Sistemes de coordenades

$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$ Podem homogeneïtzar les matrius (4×4) afegint un component homogeni (w). Ho podem fer la translació as matrius de 3×3.

Punts

Object space

↓ Transformació del model

World space

↓ Transformació de la càmera (visualització)

Eye space

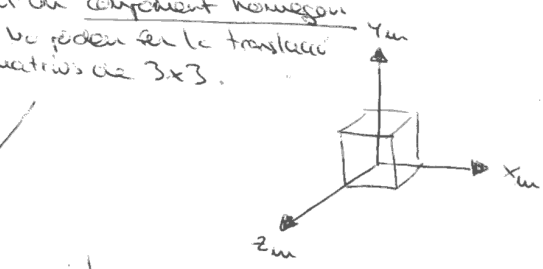
↓ Transformació de projecció.

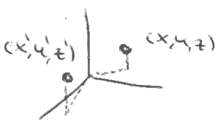
Clip space

Transformacions geomètriques:

- Translació

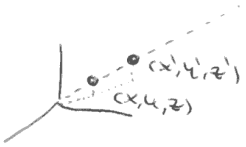
$$\begin{pmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

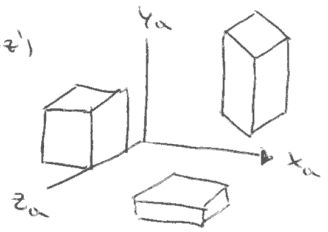




- Escalat

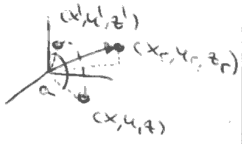
$$\begin{pmatrix} x_s & 0 & 0 & 0 \\ 0 & y_s & 0 & 0 \\ 0 & 0 & z_s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$





- Rotació

$$\begin{pmatrix} x_r^2 d + c & x_r y_r d - z_r s & x_r z_r d + y_r s & 0 \\ x_r y_r d + z_r s & y_r^2 d + c & y_r z_r d - x_r s & 0 \\ x_r z_r d - y_r s & y_r z_r d + x_r s & z_r^2 d + c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$



$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c & -s & 0 \\ 0 & s & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \xrightarrow{x} \begin{pmatrix} c & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ -s & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \xrightarrow{y} \begin{pmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \xrightarrow{z}$$

$s = \sin(a) \quad c = \cos(a) \quad d = 1 - \cos(a)$

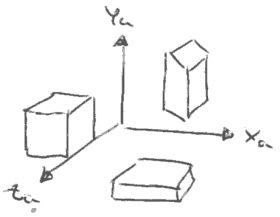
Transformem el model aplicant transformacions $(T \mid S \mid R)^*$

Generem una matriu de transformació a partir de les matrius de transformació:

$$M = M_n * \cdots * M_1 \qquad M \begin{pmatrix} x_m \\ y_m \\ z_m \\ 1 \end{pmatrix} = \begin{pmatrix} x_a \\ y_a \\ z_a \\ 1 \end{pmatrix}$$

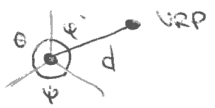
Transformem l'escena:

- $\text{gluLookAt}(eye, target, up)$



- $T(0,0,-d) * R_z(-\Psi) * R_x(\Theta) * R_y(-\Psi) * T(-VRP)$

$$V \begin{pmatrix} x_a \\ y_a \\ z_a \\ 1 \end{pmatrix} = \begin{pmatrix} x_e \\ y_e \\ z_e \\ 1 \end{pmatrix}$$

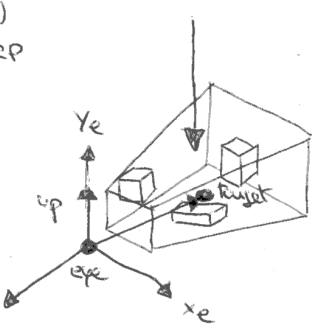


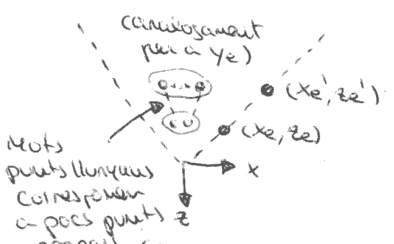
Projectem l'escena:

- $\text{gluPerspective}(fov, aspect, z_n, z_f)$

$$\begin{pmatrix} \frac{\cot an \frac{fov}{2}}{aspect} & 0 & 0 & 0 \\ 0 & \cot an \frac{fov}{2} & 0 & 0 \\ 0 & 0 & \frac{z_n + z_f}{z_n - z_f} & \frac{2 z_n z_f}{z_n - z_f} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$\cot an \theta = \frac{\cos \theta}{\sin \theta}$$





$$\frac{x_e}{z_e} = \frac{x_e'}{z_e'} \qquad \frac{y_e}{z_e} = \frac{y_e'}{z_e'}$$

$$P \begin{pmatrix} x_e \\ y_e \\ z_e \\ 1 \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \\ z_c \\ -z_e \end{pmatrix}$$

$-w_c \leq x_c, y_c, z_c \leq w_c$

propers (valors dintre uns més petits) 1

Clip space

↓ Divisió de perspectiva

Normalized Device Space (NDS)

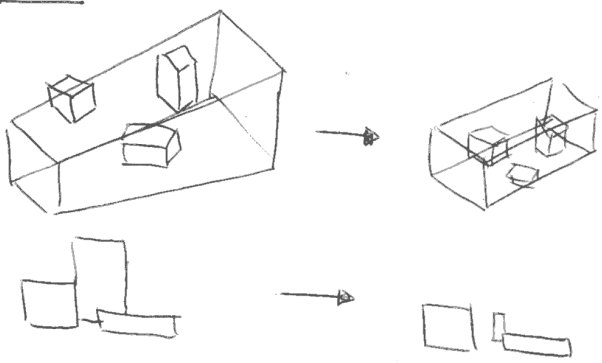
↓ Transformació de finestra (viewport) i de profunditat

Window space

Apliquem la divisió de perspectiva:

$$\begin{pmatrix} \frac{x_c}{w_c} \\ \frac{y_c}{w_c} \\ \frac{z_c}{w_c} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix}$$

$$-1 \le x_n, y_n, z_n \le 1$$
$$z_e = z_N \Rightarrow z_n = -1$$
$$z_e = z_F \Rightarrow z_n = 1$$



(veiem objectes llunyans més petits)

Ho adaptem a la finestra:

$$\begin{pmatrix} \frac{x_n + 1}{2} w \\ \frac{y_n + 1}{2} h \\ \frac{z_n + 1}{2} \end{pmatrix} = \begin{pmatrix} x_d \\ y_d \\ z_d \end{pmatrix}$$

- Si Viewport (0, 0, w, h)
- Si Depth Range (0, 1)

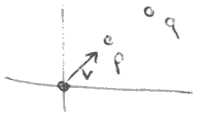
$$0 \le x_d \le w$$
$$0 \le y_d \le h$$
$$0 \le z_d \le 1$$
$$z_e = z_N \Rightarrow z_d = 0$$
$$z_e = z_F \Rightarrow z_d = 1$$

$$w_d = \frac{1}{w_c} = -\frac{1}{z_e}$$

Vectors (d. teràcia entre dos punts)

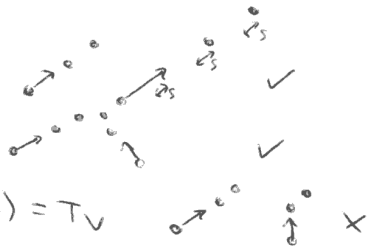
En comptes de $\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$ utilitzem $\begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$ per a evitar traslladar el vector.

Si que l'escalem i el girem.



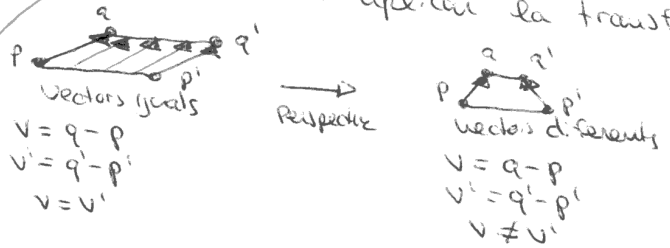
$v = q - p$

$$Sq - Sp = S(q-p) = Sv$$
$$Rq - Rp = R(q-p) = Rv$$
$$Tq - Tp = (q-p) \neq T(q-p) = Tv$$



Object space
↓ M
World space
↓ V
Eye space
↓
...

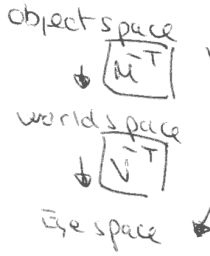
No li podem aplicar la transformació de projecció:



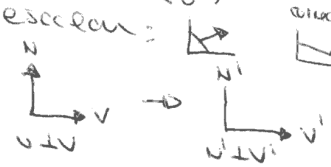
$v = q - p$
$v' = q' - p'$
$v = v'$

$v = q - p$
$v' = q' - p'$
$v \neq v'$

Vectors normals



utilitzem $\begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$ per la mateixa raó, però no podem escalar:



(escalat no conforme)

$$N \perp V \Leftrightarrow N \cdot V = 0 \Leftrightarrow N^T V = 0 \Leftrightarrow N^T I V = 0 \Leftrightarrow$$
$$N^T (M'^{-1} M) V = 0 \Leftrightarrow N^T M'^{-1} (MV) = 0 \Leftrightarrow N^T M^{-T} MV = 0 \Leftrightarrow$$
$$(M^{-T} N)^T MV = 0 \Leftrightarrow V'^T V' = 0 \Leftrightarrow V' \cdot V' = 0 \Leftrightarrow N' \perp V'$$

$N = (VM)^{-T} = (VM)^{-1^T} = (M'^{-1} V'^{-1})^T = (V'^T M^{-T})$ (com que no traslladem utilitzem $\begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$ i fem servir N)

Procés de renderitzat

Generació de primitives

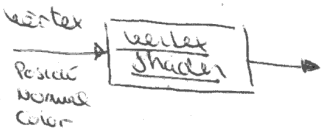
Tipus de primitives:

- Punts
- Línies
- Polígons
- Tira de strip
- Ventall strip

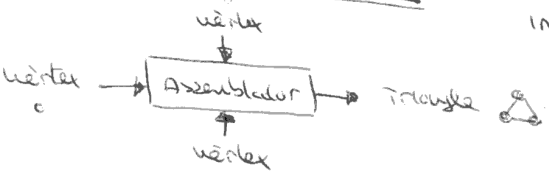
```
glBegin(GL_POLYGON)
glVertex3f(x,y,z)
glNormal3f(a,b,c)
....
glEnd()
```

```
glGenVertexArrays(1, &VAO)
glBindVertexArray(VAO)
glGenBuffers(1, &VBO)
glBindBuffer(GL_ARRAY_BUFFER, VBO)
glBufferData(...)
glVertexAttribPointer(...)
...
glDrawArrays(GL_TRIANGLES, 0, n)
  o bé
glDrawElements(GL_TRIANGLES, nt, GL_TRIANGLES / GL_TRIANGLE_STRIP, INDICES)
```

Transformació de vèrtexs

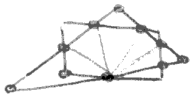


Assemblatge de primitives



Processament de primitives

- Clipping (retallat)



Interpolem els atributs dels nous vèrtexs.

- Viewport & depth transformation

$$x = \frac{x+1}{2} w$$

$$y = \frac{y+1}{2} h$$

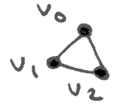
$$z = \frac{z+1}{2}$$

- Divisió de perspectiva

$$x = x/w$$
$$y = y/w$$
$$z = z/w$$

- Backface culling

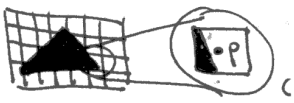
```
glEnable(GL_CULL_FACE)
```



Calcula la normal als vèrtexs v_0, v_1, v_2 (l'ordre dels vèrtexs importa).

Rasterització

Generació de fragments.
Interpolació dels atributs.



p ∈ Triangle? ■ : □ (centre)

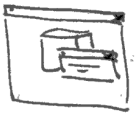
Processament de fragments

Càlcul del color i la profunditat.

Operacions als fragments

- Pixel ownership test

El frame buffer és extern a OpenGL. Cal comprovar que un pixel sigui part del context actiu.



- Blending

Per a objectes translúcids. Barreja els colors.

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

- Stencil test

Retallar de màscares

(ombres, reflexos, etc.)

```
glEnable(GL_STENCIL_TEST)
glStencilFunc(...)
glStencilOp(...)
```

- Depth test

Comprovar la z en comptes de l'ordre ~~de dibuix~~ en què es pinten els objectes.

```
glEnable(GL_DEPTH_TEST)
glDepthFunc(...)
```

Operacions al frame buffer

- Escriu al buffer de color

  glColorMask(GL_TRUE / GL_FALSE, ..., ..., ...) (RGBA)

- Escriu al buffer de profunditat

  glDepthMask(GL_TRUE / GL_FALSE)

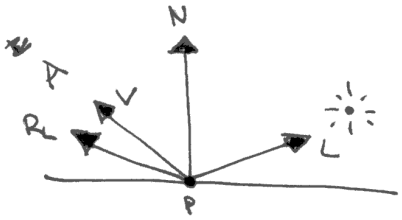
Model de Phong

Reflexió especular — Terme especular

$$I(p) = k_a I_a + \sum_L k_d I_L (N \cdot L) + \sum_L k_s I_L (R_L \cdot V)^n$$

Llum ambient

Si $N \cdot L > 0$ ) $N \cdot L > 0$) $N \cdot L < 0$



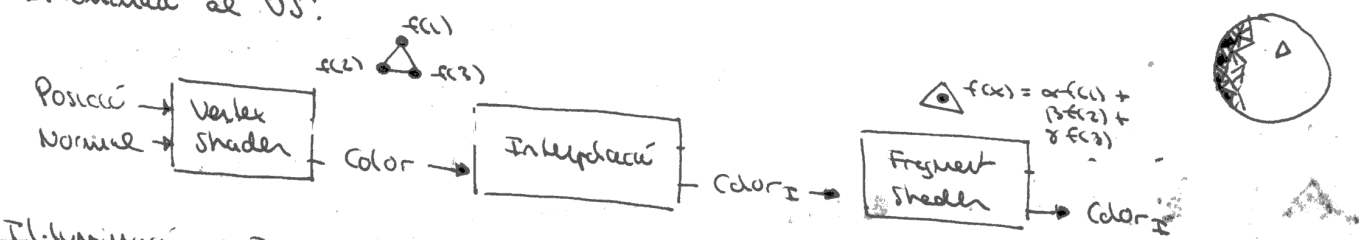
k_a k_d k_s
coeficients del material (RGB)

I_a I_L
llum (RGB)

n
brillantor del material (mida de la taca)

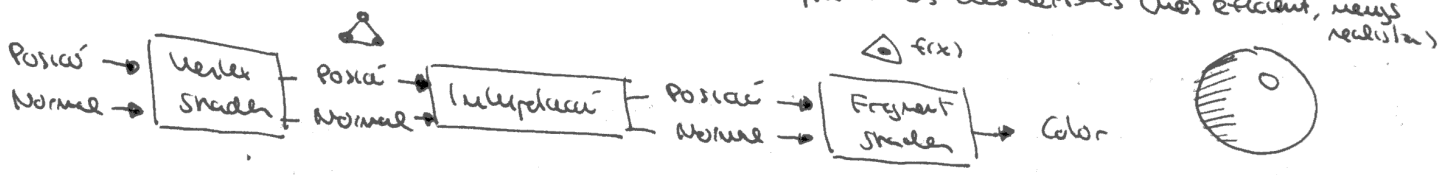
```
phong(L, N, V)
  let ambient = ka · Ia
  if (N · L > 0)
    let RL = reflect(-L, N)
    let difuse = kd · IL · (N · L)
    let especular = ks · IL · max(0, R·V)^n
  else
    let difuse = (0,0,0)
    let especular = (0,0,0)
  return ambient + difuse + especular
```

Il·luminació al VS:



A cada fragment es calcula el color interpolant els dels vèrtexs (més eficient, menys realista)

Il·luminació al FS:



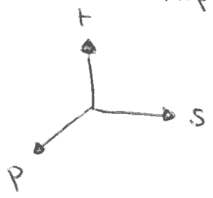
A cada fragment es calcula el color a partir de la posició i normal interpolades al fragment. (menys eficient, més realista)

Textures

Principals motius:

- Relleu
- Propietats optiques

Refinació: textura de 1 a 3 dimensions que conté texels amb entre 1 i 4 canals d'informació.



Exemples:

- Coloració de superfícies lenestres en funció de la corba de nivell (1D).
- Coloració de superfícies (engravats) (2D).
- Coloració de volums (3D).

Tipus de textures:

- Color (k_d)
- Brillantor o gloss (k_s)
- Opacitat (α)
- Normal (N)
- ~~Desplaçament o bump (z)~~
- Alçada o height (y)

Espai normalitzat:

$s, t, p \in [0,1]$ } $s,t,p \in \mathbb{R}$ (repetició de textura)

$x \in [1, width]$
$y \in [1, height]$ } Habitualment potències de 2
$z \in [1, depth]$

```
glTexParameteri(GL_TEXTURE_2D,
  GL_TEXTURE_WRAP_S, GL_REPEAT
glTexParameteri( ...,
  GL_TEXTURE_WRAP_T, ... )
```

Generació de textures

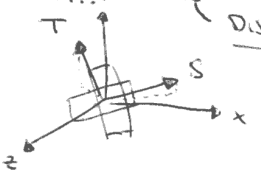
On:

- Al model
- A l'aplicació / GS
- Al VS (quan depenen de la càmera)
- Al FS (quan no volem que s'interpoli)

- **Plans S i T**

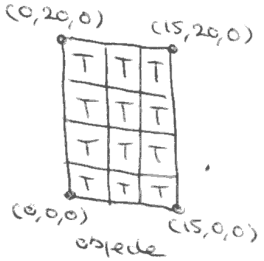
$$s(p) = d(p,S) = a_s p_x + b_s p_y + c_s p_z + d_s = \begin{pmatrix} a_s \\ b_s \\ c_s \\ d_s \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = S \cdot p$$

$$t(p) = d(p,T) = a_t p_x + b_t p_y + c_t p_z + d_t = \begin{pmatrix} a_t \\ b_t \\ c_t \\ d_t \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = T \cdot p$$



Distància al pla

Exemple:



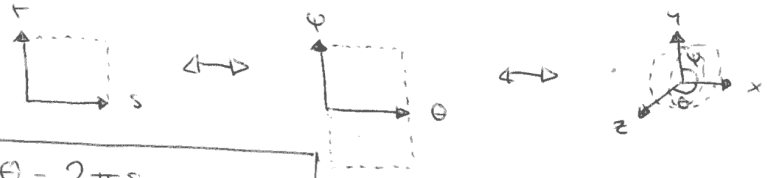
$s(0,0,0) = 0 \Rightarrow d_s = 0$
$s(0,20,0) = 0 \Rightarrow b_s = 0$
$s(15,0,0) = 3 \Rightarrow a_s = 3/15$

$S = (\frac{3}{15}, 0, c_s, 0)$, $c_s \in \mathbb{R}$

$t(0,0,0) = 0 \Rightarrow d_t = 0$
$t(0,20,0) = 4 \Rightarrow b_t = 4/20$
$t(15,0,0) = 0 \Rightarrow a_t = 0$

$T = (0, \frac{4}{20}, c_t, 0)$, $c_t \in \mathbb{R}$

- Mapatge esfèric

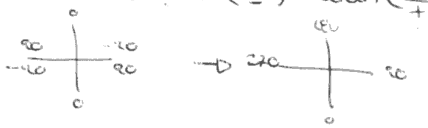


$\theta = 2\pi s$
$\varphi = \pi(t - 0.5)$

$x = \sin(\theta)\cos(\varphi)$
$y = \sin(\varphi)$
$z = \cos(\theta)\cos(\varphi)$

$\theta = \text{atan2}(x, z)$ *
$\varphi = \text{asin}(y)$
$s = \frac{\theta}{2\pi}$
$t = \frac{\varphi}{\pi} + 0.5$

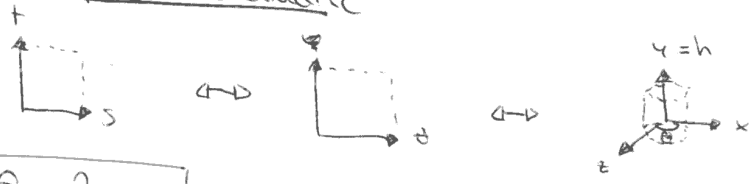
* atan2 té en compte el signe de x i z, a diferència d'atan (atan($\frac{-}{-}$) = atan($\frac{+}{+}$))



$S: \mathbb{R}^2 \to \mathbb{R}^3$
$s, t \to x, y, z$

$S^{-1}: \mathbb{R}^3 \to \mathbb{R}^2$
$x, y, z \to s, t$

- Mapatge cilíndric



$\theta = 2\pi s$
$h = t$

$x = \sin\theta$
$y = h$
$z = \cos\theta$

$\theta = \text{atan2}(x, z)$ *
$h = y$
$s = \frac{\theta}{2\pi}$
$t = h$

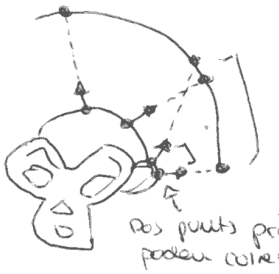
$C: \mathbb{R}^2 \to \mathbb{R}^3$
$s, t \to x, y, z$

$C^{-1}: \mathbb{R}^3 \to \mathbb{R}^2$
$x, y, z \to s, t$

- Mapatge d'esferes

Raig de visió reflectit

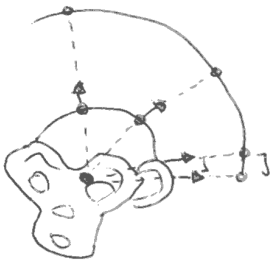
Normal de la superfície interseca



Normal de l'esfera

Els punts pròxims poden correspondre a dos punts allunyats

Solució



Centre de l'esfera

- Mapatge qualsevol (mesh parameterization)
  - Partint de la malla

  - Preservant àrea, angle, extensió, ...

Ús de textures

```
QImage image("image.png")
image = image.convertToFormat(QImage::Format_ARGB32).rgbSwapped().mirrored()
gl.GenTextures(1, &texture)
gl.BindTexture(GL_TEXTURE_2D, texture)                    <- Càrrega
gl.TexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image.width(), image.height(),
              0, GL_RGBA, GL_UNSIGNED_BYTE, image.bits())
gl.ActiveTexture(GL_TEXTURE0)
gl.BindTexture(GL_TEXTURE_2D, texture)
program->setUniformValue("sampler0", 0)                   <- Renderització
```

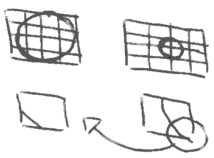
Shader

```
uniform sampler2D
sampler0

texel = texture(
sampler0, st)
```

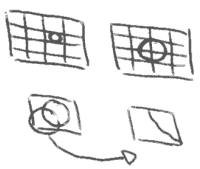
Filtrage de textures

- Magnification
- Minification



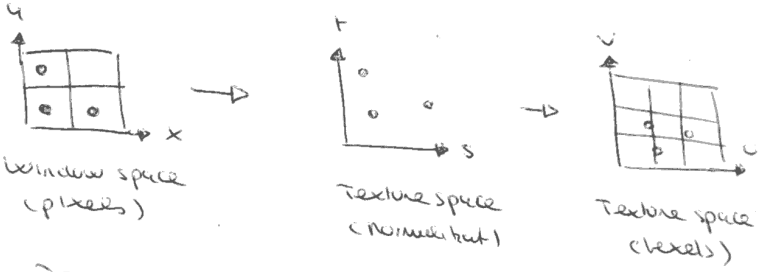
pixel < texel

$\frac{\partial s,t}{\partial x,y} < 1$



pixel > texel

$\frac{\partial s,t}{\partial x,y} > 1$



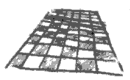
$$\frac{\partial s}{\partial x} \approx s(x+1,y) - s(x,y) = dFdx(st.s)$$

$$\frac{\partial s}{\partial y} \approx s(x,y+1) - s(x,y) = dFdy(st.s)$$

$$\frac{\partial t}{\partial x} \approx t(x+1,y) - s(x,y) = dFdx(st.t)$$

$$\frac{\partial t}{\partial y} \approx t(x,y+1) - s(x,y) = dFdy(st.t)$$

Exemple:



$\frac{\partial s}{\partial x} > 1, \frac{\partial t}{\partial y} \ll 1$

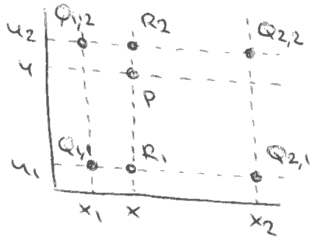
$\frac{\partial s}{\partial x} < 1, \frac{\partial t}{\partial y} < 1$

- Filtrage de magnification
  - Proximilat

```
glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MAG_FILTER, GL_NEAREST)
```

  - Interpolation bilinéaire

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



$$f(R_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{1,1}) + \frac{x - x_1}{x_2 - x_1} f(Q_{2,1})$$

$$f(R_2) = \frac{x_2 - x}{x_2 - x_1} f(Q_{1,2}) + \frac{x - x_1}{x_2 - x_1} f(Q_{2,2})$$

$$f(P) = \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

- Filtrage de minification
  - Sans mipmapping (LOD 0)

```
glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MIN_FILTER, GL_NEAREST)
```

  - Proximilat

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

  - Interpolation bilinéaire
- Avec mipmapping

```
glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MIN_FILTER, ...)
```

GL_NEAREST_MIPMAP_NEAREST
  - Proximilat (LOD λ)

GL_LINEAR_MIPMAP_NEAREST
  - Interpolation bilinéaire (LOD λ)

GL_NEAREST_MIPMAP_LINEAR
  - Interpolation linéaire de proximilat a LODs λ, λ'

GL_LINEAR_MIPMAP_LINEAR
  - Interpolation linéaire de interpolation bilinéaire a LODs λ, λ'

Idéalement puissance de 2

Resolution de la texture

Minification

...	...	...
1	1	0
	2	1
	4	2
	8	3
	...	...
	2^λ	λ

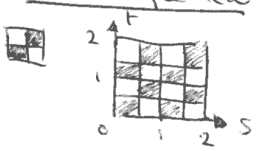
$f(\frac{\partial s}{\partial x}, \frac{\partial s}{\partial y}, \frac{\partial t}{\partial x}, \frac{\partial t}{\partial y})$ LOD = $\lambda = \log_2 f(\dots)$

Level of detail

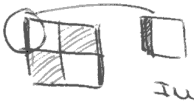
Repetició de la textura (wrapping)

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT
GL_TEXTURE_WRAP_T GL_CLAMP_TO_EDGE)

- Dins repetició de la textura GL_REPEAT

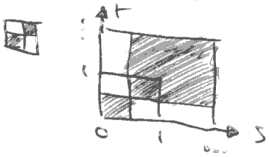


Problema de filtratge:



Interpola punts a l'altra banda de la textura

- Fora repetició de la textura GL_CLAMP_TO_EDGE

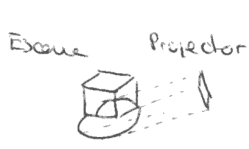


Sense problema de filtratge

Combinació del color de la textura

- Substitució fragmentColor = textureColor
- Modulació fragmentColor = textureColor * vertexColor
- Transparència (decal) fragmentColor = mix(vertexColor, textureColor, textureColor.a)
  if (textureColor.a < threshold) discard

Mapatge projectiu



Al VS:

$$T(0.5) \cdot S(0.5) \cdot P_T \cdot V_T \cdot M \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w \end{pmatrix}$$

Al FS:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} / w = \begin{pmatrix} s \\ t \end{pmatrix}$$

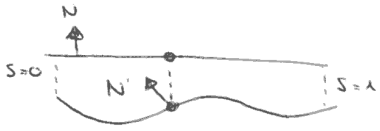
Aixecem la divisió de perspectiva per a poder interpolar linealment x', y' i z'.

Aplicacions

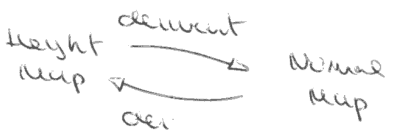
- **Mapejat de color**

- La textura és un mapa de color (k_d). RGB.
- S'aplica al FS.

- **Bump / normal mapping**

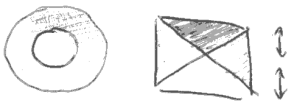


- La textura és un mapa d'alçades / normals D / XYZ.
- S'aplica al FS.

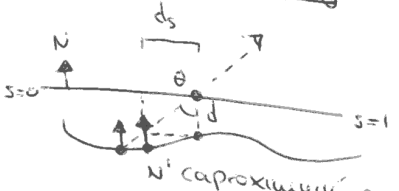


Problema:

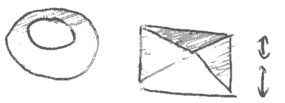
Sempre assumeix la mateixa N'. (Sense paral·laxi).
Solució: parallax mapping.



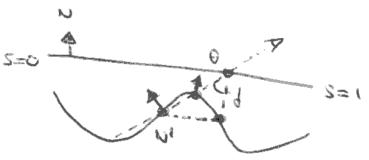
- **Parallax mapping**



- La textura és un mapa d'alçades i normals XYZD.
- S'aplica al FS.

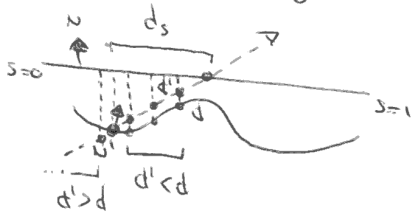


Problemes:

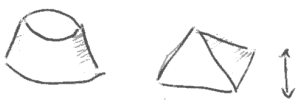


Assumeix V' erròniament (sense obstrucció).
Solució: relief mapping.

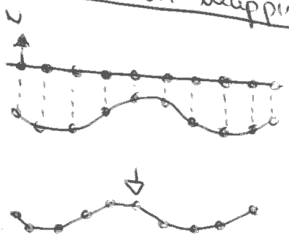
- **Relief mapping**



- La textura és un mapa d'alçades i normals XYZD.
- S'aplica al FS.



- **Displacement mapping**

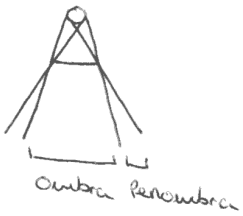


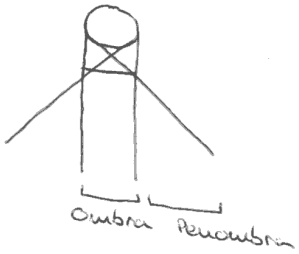
- La textura és un mapa d'alçades D.
- S'aplica al model, al GS o als TCS / TES (Tessellation control / evaluation)
  En els dos darrers casos es controla el nombre de subdivisions pot variar en funció de la posició de la càmera (més eficient)

Ombres

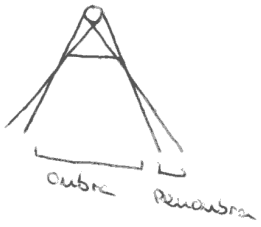
Ombra i penombra

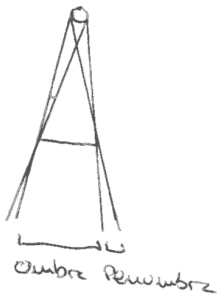
- Depèn de la mida de la font.





- Depèn de la distància entre la font:





Projecció (sense stencil)

- Sense test de profunditat

1) draw (receptor)
2) glDisable (GL_LIGHTING)
   glDisable (GL_DEPTH_TEST)
   glMatrixMode (GL_MODELVIEW)
   glPushMatrix()
   glMultMatrixf (matriu Projecció)
   draw (objecte ~~Projectat~~)
   glPopMatrix()
3) glEnable (GL_LIGHTING)
   glEnable (GL_DEPTH_TEST)
   draw (objecte)

- Amb desplaçament

1) draw (receptor)
2) shader → setUniform ("shadow", true)
   shader → setUniform ("modelMatrix", ...)
   glEnable (GL_POLYGON_OFFSET_FILL)
   * glPolygonOffset (-1, -1)
   draw (object)
3) shader → setUniform ("shadow", false)
   glDisable (GL_POLYGON_OFFSET_FILL)
   draw (object)

Z-fighting

* offset variable (↓) — (↑) offset constant

glPolygonOffset (f, u) ⇒

$z' = z + \max\left(\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}\right) f + r \cdot u$

r valor mínim que garanteix que offset > 0.

Problema: l'ombra pot excedir el receptor.

Projecció (amb stencil)

Stencil buffer *

- Habilitació

```
QOpenGLFormat format
format.setStencil(true)
QOpenGLFormat::setDefaultFormat(format)
```

- Obtenció d'un formació

```
glGetIntegerv(GL_STENCIL_BITS, &bitCount)
```

- Esborrat

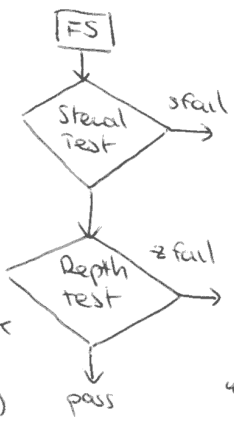
```
glClearStencil(0)
glClear(GL_STENCIL_BUFFER_BIT)
```

- Activació

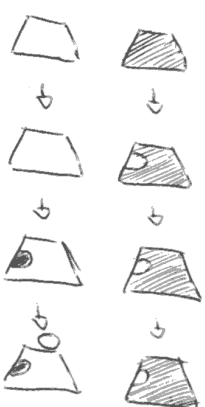
```
glEnable(GL_STENCIL_TEST)
glStencilFunc(GL_NEVER / GL_ALWAYS / GL_LEQUAL / GL_LESS, Vref, mask)
```

(Vref & mask (op) V & mask)

glStencilOp (sfail, zfail, pass)
GL_KEEP / GL_ZERO / GL_INCR / GL_DECR / GL_INVERT / GL_REPLACE (Vref)

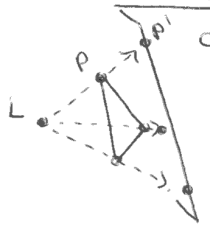


1) glEnable (GL_STENCIL_TEST)
   glStencilFunc (GL_ALWAYS, 1, 1)
   glStencilOp (GL_KEEP, GL_KEEP, GL_REPLACE)
   draw (receptor)
2) glDisable (GL_DEPTH_TEST)
   glColorMask (GL_FALSE, ..., GL_FALSE)
   glStencilFunc (GL_EQUAL, 1, 1)
   glStencilOp (GL_KEEP, GL_KEEP, GL_ZERO)
   glPushMatrix()
   glMultMatrixf (matriu Projecció)
   draw (objecte)
   glPopMatrix()
3) glEnable (GL_DEPTH_TEST)
   glDepthFunc (GL_LEQUAL)
   glColorMask (GL_TRUE, ..., GL_TRUE)
   glDisable (GL_LIGHTING)
   glStencilFunc (GL_EQUAL, 0, 1)
   draw (object)
4) glEnable (GL_LIGHTING)
   glDepthFunc (GL_LESS)
   glDisable (GL_STENCIL_TEST) draw (objecte)



Projecció (matriu de projecció)

Sigui $ax + by + cz + d = 0$ l'equació del pla receptor projectat; (l_x, l_y, l_z) les coordenades del focus de llum; (p_x, p_y, p_z) les coordenades del punt projectat; (p'_x, p'_y, p'_z) les coordenades d'on projecció:



$ap'_x + bp'_y + cp'_z + d = 0$ (P' pertany al pla receptor)

$$\frac{p'_x - l_x}{p_x - l_x} = \frac{p'_y - l_y}{p_y - l_y} = \frac{p'_z - l_z}{p_z - l_z}$$ (P' pertany a la recta LP).

Resolent obtenim que $P' = MP$.

- Focus puntual

$$M = \begin{pmatrix} -d - bl_y - cl_z & bl_x & cl_x & dl_x \\ al_y & -d - al_x - cl_z & cl_y & dl_y \\ al_z & bl_z & -d - al_x - bl_y & dl_z \\ a & b & c & -al_x - bl_y - cl_z \end{pmatrix}$$

- Focus puntual (origen)

$$M = \begin{pmatrix} -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & -d & 0 \\ a & b & c & 0 \end{pmatrix}$$

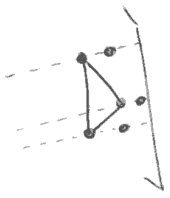
Demostració:

$$\left.\begin{array}{l} ap'_x + bp'_y + cp'_z + d = 0 \\ P' = \lambda P, \ \lambda \neq 1 \end{array}\right\} \Rightarrow a\lambda p_x + b\lambda p_y + c\lambda p_z + d = 0 \Rightarrow \lambda = \frac{-d}{ap_x + bp_y + cp_z}$$

$$\therefore \quad M_P = \begin{pmatrix} -dp_x \\ -dp_y \\ -dp_z \\ ap_x + bp_y + cp_z \end{pmatrix}$$

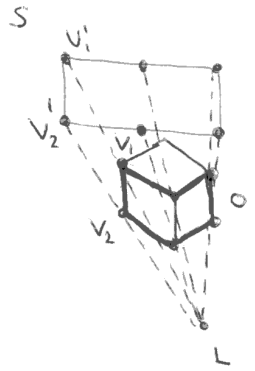
Dividim per aquest component

- Focus direccional (punt a distància infinita)



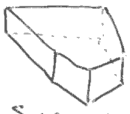
$$M = -\begin{pmatrix} -d - bl_y - cl_z & \cdots & dl_x \\ \vdots & \ddots & \vdots \\ a & \cdots & -al_x - bl_y - cl_z \end{pmatrix} + \begin{pmatrix} -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & -d & 0 \\ a & b & c & 0 \end{pmatrix}$$

Volums d'ombres



$SV_s(O,L) = \{ P : P \in S \wedge PL \cap O \neq \emptyset \}$ és el volum de l'ombra de l'objecte O resultant de la font de llum L. Són els punts P de l'espai delimitat S tals que la recta entre P i L intersecta amb O.

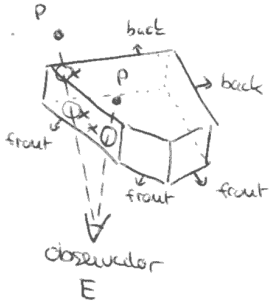
↑ Per exemple, l'escena.



El generem a partir de les arestes de contorn, les arestes de O amb una cara visible i una altra no visible.

$V_1' = V_1 + \lambda(V_1 - L)$
$V_2' = V_2 + \mu(V_2 - L)$ → Generem la cara del volum (V_1, V_2, V_1', V_2') a partir de l'aresta (V_1, V_2).

Les cares del volum infinit són innecessàries perè no afecten el comportament de l'algorisme.



$F_s(O,L)$ són les cares de $SV_s(O,L)$. Un punt P qualsevol:

$P \in SV_s(O,L) \Leftrightarrow \exists \{ f : f \in F_s(O,L) \wedge P \in f \neq \emptyset \}$

$\Leftrightarrow |\{ f : f \in front(F_s(O,L), E) \wedge P \in f \neq \emptyset \}| > |\{ f : f \in back(F_s(O,L), E) \wedge P \in f \neq \emptyset \}|$

1) glColorMask(GL_FALSE, ..., GL_FALSE)) dibuixa al z-buffer
   draw(escena)

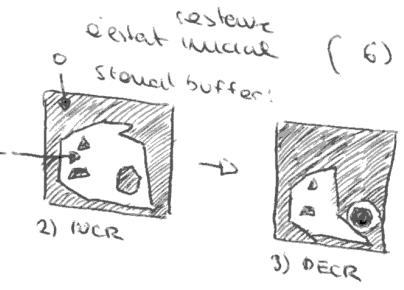
2) glEnable(GL_STENCIL_TEST)) dibuixa al stencil buffer les cares frontals
   glDepthMask(GL_FALSE)
   glStencilFunc(GL_ALWAYS, 0, 0)
   glEnable(GL_CULL_FACE) *
   glStencilOp(GL_KEEP, GL_KEEP, GL_INCR)
   glCullFace(GL_BACK)
   draw(volumOmbra)

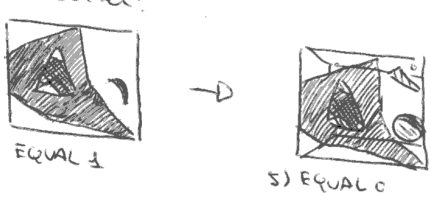
3) glStencilOp(GL_KEEP, GL_KEEP, GL_DECR)) dibuixa al stencil buffer les cares posteriors
   glCullFace(GL_FRONT)
   draw(volumOmbra)

4) glDepthMask(GL_TRUE)) dibuixa al color buffer la part fosca
   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)
   draw(escena)

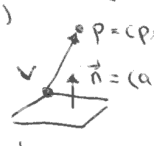
5) glStencilFunc(GL_EQUAL, 0, 1)) dibuixa al color buffer la part clara
   glEnable(GL_LIGHTING)
   draw(escena)

6) glDepthFunc(GL_LESS)
   glDisable(GL_STENCIL_TEST)





Face culling *

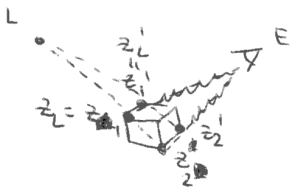


distancia(P, (a,b,c,d)) = $apx + bpy + cpz + d$ $\begin{cases} > 0 : GL\_FRONT \\ < 0 : GL\_BACK \end{cases}$

Alternativament:

$(p - v)\vec{n}$ $\begin{cases} > 0 : GL\_FRONT \\ \leq 0 : GL\_BACK \end{cases}$

Mapatge d'ombres



Generem el mapa d'ombres a partir del mapa de profunditat resultant de renderitzar l'escena des de la perspectiva de la font de llum L.

El mapa d'ombres conté les z_L més properes a L.

Renderitzem l'escena des de la perspectiva de la càmera E

Al VS:

$$T(0.5) \cdot S(0.5) \cdot P_L \cdot V_L \cdot M \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w \end{pmatrix}$$

$T(0.5)$: $[-0.5, 0.5] \to [0,1]$ (textura)

$S(0.5)$: $[-1,1] \to [-0.5, 0.5]$

Al FS:

$$\begin{pmatrix} x' \\ y' \\ \hline w \end{pmatrix} = \begin{pmatrix} s \\ t \end{pmatrix}$$

Resposem la divisió de perspectiva per a poder interpolar linealment x', y', z'.

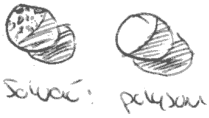
if $\frac{z'}{w} \leq \text{ShadowMap}(s,t) = z_L$ → visible des de L

else → ocult des de L (ombra)

1) glActiveTexture(GL_TEXTURE0) — Habilitació de la textura
   glGenTextures(1, &textureId)
   glBindTexture(GL_TEXTURE_2D, textureId)
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE)
   " " GL_TEXTURE_WRAP_T, "
   " " GL_TEXTURE_MIN_FILTER, GL_LINEAR)
   " " 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)
2) glViewport(0,0, SHADOW_MAP_WIDTH, SHADOW_MAP_HEIGHT)
   glMatrixMode(GL_PROJECTION)
   glLoadIdentity()
   gluPerspective(fov, ar, near, far) // Des de la font de llum — Generació de la matriu de visualització i projecció des de la font de llum
   glMatrixMode(GL_MODELVIEW)
   glLoadIdentity()
   gluLookAt(lightPosition, ..., lightTarget, ..., lightUp)
3) glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
   glPolygonOffset(1,1)
   glEnable(GL_POLYGON_OFFSET_FILL) — Rasterització: eliminar del depth buffer coses de la font de llum
   drawScene()
   glDisable(GL_POLYGON_OFFSET_FILL)
4) glBindTexture(GL_TEXTURE_2D, textureId)
   glCopyTexSubImage2D(GL_TEXTURE_2D, 0,0,0,0,0, SHADOW_MAP_WIDTH, SHADOW_MAP_HEIGHT) — Còpia del depth buffer al shadow buffer (textura)
5) Generació de la matriu de visualització i projecció des de la càmera
6) Rasterització des de la càmera amb VS i FS

Problemes:

- Acne (sampling errors)



Solució: polygon offset

- Duelling Frusta

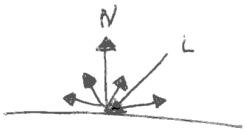
- Aliasing

Solució: filtering

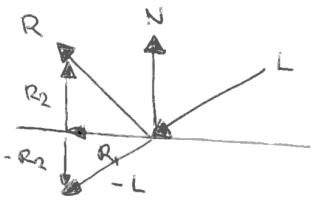
- Llum direccional

Reflexió

Reflexió difusa



Reflexió especular



$$R = R_1 + R_2 = \boxed{2(N \cdot L)N - L}$$

$$R_1 = -L + R_2$$

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

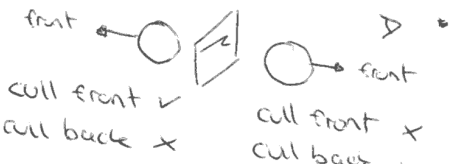
Objectes virtuals reflectits (sense stencil)

1) glPushMatrix()
   glMultMatrix(matriu Simetria)) objectes virtuals i llum reflectits
   glLightfv(GL_LIGHT0, GL_POSITION, posicio)
   glCullFace(GL_FRONT)
   draw(escena)
   glPopMatrix()

2) glLightfv(GL_LIGHT0, GL_POSITION, posicio)) llum
   glCullFace(GL_BACK)) Mirall semi-transparent
   draw(mirall)

3) draw(escena)) objectes reals escena

Problema:



Solució: invertir front i back

Problema:

Solució: reflectir llum

Problema:

Solució: només reflectim els objectes davant del mirall.

Problema:

Solució: només reflectim els objectes visibles a través del mirall (retallem els ocults as glClipPlane) (complicat)

Solució: stencil

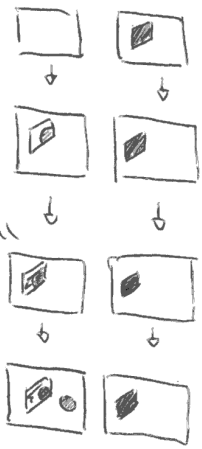
Objectes virtuals reflectits (sense stencil)

1) glEnable(GL_STENCIL_TEST)
   glStencilFunc(GL_ALWAYS, 1, 1)
   glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE)
   glDepthMask(GL_FALSE)
   glColorMask(GL_FALSE, ...)) stencil mirall
   draw(mirall)

2) glStencilFunc(GL_EQUAL, 1, 1)
   glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP)
   glDepthMask(GL_TRUE)
   glColorMask(GL_TRUE, ...)) objectes virtuals reflectits al mirall
   glPushMatrix()
   glMultMatrix(matriu Simetria)
   glLightfv(GL_LIGHT0, GL_POSITION, posicio)
   glCullFace(GL_FRONT)
   draw(escena)
   glPopMatrix()
   glDisable(GL_STENCIL_TEST)

3) glLightfv(GL_LIGHT0, GL_POSITION, posicio)
   glCullFace(GL_BACK)
   draw(mirall)) Ombra, mirall i escena

4) draw(escena)

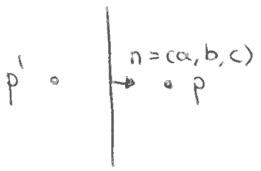


Objectes virtuals reflectits (amb textures dinàmiques)

Renderitzacio de l'escena reflectida i copia en una textura. Renderitzacio de l'escena i el mirall aplicant la textura (la part corresponent).

Es pot aprofitar la textura fins que no hi ha un canvi.

Matriu de reflexió

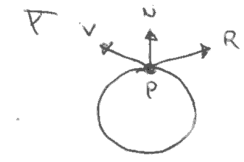


$ax+by+cz+d=0$

$$p' = p - 2 \cdot \text{distància}(p, \text{pla}) \cdot n$$
$$= p - 2(ap_x + bp_y + cp_z + d) \cdot n$$
$$= Mp$$

$$M = \begin{pmatrix} 1-2a^2 & -2ab & -2ac & -2ad \\ -2ab & 1-2b^2 & -2bc & -2bd \\ -2ac & -2bc & 1-2c^2 & -2cd \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Mapatge de l'entorn

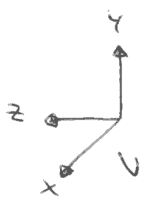


$$V = \frac{O-P}{\|O-P\|} \qquad R = \text{reflect}(-V, N) = 2(N \cdot V)N - V$$

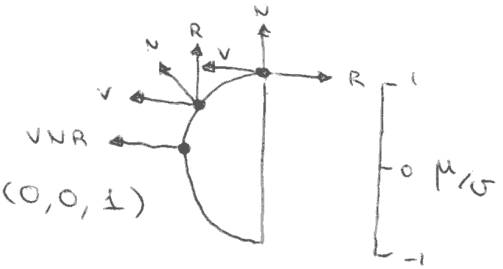
fragmentColor = environmentMap(R) ← textura de l'entorn que conté el color de cada direcció R

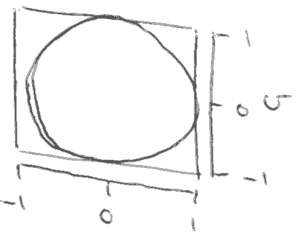
També es fa servir per a representar l'entorn (background) amb $R = -V$.

- **Sphere mapping**



$V = (0,0,1)$





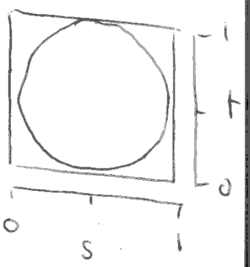
$$R = 2(N \cdot V)N - V = 2\left(\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}\right)\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = 2\begin{pmatrix} n_z \\ n_z \\ n_z \end{pmatrix}\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$
$$= (2n_z n_x, 2n_z n_y, 2n_z^2 - 1) = (r_x, r_y, r_z)$$

$$M = p_x = n_x = \frac{r_x}{2n_z} \qquad S = \frac{M+1}{2}$$

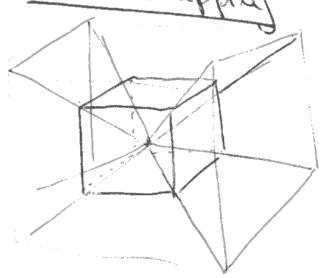
$$V = p_y = n_y = \frac{r_y}{2n_z} \qquad t = \frac{V+1}{2}$$

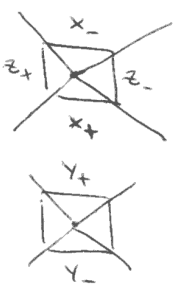
$$p_z = n_z = \sqrt{\frac{r_z+1}{2}}$$

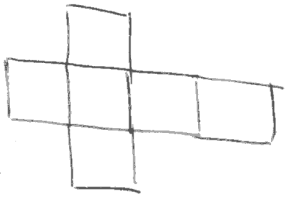
uniform sampler2D sampler
fragColor = texture(sampler, st)



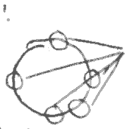
- **Cube mapping**







Problema!
worldspace ✗
Eyespace ✓



Realz Distorsió a les vores

glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X_EXT, ...)
NEGATIVE_X ″
POSITIVE_Y ″
NEGATIVE_Y ″
POSITIVE_Z ″
NEGATIVE_Z ″

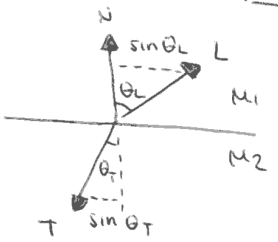
uniform samplerCube sampler
fragColor = textureCube(sampler, R)

worldspace ✓ Millora
Eyespace ✓ poca distorsió

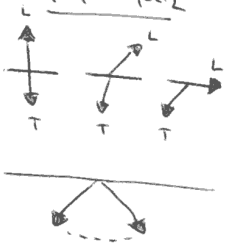
Refracció

Llei d'Snell

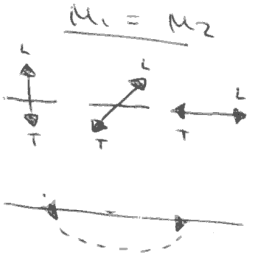
$$\sin\theta_T = \frac{\mu_1}{\mu_2}\sin\theta_L$$



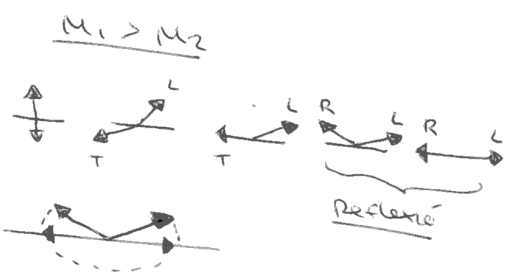
$\mu_1 < \mu_2$



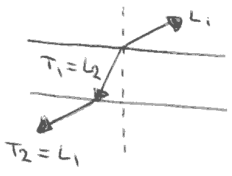
$\mu_1 = \mu_2$



$\mu_1 > \mu_2$



Refracció en superfícies paral·leles:
(per exemple, finestres)

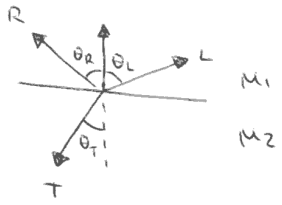


L // T (cas desplaçament)

Angle d'incidència crític:

$$\lambda = \frac{\mu_1}{\mu_2}\sin\theta_L \Rightarrow \boxed{\theta_L = \operatorname{asin}\frac{\mu_2}{\mu_1}}$$

Equacions de Fresnell

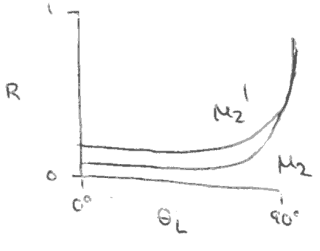


Assumim:
- Comportament especular pur
- No hi ha absorció ($R+T=1$)
- Material dielèctric (no conductor)

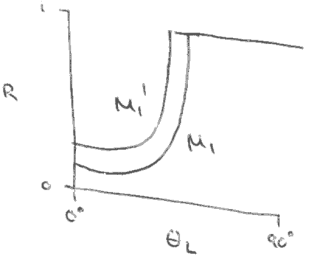
$$\boxed{R = \frac{R_S + R_P}{2}} \qquad \boxed{T = 1 - R}$$

$$\boxed{R_P = \left(\frac{\tan(\theta_T - \theta_L)}{\tan(\theta_T + \theta_L)}\right)^2} \qquad \boxed{R_S = \left(\frac{\sin(\theta_T - \theta_L)}{\sin(\theta_T + \theta_L)}\right)^2}$$

Modelen la proporció de llum reflectida R i llum refractada T.



$\mu_1 < \mu_2 < \mu_2'$



$\mu_1' > \mu_1 > \mu_2$

Aproximació d'Schlick:

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

$$\mu = \frac{\mu_1}{\mu_2} \qquad f = \frac{(1-\mu)^2}{(1+\mu)^2}$$

Alpha blending

Possibilitats:
- Objectes opacs ordenats / objectes transparents ordenats
- Objectes opacs desordenats / objectes transparents ordenats
- Objectes opacs desordenats / objectes transparents desordenats

Back-to-front →

z test	z write
=	=
✓	✓
✓	—
✓	✗

← Interpolació lineal

s'Enable(GL_BLEND)

s'BlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)

color = $\alpha_S \cdot (r_S, g_S, b_S, a_S) + (1-\alpha_S)\cdot(r_d, g_d, b_d, a_d)$) L'ordre importa

s'BlendFunc(GL_SRC_ALPHA, GL_ONE)

color = $\alpha_S \cdot (r_S, g_S, b_S, a_S) + 1\cdot(r_d, g_d, b_d, a_d)$) L'ordre no importa

Addició

Il·luminació global

- A diferència de la il·luminació local, té en compte la llum indirecta, a més de la directa (font → superfície → càmera).
- Hi ha dues grans famílies de tècniques: ray tracing i radiositat.

Radiometria

Símbol	Nom i unitat (radiometria)	Nom i unitat (fotometria)	Definició
Φ	Flux (W)	Flux (lm)	Energia emesa en un interval de temps.
E	Irradiància ($\frac{W}{m^2}$)	Il·luminació ($lux = \frac{lm}{m^2}$)	Energia incident en una superfície en un interval de temps (des de qualsevol direcció).
I	Intensitat ($\frac{W}{sr}$)	Intensitat ($cd = \frac{lm}{sr}$)	Energia emesa en una direcció en un interval de temps.
L	Radiància ($\frac{W}{sr \cdot m^2}$)	Lluminància ($\frac{cd}{m^2}$)	Energia emesa o incident en una direcció en una superfície en un interval de temps.

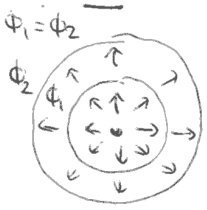
$E = \frac{\Phi}{A} \cos\theta = \frac{\Phi}{A} N \cdot L$

Angle pla (rad)

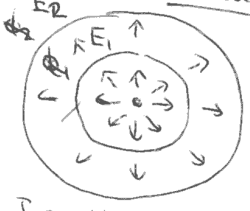
$[0, 2\pi]$

Angle sòlid (sr)

$[0, 4\pi]$

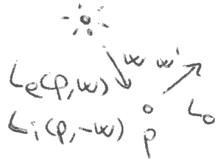


Flux



Irradiància

Irradiància i llei de Lambert

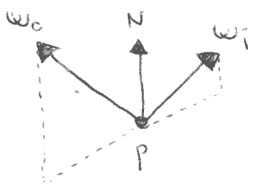


$$E(p) = \int_\Omega L_i(p,\omega) \cos\theta_\omega \, d\omega$$

L_e Radiància emesa
L_i Radiància d'entrada
L_o Radiància de sortida

Funcions de distribució bidireccional

- **BRDF** (reflexió)



$$f_r(p, \omega_o, \omega_i) = \frac{L_o(p, \omega_o)}{L_i(p, \omega_i) \cos\theta_i}$$

$f_r(p, \omega_o, \omega_i) \geq 0$ (no negativa)

$f_r(p, \omega_o, \omega_i) = f_r(p, \omega_i, \omega_o)$ (reciprocitat)

- **BTDF** (refracció)

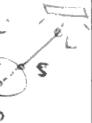
Light paths

Notació per a classificar camins de la font de llum (L) a l'observador (E).

Light → $L(DIS)^*E$ ← Eye

Difusa, Especular, Exacte

Construccions:
- DD (especularitat radiositat)
- DS
- SD
- SS (especularitat Ray tracing)



$\int_\Omega f_r(p, \omega_o, \omega_i) \cos\theta_i \, d\omega_i \leq 1$ (conservació)

Equació general del rendering

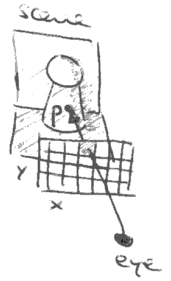
$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_\Omega f(p, \omega_o, \omega_i) \cdot L_i(p, \omega_i) \cos\theta_i \, d\omega_i$$

← Per a una freqüència (RGB) i en un instant de temps.

Ray Tracing

Ray casting

```
for each y ∈ {y_1, ..., y_h}
  for each x ∈ {x_1, ..., x_w}
    let r = ray(x, y, eye)
    let p = intersection(r, scene)
    let color_{x,y} = L_o(p, r)
```

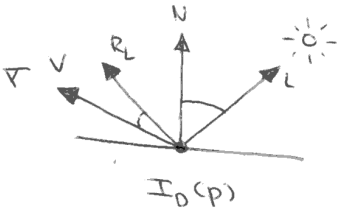


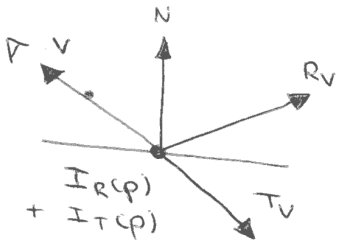
Ray tracing clàssic

$$I(p) = \underbrace{I_D(p)}_{\text{Directa}} + \underbrace{I_R(p) + I_T(p)}_{\text{Indirecta (reflexió i refracció)}}$$

$$I_D(p) = \underbrace{K_a I_a}_{\text{Ambient}} + \underbrace{K_d \sum_L I_L (N \cdot L)}_{\text{Difosa}} + \underbrace{K_s \sum_L I_L (R_L \cdot V)^n}_{\text{Especular}}$$

$$I_R(p) = K_R L_R \qquad I_T(p) = K_T L_T$$

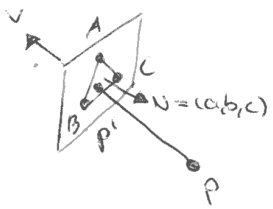





```
for each y ∈ {y_1, ..., y_h}
  for each x ∈ {x_1, ..., x_w}
    let R = ray(x, y, eye)
    let color_{x,y} = L(R, 1, 0)

L(R, M, depth):
  if depth < MAX-DEPTH ← else color = bg(color SCENE) / (0,0,0) / ...
    let P = intersection(R, SCENE)
    if P                ← else let color = bg(color(R, SCENE))
      let color = I_D(P, SCENE)
      if reflects(P)
        let R_R = reflect(R, N_P))
        let L_R = L(R_R, M, depth + 1)
        let color = color + K_R(P) · L_R
      if refracts(P)
        let T_R = refract(R, N(P), M(P) / M)
        let L_T = L(T_R, M(P), depth + 1)
        let color = (color + K_T(P) · L_T
  return color
```

Intersecció ray-triangle



1) Intersecta un el pla?

$p' = p + \lambda v, \quad \lambda \geq 0$

$p' \in pla \Leftrightarrow ap'_x + bp'_y + cp'_z + d = 0$

$\Leftrightarrow a(p_x + \lambda v) + b(p_y + \lambda v) + c(p_z + \lambda v) + d = 0$

$\Leftrightarrow \lambda = \dfrac{-(ap_x + bp_y + cp_z + d)}{aV_x + bV_y + cV_z}$

$\Leftrightarrow \boxed{\lambda = \dfrac{-d(p, pla)}{N \cdot V} = \dfrac{-\left(\frac{N}{d}\right)\left(\frac{p}{1}\right)}{N \cdot V}}$

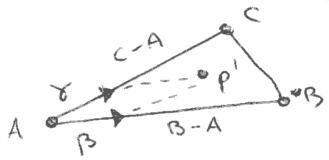
$N \cdot V = 0 \Rightarrow N \perp V$
- $N \cdot P = 0 \Rightarrow p' \in Pla$
- $N \cdot P \neq 0 \Rightarrow p' \notin Pla$

$N \cdot V \neq 0 \Rightarrow$
- $\lambda < 0 \Rightarrow p' \notin Pla$
- $\lambda \geq 0 \Rightarrow p' \in Pla$

$d = -(aA_x + bA_y + cA_z) = -N \cdot A$

(qualsevol punt del triangle pertany al pla)

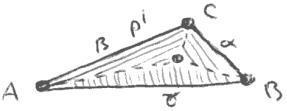
2) Intersecta un el triangle?



$p' = A + \beta(B-A) + \gamma(C-A)$

$= (1 - \beta - \gamma)A + \beta B + \gamma C$

$= \alpha A + \beta B + \gamma C$



$\alpha = \dfrac{\text{Àrea}(P', B, C)}{\text{Àrea}(A, B, C)}$

$\beta = \dfrac{\text{Àrea}(P', A, C)}{\text{Àrea}(A, B, C)}$

$\gamma = \dfrac{\text{Àrea}(P', A, B)}{\text{Àrea}(A, B, C)}$

$\alpha + \beta + \gamma = 1$

$0 \leq \alpha \leq 1$

$0 \leq \beta \leq 1$

$0 \leq \gamma \leq 1$

$\Downarrow$

$p' \in$ triangle

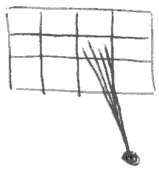
$$\alpha = \frac{\|(B-p') \times (C-p')\|}{\|(B-A) \times (C-A)\|}$$

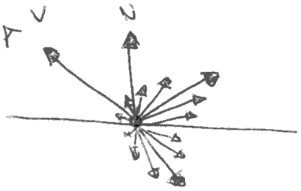
$$\beta = \frac{\|(A-p') \times (C-p')\|}{\|(B-A) \times (C-A)\|}$$

$$\gamma = \frac{\|(A-p') \times (B-p')\|}{\|(B-A) \times (C-A)\|}$$

Soporte caminos de tipos LDS*E

Path tracing





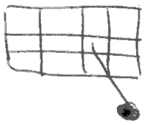
```
for each y ∈ {y1, ..., yn}
  for each x ∈ {x1, ..., xw}
    let color_x,y = (0,0,0)
    for each i ∈ {1, ..., n}
      let R = ray(x + random(-0.5, 0.5), y + random(-0.5, 0.5), eye)
      let color_x,y = color_x,y + L'(R, 1, 0)
    let color_x,y = color_x,y / n
```

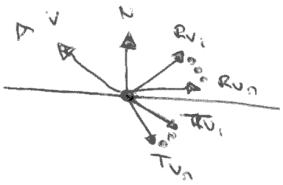
Prenem n mostres per a cada pixel i calculem els rajos reflectits i transmesos en funció de la distribució de prova d'estat, per a reduir la reflexió difusa.

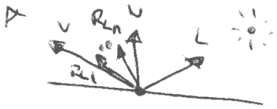
$L(D|S)^*E$

Problema: molt de soroll o moltes mostres (ineficient)

Distributed Ray Tracing



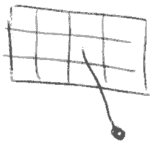


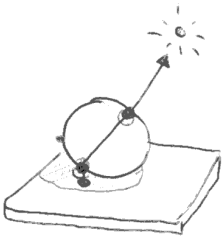


Prenem múltiples mostres (múltiples rajos) a cada punt, tenint en transicions suaus.

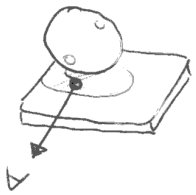
LDS^*E

Two-pass Ray Tracing





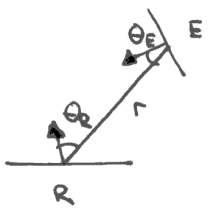
1) Ray tracing inversos des de la font de llum fins a un punt difós, on emmagatzem la llum incident (estructura de dades k-d tree).



2) Ray tracing des de l'observador fins a un punt difós, on recuperem la llum incident del punt més proper.

LS^*DS^*E

Oclusió ambient



L'emissor ocluu el receptor en funció de r, θ_R i θ_E.

$$A(p) = \frac{1}{\pi} \int_{\Omega} V(p,\omega)(N \cdot \omega)\, d\omega$$

Funció de visibilitat (cada p cap a ω) { Booleana (0 o 1) / Real (distància)

Més distància ⇒ Menys oclusió

Menys angle ⇒ Més oclusió