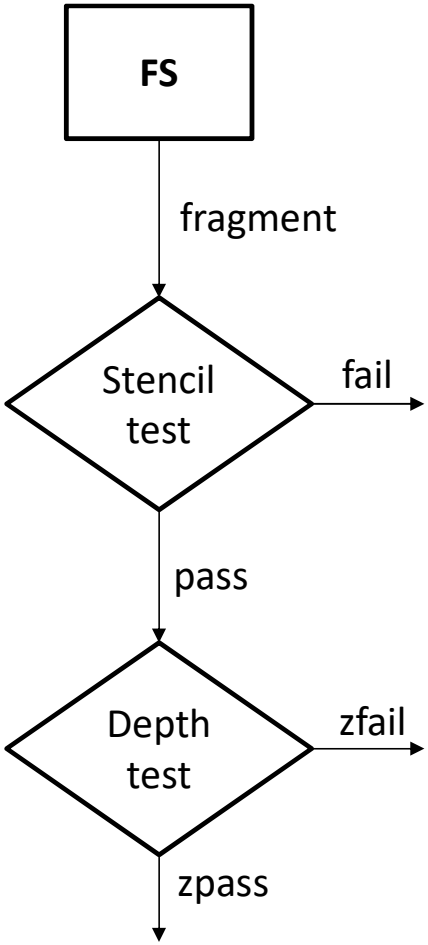
- Establir el test de comparació:
  - glEnable(GL_STENCIL_TEST);
  - **glStencilFunc**(comparació, valorRef, mask)
    - Comparació pot ser: GL_NEVER, GL_ALWAYS, GL_LESS…
    - Ex: GL_LESS: (valorRef & mask) < (valorStencil &mask)

- 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 refèrencia)

Stencil buffer

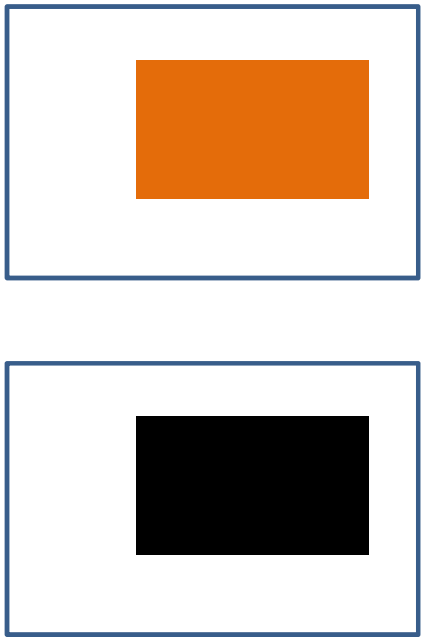
- Establir el test de comparació:
  - glEnable(GL_STENCIL_TEST);
  - **glStencilFunc**(comparació, valorRef, mask)
    - Comparació pot ser: GL_NEVER, GL_ALWAYS, GL_LESS…
    - Ex: GL_LESS: (valorRef & mask) < (valorStencil &mask)

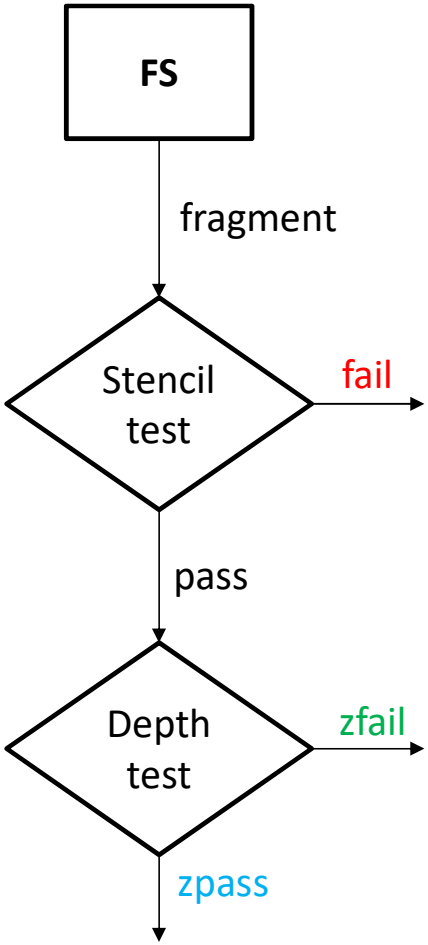
- Operacions a fer a stencil buffer segons el resultat del test:
  - **glStencilOp**(fail, zfail, zpass)
    - fail -> op. a fer quan el fragment no passa el test de stencil
    - Zfail -> op. a fer quan passa stencil, pero no passa z-buffer
    - Zpass -> op. a fer quan passa stencil i passa z-buffer
  - Cadascú dels paràmetres anteriors pot ser:
    - GL_KEEP, GL_ZERO, GL_INCR, GL_DECR, GL_INVERT
    - GL_REPLACE (usa valor refèrencia)



Stencil buffer

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

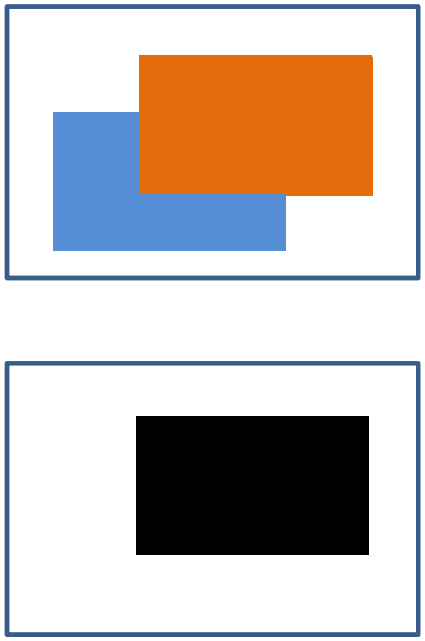


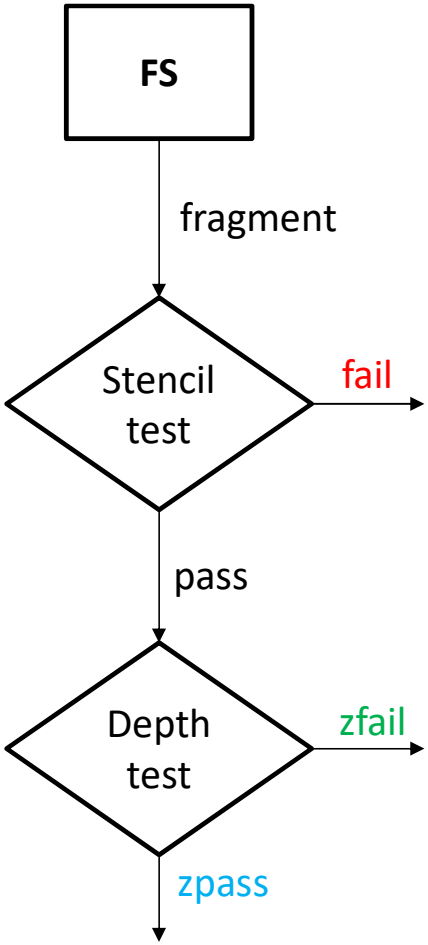


Stencil buffer

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

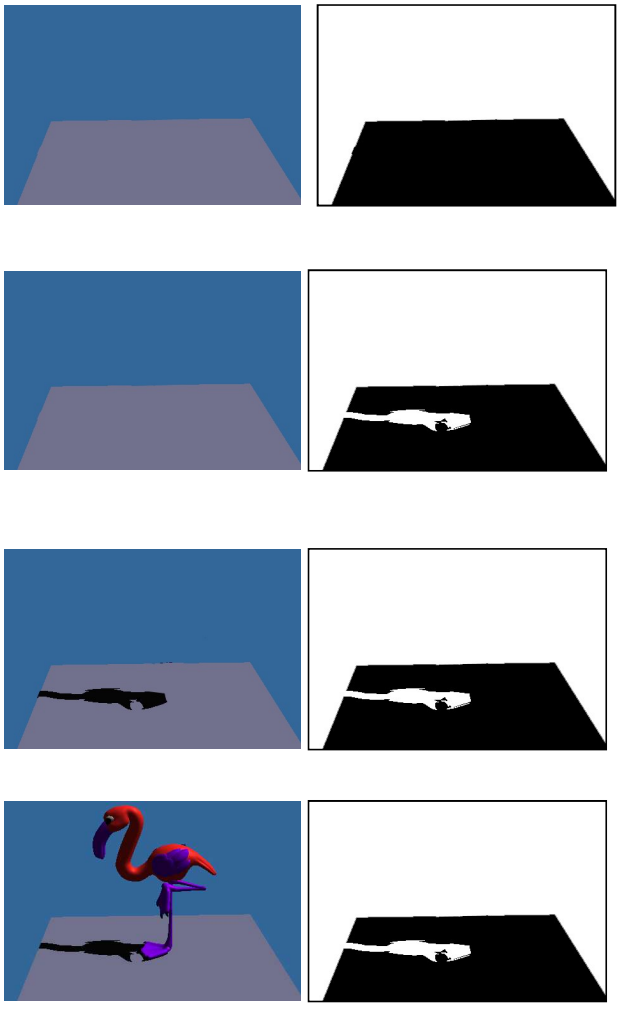
```
glStencilFunc(GL_EQUAL, 0, 255);
glStencilOp(GL_KEEP,  GL_KEEP,  GL_KEEP);
drawQuad();
```





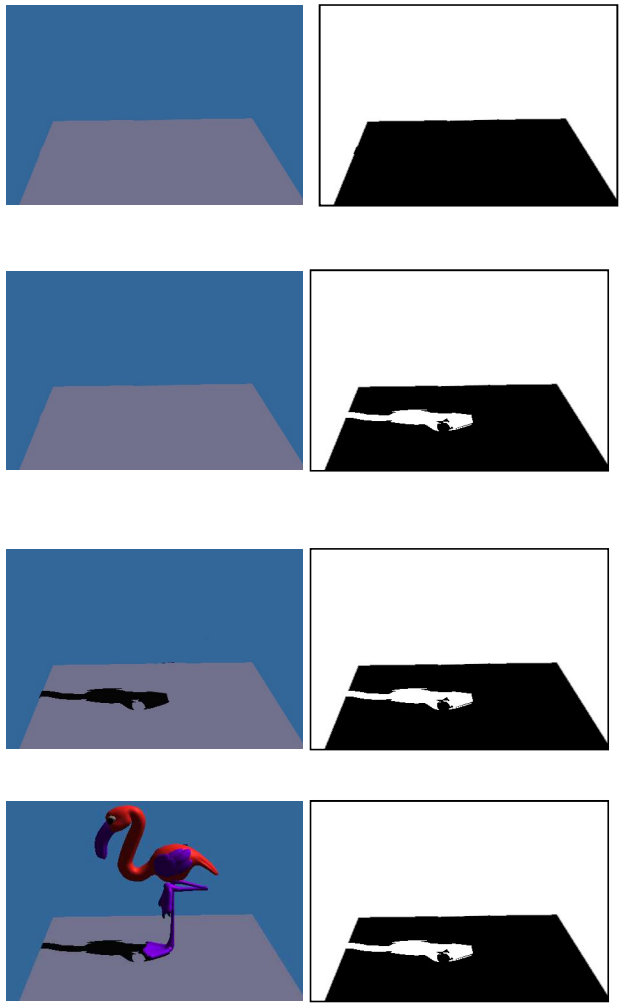
Ombres per projecció (amb stencil)

Targets	
Color, Z, Stencil	// 1. Dibuixa el **receptor** al color buffer i al stencil buffer
Stencil	// 2. Dibuixa **oclusor** per netejar l'stencil a les zones a l'ombra
Color, Z	// 3. Dibuixa la part fosca del **receptor**
Color, Z	// 4. Dibuixa l'**oclusor**



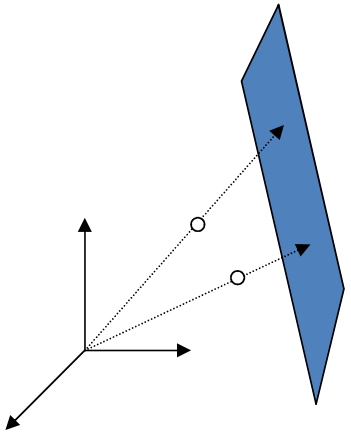
Ombres per projecció (amb stencil)

```
// 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

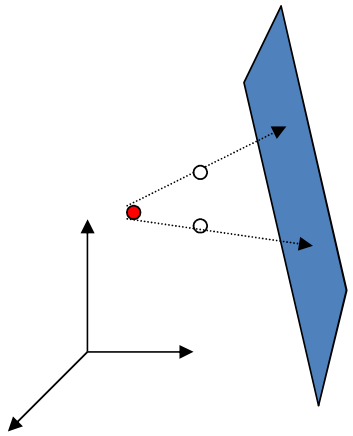
- Donats els coeficients (a,b,c,d) d'un pla, la matriu de projecció respecte l'origen és:



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

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:

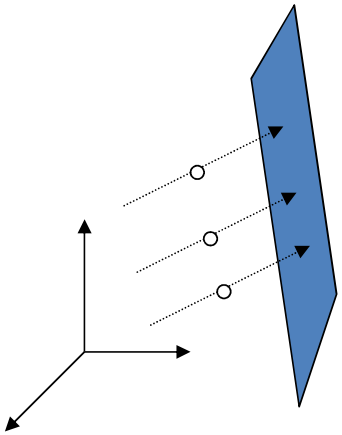


$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -(d+ax+by+cz) & 0 & 0 & 0 \\ 0 & -(d+ax+by+cz) & 0 & 0 \\ 0 & 0 & -(d+ax+by+cz) & 0 \\ a & b & c & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 &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

$$\begin{bmatrix} -d-by-cz & xb & xc & xd \\ ya & -d-ax-cz & yc & yd \\ za & zb & -d-ax-by & zd \\ a & b & c & -ax-by-cz \end{bmatrix}$$

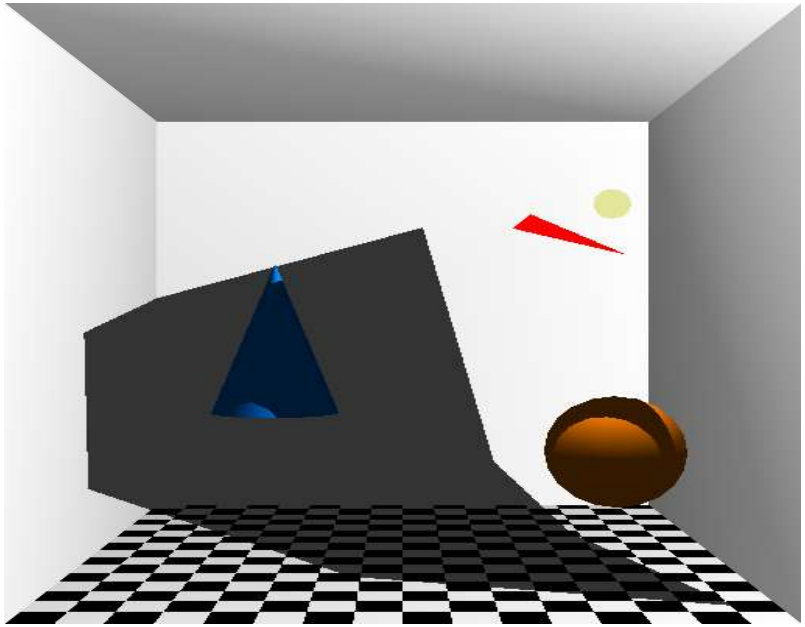
Projecció en la direcció (x,y,z)

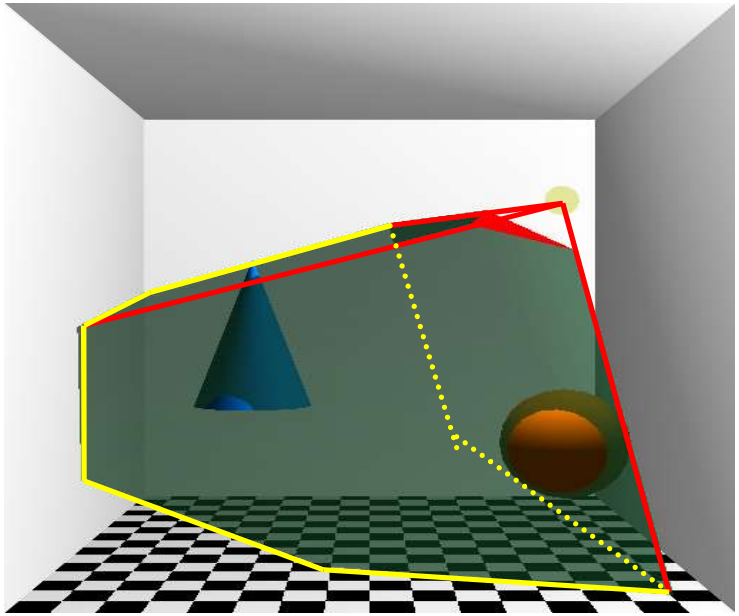
- Donats els coeficients (a,b,c,d) d'un pla, la matriu de projecció en la direcció del vector (x,y,z) és:



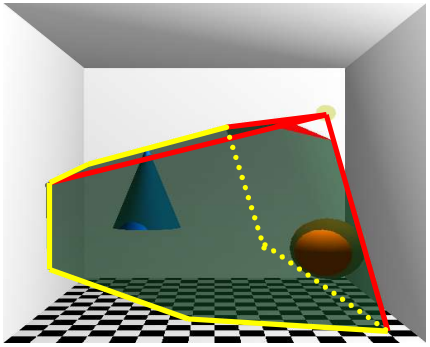
$$\begin{bmatrix} by+cz & -bx & -cx & -dx \\ -ay & ax+cz & -cy & -dy \\ -az & -bz & ax+by & -dz \\ 0 & 0 & 0 & ax+by+cz \end{bmatrix}$$

Volums d'ombra

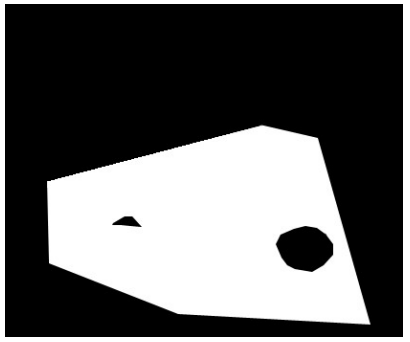


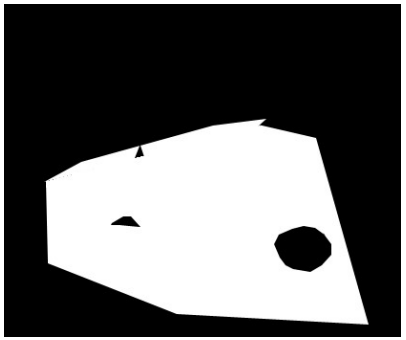


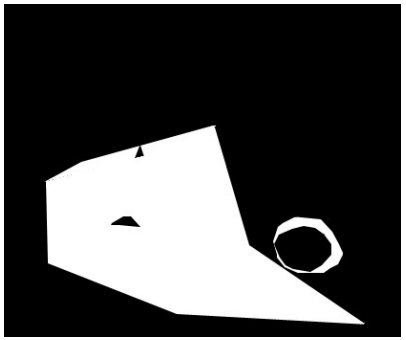
Volums d'ombra (1/2)



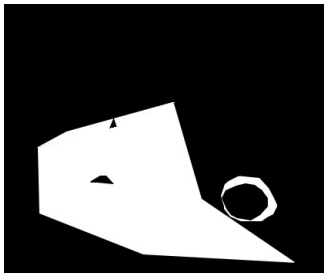
```
// 1. Dibuixa l'escena al z-buffer
glColorMask(GL_FALSE, ..., GL_FALSE);
dibuixa(escena);

// 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);
```





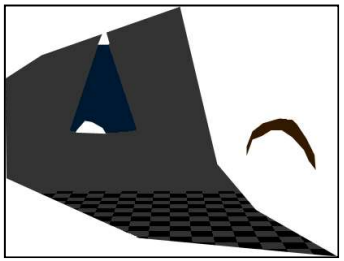


Volums d'ombra (2/2)



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

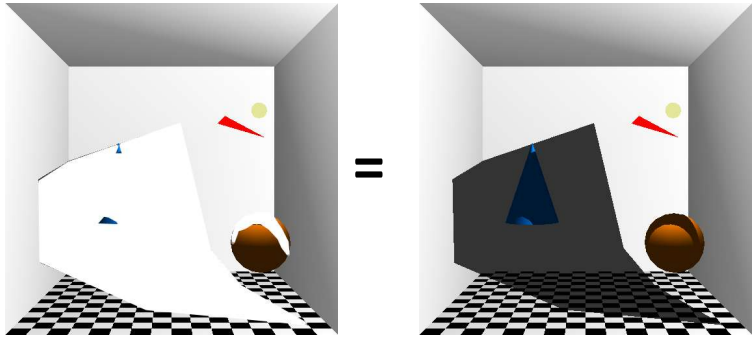
glDepthMask(GL_TRUE);
glColorMask(GL_TRUE, ... , GL_TRUE);
glCullFace(GL_BACK);
glDepthFunc(GL_LEQUAL);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
glStencilFunc(GL_EQUAL, 1, 1);
glDisable(GL_LIGHTING);
dibuixa(escena);



+

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

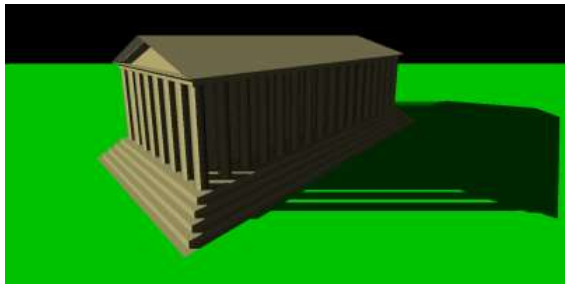
glStencilFunc(GL_EQUAL, 0, 1);
glEnable(GL_LIGHTING);
dibuixa(escena);

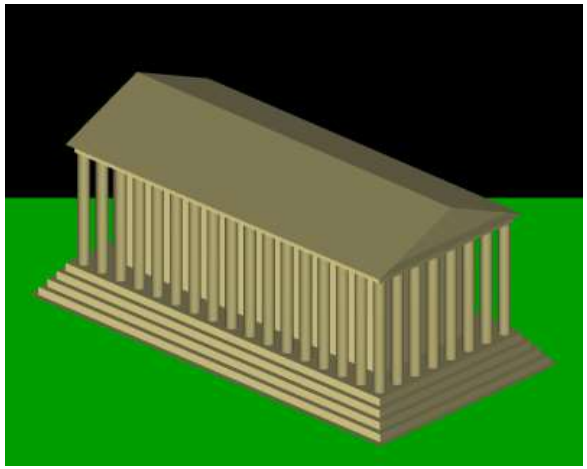


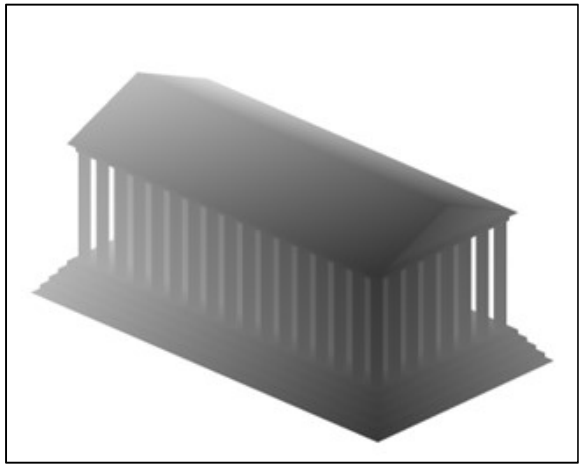
// 6. Restaura l'estat inicial

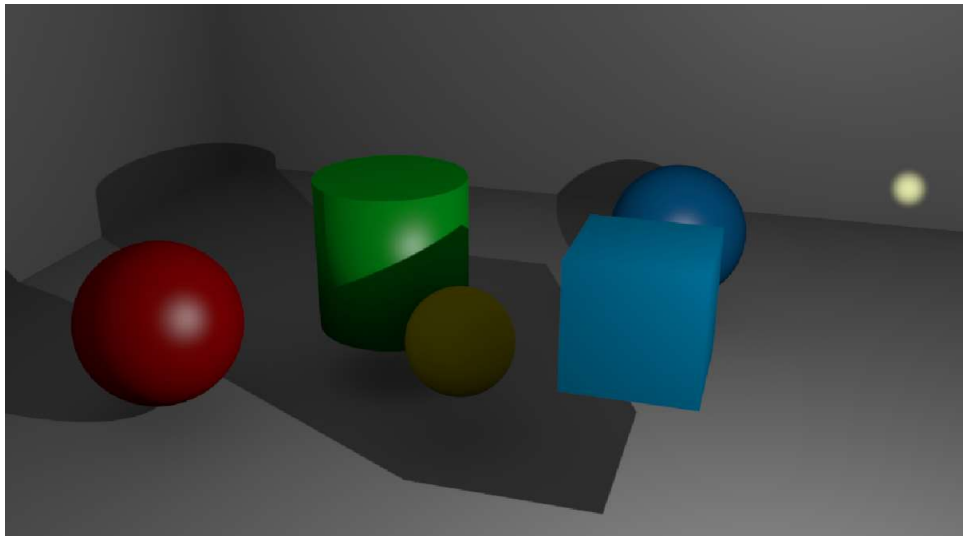
glDepthFunc(GL_LESS);
glDisable(GL_STENCIL_TEST);

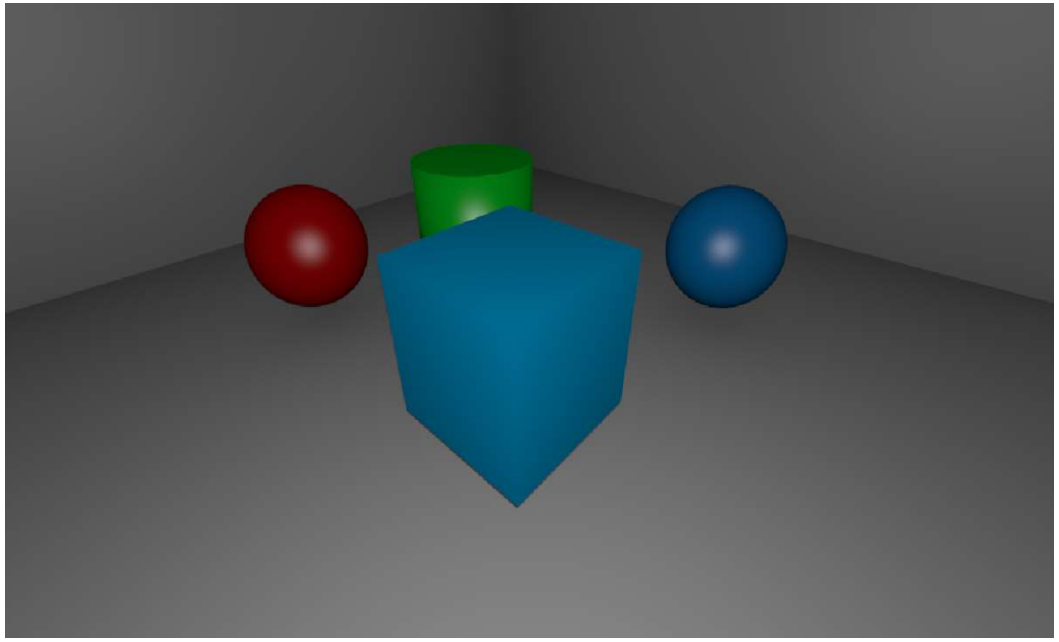
Shadow mapping

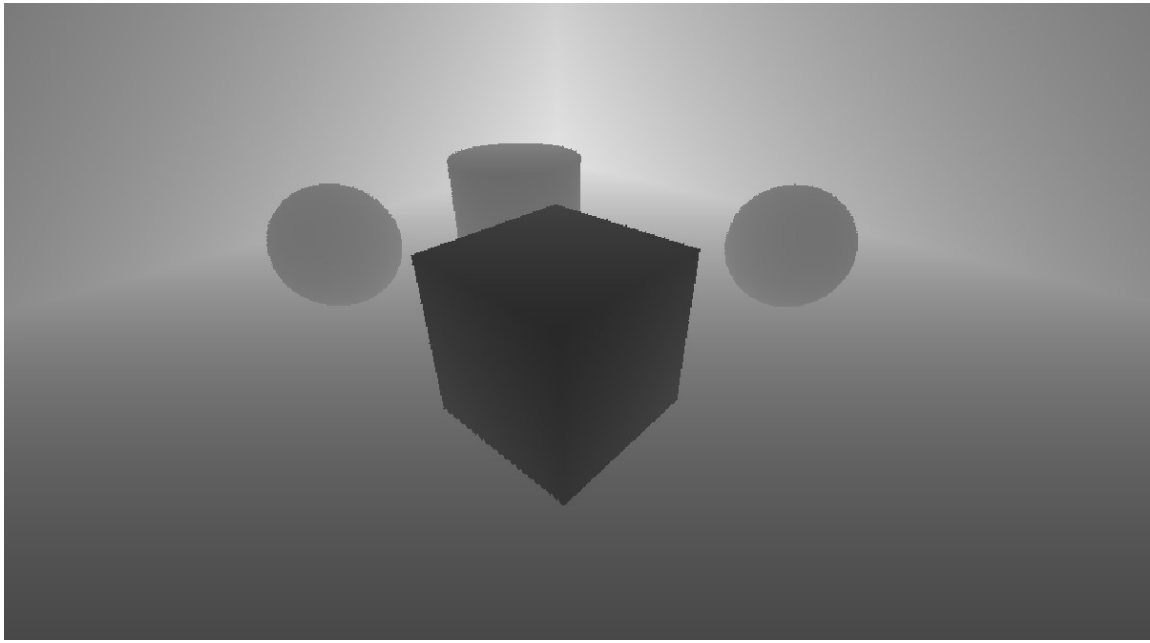




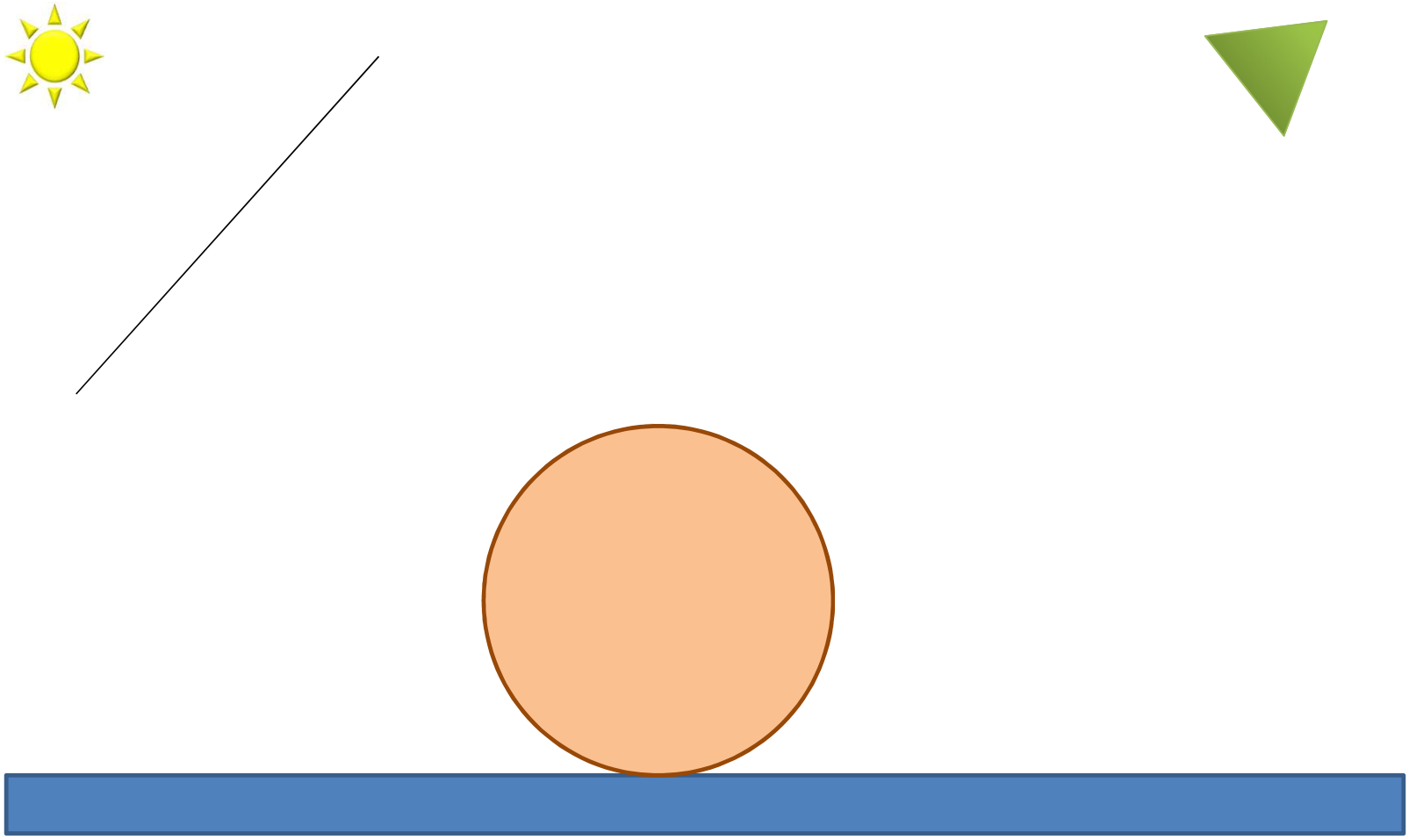








Shadow mapping



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