

G-RESUM-ESCRIT.pdf



Arnau_FIB



Gráficos



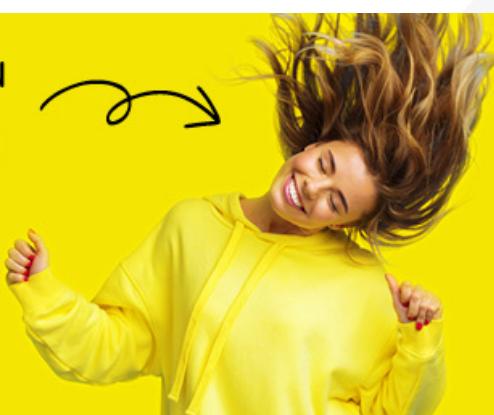
3º Grado en Ingeniería Informática



Facultad de Informática de Barcelona (FIB)
Universidad Politécnica de Catalunya

SI ESTÁS VIENDO ESTO TIENES UN
15% DE DESCUENTAZO

en tu carnet de conducir
con el código **HWWUO22**



saboteas a tu propia persona? cómo?? escríbelo **aquí** y táchalo

manual de instrucciones: escribe sin filtros

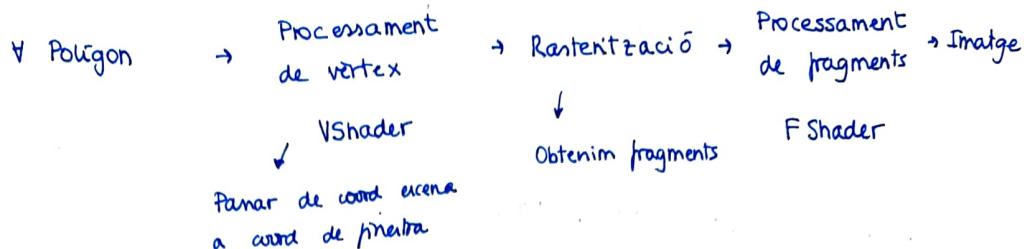
y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)



GRÀFICS

TEMA 0. PRESENTACIÓ

Paradigma projectiu



Paradigma Ray Tracing



TEMA 1. INTRODUCCIÓ

Physically-based rendering: El més realista, cost alt.

Limitacions HVS (Human Visual System):

Efectes visuals

Efectes de la llum (càustica, p.e.)

Elements de realisme:

Il·luminació (sombres, reflexió espectral, translucids,...)

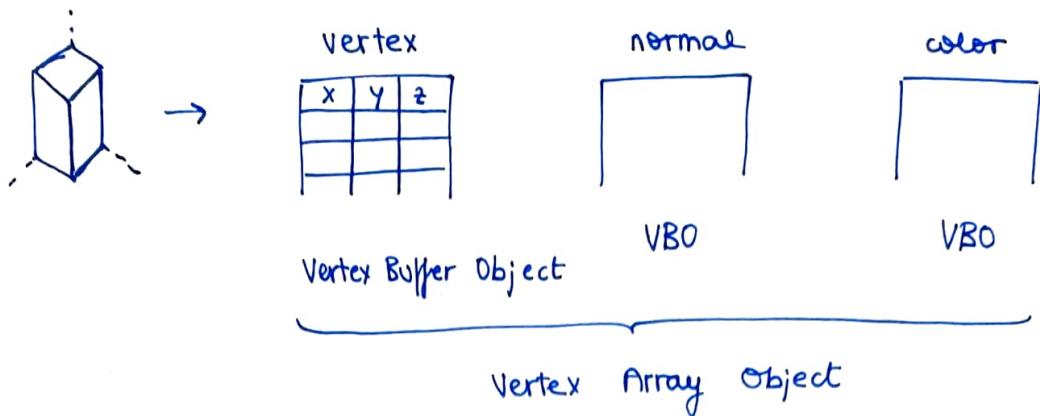
Details (nivell de polígons, textures,...)

Subsurface scattering: Com afecta la llum quan incideix sobre un material en concret

Conceptes llum:

- Llum directa / indirecta: llum de la font / que rebota
- Reflexió espectral / difusa: angle reflexió petit / gran (rap. N)
- Absorció: que traspasa el material
- Transmissió espectral / difusa: després de traspasar, angle petit / gran (respecte normal)

TEMA 3. PROCÉS DE VISUALITZACIÓ



SISTEMES DE COORDENADES I

TRANSFORMACIONS GEOMÈTRIQUES

Informació Punts
 Vectors Normal (perp a superfície)
 La recta

coordenades Homogènies

$$\text{Punt} \rightarrow \begin{array}{ccc} \text{3D} & & \text{Homog.} \\ (x, y, z) & \longrightarrow & (x, y, z, 1) \end{array} \quad w: \text{Divisió postergada de les components } x, y, z.$$

$$(x/w, y/w, z/w) \leftarrow (x, y, z, w) \quad w \neq 0$$

$$\text{Vectors} \rightarrow \begin{array}{ccc} \text{3D} & & \text{Homog.} \\ (x, y, z) & \longrightarrow & (x, y, z, 0) \end{array} \quad w=0$$

En coordenades homogènies tenim ventatges en transformacions geomètriques

Sistemes de coordenades (punts) Normalment $w_m = 1$

OBJECT SPACE (model space) $\rightarrow (x_m, y_m, z_m, w_m)$

On s'han creat els models

$\overbrace{\text{T.R.S.}}^{\text{MM}} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \begin{array}{l} \text{Translate} \\ \text{Scale} \\ \text{Rotate} \end{array}$

Modeling space: situar cada instància d'un objecte en relació a l'escena

WORLD SPACE (sc de l'aplicació) $\rightarrow (x_a, y_a, z_a, w_a)$

Per a representar l'escena.

$$\begin{bmatrix} x_a \\ y_a \\ z_a \\ w_a \end{bmatrix} = \text{ModelMatrix} \cdot \begin{bmatrix} x_m \\ y_m \\ z_m \\ 1 \end{bmatrix}$$

Si angle positiu, rotació Antihorari

WUOLAH

SI ESTÁS VIENDO ESTO TIENES UN

15%

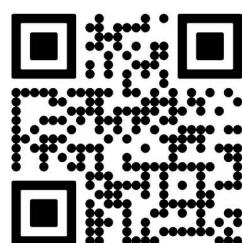
DE DESCUENTAZO

en tu carnet de conducir
con el código HVWU022

- Con nuestro método...
→ si no apruebas, no pagas
#NOESBROMA
- Los mejores precios variables
según ocupación ¡PÍLLATE UN
PACK Y AHORRA! Yasss kween
- > + de 100.000 alumnos ya
han confiado en nosotros ;)



Estamos en
Av. Cubelles, 21-23
Vilanova

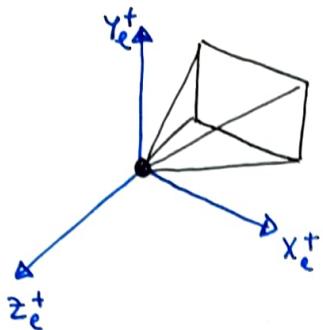


View Transform *

EYE SPACE

(x_e, y_e, z_e, w_e)

Sistemes de coordenades de l'observador



En OpenGL la càmera mira a la Z^-

Per tal que un punt sigui visible:

$$z_e < 0'$$

$$z_e < -z_{\text{near}}$$

* View Transform

Transformacions de rotació, escala i translació per orientar l'escena a fi que la càmera ho regi.

`lookAt(eye, target, up);` → normalment $(0, 1, 0)$

Amb angles euler:

$$T(0, 0, -d) \cdot R_z(-\varphi) \cdot R_x(\theta) \cdot R_y(-\psi) \cdot T(-VRP)$$

Projection Transform: Depèn del tipus de càmera (persp., axon.)

`perspective(fovy, aspect, near, far)`

60° ↓
width
height

CLIP SPACE

(x_c, y_c, z_c, w_c)

Coordenades de retallat (eu valen si estan entre $-w_c$ i w_c)

Perspective division: Dividir entre $(x_n, y_n, z_n) = \left(\frac{x_c}{w_c}, \frac{y_c}{w_c}, \frac{z_c}{w_c} \right)$

NORMALIZED DEVICE SPACE (x_n, y_n, z_n)

Coordenades normalitzades (entre -1 i 1)

Viewport Transform & Depth Transform: Depèn del viewport, la profunditat depèn de glDepthRange.

WINDOW SPACE (x_d, y_d, z_d)

Coordenades de dispositiu

Transformació de vectors

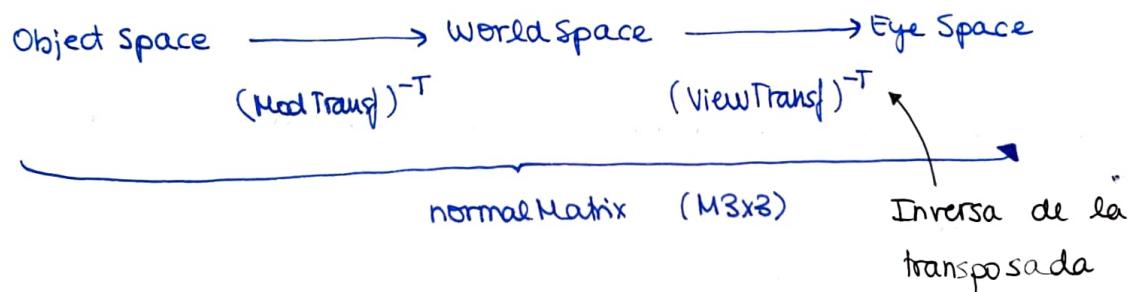
Object space: w_m serà 0 (en lloc d' w , en punts)

Model transform: No canvia res. Només que el resultat serà diferent en el $w_a = 0$

$$\begin{bmatrix} x_a \\ y_a \\ z_a \\ w_a \end{bmatrix} = \text{modelMatrix} \cdot \begin{bmatrix} x_m \\ y_m \\ z_m \\ 0 \end{bmatrix}$$

En si, per poder tenir vectors al window space (o abans), com que a van fent canvis de sistemes de coordenades, no es pràctic guardar els vectors llisos. És millor guardar dos punts $\vec{v} = p - q$ i poder calcular-los en el sistema de coord. denitjat simplement fent $\vec{v}' = p' - q'$. p', q' són els punts transform.

Transformació de vectors normals





saboteas a tu propia persona? cómo?? escríbelo **aquí** y táchalo

manual de instrucciones: escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

Pipeline OPEN GL

1. Dibuix de primitives

↳ punts, línies, polígons

- Vèrtex a vèrtex : glBegin, glVertex, glNormal, glEnd (Obj. Space)

- Vertex array object (VAO) : glDrawArrays, glDrawElements (Obj. Space)
 - ↳ El que es feia al inici de IDI

2. Per-vertex operations

- Es transformen els vèrtexs (modelview i projection):

passar de Obj. Space → Clip Space

- Es transformen les normals (transp de inv de modelview)

- Es pot calcular la il·luminació del vèrtex

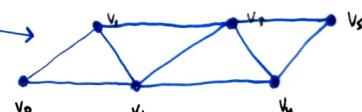
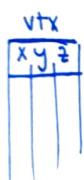
3. Primitive assembly

- Els vèrtexs s'agrupen formant primitives (punts, línies, triangles,...)

glDrawArrays (mode, first, count);

GL_POINTS, GL_LINES, GL_TRIANGLES.

GL_TRIANGLES_STRIP

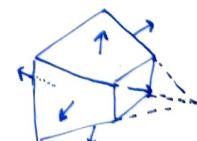


glDrawElements (mode, count, type, indices);

→ GL_TRIANGLES GL_TRIANGLES_STRIP

id defineix l'ordre dels vèrtex

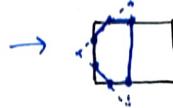
vtx	x	y	z	id
				0
				2
				4
				1



4. Primitive processing

- clipping a la piràmide de visió

si estan mig dins, mig fora, cal generar nous vèrtexs per què es vegi la part de dins



Càlcul de P i Norm per interpolació de punts

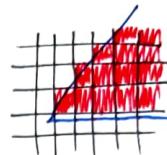
- ↓
- Up
- ↓
- Window
- Divisió de perspectiva : dividir (x, y, z) per w
 - Viewport & Depth transform : Passem de NDS a Window Space
 - Back face culling \rightarrow glEnable(GL_CULL_FACE)
(Eliminació de cares ocultes, aquelles que no es veuen)

5. Rastreització

- Generació de fragments corresponents a la primitiva

Cada fragment té :

- coordenades (window space)
- color (interpolat) \rightarrow En el centre del píxel
- coord de textura (interpolades)



- Si figura no ocupa el 100% del píxel, el criteri per decideix per pintar o no el píxel és pintar si hi ha figura al centre del píxel.

6. Fragment processing

- Càcul del color del fragment (textures, il·luminació, ...)

8. Per-fragment operations

- Pixel ownership test (si pertany a la app o no)
- Stencil test (sombres, reflexions) \rightarrow poder aplicar una màscara per renderitzar una part
- Depth test (problema visibilitat) \rightarrow cares tapades per objectes
- Blending (objectes translúcids) \rightarrow permet veure objectes de radera.

9. Frame buffer operations

- Es modifiquen els buffers que s'han enllistat en glDrawBuffers

- Es veu afectada per glColorMask (bool, bool, bool, bool)
- glDepthMask (bool)

Depth Enabled

WUOLAH

TEMA 6: TEXTURES

Textures → Details superficials { Variacions geometria
Variacions propietats òptiques

Definicions

Textura: Taula de $\frac{1}{2}$ dimensions, con cada cel·la emmagatzema una certa propietat amb $\frac{1}{2}$ canals.

Propietats de cada text-cell (texel)

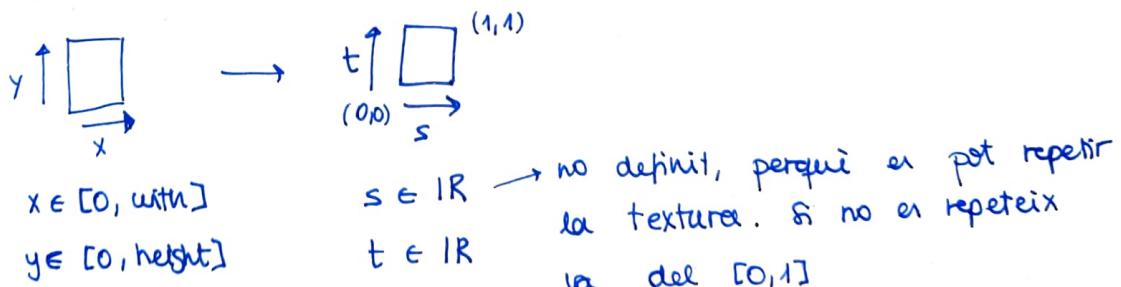
- K_d (color map) $\rightarrow (R, G, B)$
- K_s (gloss map) \rightarrow Es pot simplificar $K_s = (R_s, G_s, B_s) \rightarrow K_s \in [0, 1]$
- Opacitat (alpha map) $\rightarrow 0\%$ (transparent) $\rightarrow 100\%$ (opac)
- Normal (normal map) \rightarrow

$$\begin{matrix} (n_x, n_y, n_z) & \in [-1, 1] \\ \downarrow & \downarrow & \downarrow \\ (R, G, B) & \in [0, 1] \end{matrix}$$
- Desplaçament (bump mapping)
- Desplaçament (displacement mapping) \rightarrow crear nova geometria bon resultat amb vistes frontals i tangencials

Mida d'una textura

#texels en cada dimensió, potències de 2

- Espai normalitzat de la textura:



MAPPING

Param. superficial: $(s, t) \rightarrow (x, y, z) \rightarrow (x, y)$

Forward mapping: Funcions que donat un texel diuen a quin píxel pertany. (Image warping)

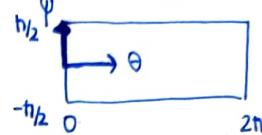
Inverse mapping: Donat un píxel, saben de quin texel han d'agafar la textura.

Mapping esfèric:

De $(s, t) \rightarrow (x, y, z)$

$(s, t) \rightarrow (\theta, \psi) \rightarrow (x, y, z)$

$$\begin{cases} \theta = 2\pi s \\ \psi = \pi(t - 0.5) \end{cases}$$



$$\begin{cases} x = \cos(\psi) \cdot \sin(\theta) \\ y = \sin(\psi) \\ z = \cos(\psi) \cdot \cos(\theta) \end{cases}$$

Mapping cilíndric:

De $(s, t) \rightarrow (x, y, z)$

$(s, t) \rightarrow (\theta, h) \rightarrow (x, y, z)$

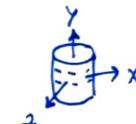
$$\theta = 2\pi s$$

$$h = t$$

$$x = \sin(\theta)$$

$$y = h$$

$$z = \cos(\theta)$$



MAPPING EN OPEN GL

Habitualment, cada vèrtex (obj. space) té una coordenada de textura.

```
in vec2 texCoord;
```

```
out vec2 vTexCoord;
```

```
uniform sampler2D colorMap;
```

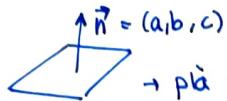
- On es generen les textures? A blender, a la aplicació (CPU), al Vertex Shader o al Fragment Shader.
Com abans s'ha dit, millor (menys cost)

**TOMARSE UN
CAFELITO YA HECHO
FEELS LIKE...**

**HACERSE CHULETAS
EN LA CALCULADORA.**

GENERAR COORDENADES DE TEXTURA

PLANS S, T



$(x, y, z, 1) \rightarrow \square \rightarrow (s, t)$

$$S = (a_s, b_s, c_s, d_s)$$

$$T = (a_t, b_t, c_t, d_t)$$

$$S = ax + by + cz + d \cdot w$$

$$= (a_s, b_s, c_s, d_s) \cdot (x, y, z, w)$$

$$t = (a_t, b_t, c_t, d_t) \cdot (x, y, z, w)$$

↓

$$s = \text{dot}(S, \text{vertex})$$

$$t = \text{dot}(T, \text{vertex})$$

AMB SUPERFÍCIE AUXILIAR

Vertex → Punt d'una ofera

s-mapping (mapping estèric)

coordinades (s, t)

s'-mapping

$$(x, y, z) \rightarrow (\theta, \psi) \rightarrow (s, t)$$

$$\begin{aligned} x &= \sin(\theta) \cos(\psi) & \theta &= \text{atan2}(x, z) & s &= \frac{\theta}{2\pi} \\ y &= \sin(\psi) & \psi &= \text{asin}(y) & t &= \frac{\psi}{\pi} + 0.5 \\ z &= \cos(\theta) \cos(\psi) \end{aligned}$$

funció que calcula
 $\text{atan2}(x/z)$ que preserva
els signes de x i z per
fer que el rang sigui
 $[0, 2\pi]$

CREACIÓ DE LA TEXTURA

S'ha d'en carregar la app mitjançant
crides de QT.

```

Load texture {
    QImage img0 ("textura.png");
    QImage T = img0.convertToFormat(QImage::Format_ARGB32);
    glGenTextures(1, &textureId0);
    glBindTexture(GL_TEXTURE_2D, textureId0);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, T.width(), T.height(),
                 detailFormatNow
                 0, GL_RGBA, GL_UNSIGNED_BYTE, T.bits());
    : margins format + valors de 0...255 buffer textura
}

```

```

glActivateTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, textureId1);
program->bind();
program->setUniformValue("colorMap", 0);

```

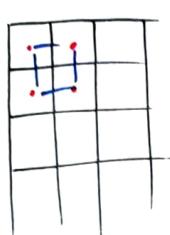
100% NATURALES

FILTRAT

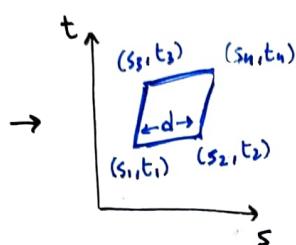
Dades unes coordenades (s, t) i un colorMap, decidir quin color mostrar en el fragment.

Es necessari perquè no sempre 1 fragment = 1 píxel de la textura. \Rightarrow Magnification (imatge > textura) ; Minification (imatge < textura)

- Magnification o minification?



window space



Texture space

Per saber si imatge > textura o al revés, s'agafen 4 píxels i es calcula la distància d entre centres.

si $d_{\text{pixel}} > d_{\text{texel}} \rightarrow$ Magnif.
 $d_{\text{pixel}} < d_{\text{texel}} \rightarrow$ Minif.

- càlcul amb derivades parcials:

$$\frac{\partial u}{\partial s} \Rightarrow \frac{\partial v}{\partial t} \rightarrow \text{(mida de la textura)} \quad \frac{\partial(u)}{\partial(y)}, \frac{\partial(u)}{\partial(x)} \quad \text{(en texels)} \quad \frac{\partial(v)}{\partial(y)}, \frac{\partial(v)}{\partial(x)}$$

TEXTURE FILTERS

com s'anava texture (sampler, tex coord)

- Magnification filters: GL_NEAREST (nearest neighbour)
 GL_LINEAR (interpolació bilineal)

- Minification filters:

Nipmapping: Tenir la textura ja carregada en reduccions més petites. Level of Details (LOD). Es fa 1 vegada

Amb fragments que no siguin el pla paral·lel a la textura, en consideren tota la der.	$\left[\begin{array}{c} \text{Factor Minification} \\ \downarrow \\ 2^{\lambda} x \end{array} \right] \quad \left[\begin{array}{c} \frac{\partial u}{\partial x} \\ \downarrow \\ 2^{\lambda} \end{array} \right] \quad \left[\begin{array}{c} \text{LOD} \\ \downarrow \\ \lambda \end{array} \right] \quad \text{(cas simple)}$
---	--

$$\rho = f\left(\frac{\partial u}{\partial x}, \frac{\partial v}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial v}{\partial y}\right)$$

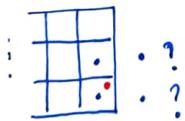
$$\lambda = \log_2(\rho)$$

Filtres disponibles a OPENGL :

Sense	{	GL_NEAREST	// Agafen les dades (color)
Mipmapping		GL_LINEAR	// texels) a LOD 0

Amb	{	GL_NEAREST_MIPMAP_NEAREST	
Mipmapping		↳ Nearest neighbour on LOD λ	
		GL_LINEAR_MIPMAP_NEAREST	
		↳ Bilinear on LOD λ	
		GL_NEAREST_MIPMAP_LINEAR	
		↳ $c_0 = \text{nearest on LOD } \lambda$	
		$c_1 = \text{nearest on LOD } (\lambda+1)$	
		mix ($c_0, c_1, \text{fract}(\lambda)$)	
		GL_LINEAR_MIPMAP_LINEAR → Tri linear	
		↳ $c_0 = \text{Bilinear on LOD } \lambda$	
		$c_1 = \text{Bilinear on LOD } \lambda+1$	
		mix ($c_0, c_1, \text{fract}(\lambda)$)	

WRAPPING

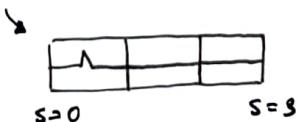


Quin color agafem quan s'acaba la textura?

(En magnification bilineal)

GL_REPEAT → $s' = \text{fract}(s)$ → es repeteix la textura

GL_CLAMP_TO_EDGE → Deixa un marge de



mig texel per poder agafar el color

COMBINACIÓ COLORS

Com li donem valor a frag color

REPLACE : frag color = tex color

Normalment usat quan la textura ja incorpora la il·luminació.

MODULATE : frag color = tex color * front color

Textura no incorpora la il·luminació.

DECAL : frag color = mix (front color, tex color, tex color.a)

Per textures amb component alpha.

A més, es pot afegir:

if (tex color.a < alpha threshold) discard;

INTERPOLACIÓ COORDENADES DE TEXTURA

En object space les coord. de textura varien linealment.

En window space (quan s'ha fet la div. de perspectiva) ja no varien linealment.

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

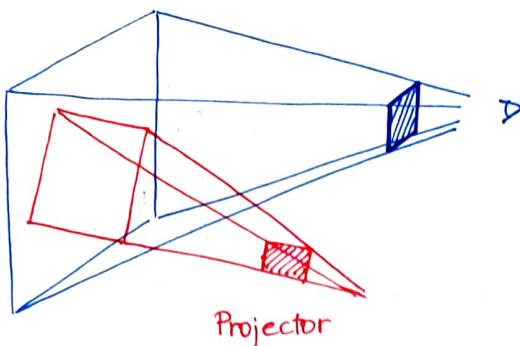
↳ Per interpolar: (s.w, t.w, w), obtenint un texel (s/t, q) accedim a la textura amb (s/q, t/q)

saboteas a tu propia persona? cómo?? escríbelo **aquí** y táchalo

manual de instrucciones: escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

PROJECTIVE TEXTURE MAPPING



Projecta una textura.

Tot ilum, no hi ha sombras.

Projector té la seva Viewmatrix, Projectionmatrix

El seu Viewport va de $(0,0)$ a $(1,1)$, per les coordenades de textura.

VS - gen coord textura

$$\begin{bmatrix} s \\ t \\ p \\ q \end{bmatrix} = T(0s) \cdot S(0t) \cdot P_p \cdot V_p \cdot M \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Passar a coord $[0,1]$

Object Space

No en fa la divisió de perspectiva perquè sinó en el FS no s'interpolaria bé.

FS - accés: Accedim a $(s/q, t/q)$

APLICACIÓNS

Color mapping: Agafar el color de la textura. En el FS.

Bump / Normal mapping: Utilitzar la normal de la textura per calcular la il·luminació. Es fa en el FS. Per vistes tangencials, en perd l'efecte de relleu.

View-motion parallax: Agafar les coordenades de textura amb un cert offset (en funció de l'angle entre la superfície i l'observador). Al FS

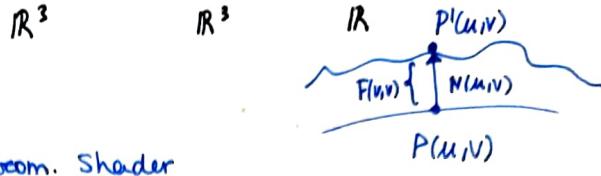
Relief mapping: com parallax mapping, però es capaç de detectar les auto-occlusions. Té un alt overhead (FS)

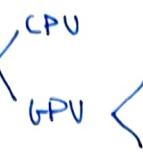
Displacement mapping: subdividir polígons + desplaçar vèrtexs

Displacement Mapping

- Subdividir caras \rightarrow 

- Desplazamiento vértices $\rightarrow P'(u,v) = P(u,v) + F(u,v) \cdot N(u,v)$



- Es pot fer a: 
 - CPU
 - GPU

Geom. Shader

TCS (Subdiv) + TES (Displac.)

Bump mapping

En aquest cas no fa subdivisió ni desplaçament de vèrtexs.

$$P'(u,v) = P(u,v) + F(u,v) \cdot N(u,v)$$

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

$$T = \frac{\partial P}{\partial u}$$

$$N \times T \equiv B$$

$$N' = N - \frac{\partial F}{\partial u} \cdot T - \frac{\partial F}{\partial v} \cdot B$$

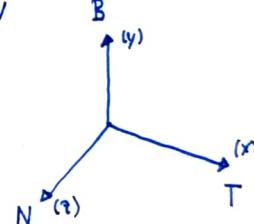
(0,0,1)
comp x comp y

$$B = \frac{\partial P}{\partial v}$$

$(T, B, N) \rightarrow$ defineixen el "tangent space"

* Calcul T, B, N en video "Disp/Bump/Normal Mapping"

$$N' = \left(-\frac{\partial F}{\partial u}, -\frac{\partial F}{\partial v}, 1 \right)$$



Matrix Tangent Space \rightarrow Eye Space

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 \\ T & B & N \\ 1 & 1 & 1 \end{bmatrix}}_{\text{vectors en eye space}} \cdot \begin{bmatrix} 1 \\ N' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ N' \\ 1 \end{bmatrix}$$

En tangent space

Normal mapping \rightarrow 1 canal

En lloc de guardar $F(u,v)$

guarda: \rightarrow 3 canals

normalize($\nabla F.x, \nabla F.y, 1$)

* $N' = \text{texture(normalmap, (s,t))}$

$$\epsilon = \frac{1}{\text{mida textura}}$$

$$F = \text{texture(bumpmap, (s,t))} \cdot x$$

$$F_x = " " (s + \epsilon, t) \cdot x ; F_y = " " (s, t + \epsilon) \cdot x ;$$

$$\nabla F = (F_x - F, F_y - F) / \epsilon \cdot \text{scale} ; N' = (\nabla F.x, \nabla F.y, 1) ;$$

$$N' = [T \ B \ N] [N']$$

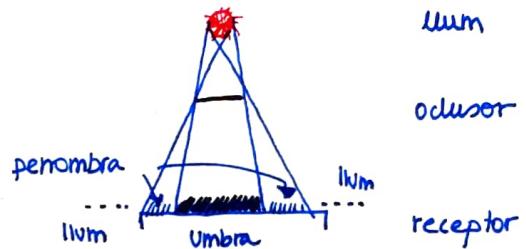
$$\text{eye } 3 \times 3$$

$$\text{frag color} = \text{phong}(N', \dots)$$

TEMA 7. OMBRES

Umbra: Absencia de illum.

Penombra: Gradient entre llum i foscor



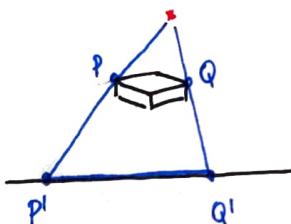
Font de llum puntual → no hi ha penombra

Augmenta mida de font de llum \rightarrow + penombra - umbra

Apropem oclusor i receptor → - penombra

OMBRES PER PROJECCIÓ

Sense Stencil Buffer. Receptor es pla. Llum puntual



1. Dibuixar
 2. Dibuixar oclusor projectat (ombra)
 3. Dibuixar oclusor

Evitar problemas de z-fighting (Window space) depèn indinacions polígon

of Polygon Offset (factor, units) ; $\Rightarrow z' = z + r \cdot \text{units} + dz \cdot \text{factor}$

En codi:

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

```
glEnable(GL_POLYGON_OFFSET_FILL);
```

```
gl_PolygonOffset(factor, units);
```

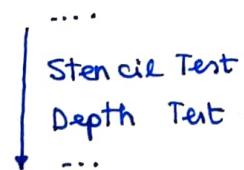
```
glDisable(GL_POLYGON_OFFSET_FILL);
```

Problema : Ombra més gran que el receptor.

STENCIL BUFFER

Creació d'una màscara.

En : 8. Per fragment operations →



L'ordre
és important

En : 9: Frame buffer operations → Modifica els buffers existents per glDrawBuffers.

Afectada per { glColor Mask
gl Depth Mask.

Quina info guarda?

Per cada pixel, un enter entre 0 i $2^n - 1$

• glStencilFunc (comp, valRef, mask);

comp: GL_NEVER, GL_ALWAYS, GL_LESS

↳ valRef & mask < valSten & mask

• glStencilOp (fail, zFail, zPass)

fail, zFail, zPass: corren els possibles estats dels tats. Poden tenir valors:

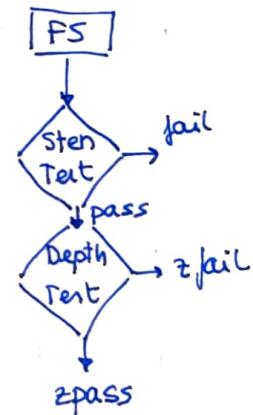
GL_KEEP / ZERO / INCR / DECR / INVERT

GL_REPLACE

Ex:

glStencilFunc (GL_ALWAYS, 1, 1)

glStencilOp (GL_KEEP, GL_KEEP, GL_REPLACE)



**TOMARSE UN
CAFELITO YA HECHO
FEELS LIKE...**

**BUSCAR EL RESUMEN EN INTERNET
EN VEZ DE LEER EL LIBRO.**

Ombres per projecció. Amb stencil buffer

- Dibuixar receptor al color buffer i al stencil buffer

Targets: Color, Z, Stencil

- Dibuixar ocultor per netejar l'stencil a les zones d'ombra.

Targets: Stencil

- Dibuixar la part fosca del receptor

Targets: Color, Z

→ glDepthFunc(GL_LESS)

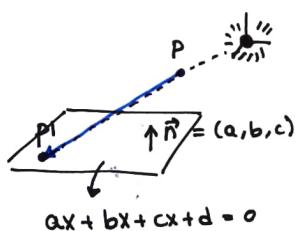
- Dibuixa l'ocultor

Evita els problemes de z-fighting

Targets: Color, Z

Ombres per projecció. Matrizes de projecció

- Llum puntual a l'origen



$$\left. \begin{array}{l} P^I \in \text{plano} \rightarrow a(P_x^I) + b(P_y^I) + c(P_z^I) + d = 0 \\ P^I \in \text{rayo} \rightarrow P^I = \lambda \cdot P \quad \lambda > 1 \end{array} \right\}$$

$$P^I = \frac{-d}{\vec{n} \cdot \vec{P}} \cdot \vec{P}$$

Direcció: \vec{OP}
(del raig de llum)

$$\begin{bmatrix} -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & -d & 0 \\ a & b & c & 0 \end{bmatrix} \cdot \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} -dP_x \\ -dP_y \\ -dP_z \\ ap_x + bp_y + cp_z \end{bmatrix}$$

- Llum en un punt F

Direcció: \vec{FP}

- Llum direccional (infinitament allunyada en direcció L)

Direcció: $\vec{-L}$

* Matrizes als apunts de l'assig

100% INGREDIENTES NATURALES



VOLUMS D'OMBRA

Receptor pot ser un volum (en ombr. per projecció havia de ser pla)

L'oculador ha de ser senzill

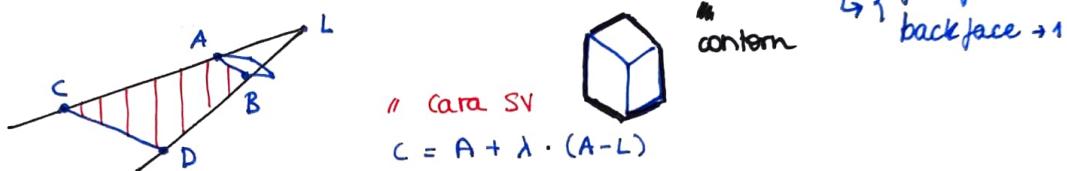
Llum puntual

Shadow Volume = SV

Punts de l'espai entre l'oculador i la superfície.

* Generació SV (Obj, light)

1. Arestes contorn (a contorn \equiv cares que comparteixen aresta)



2. Construir cares de SV

* Problema punt - interior - polígon



$P + \lambda \cdot \vec{v}$ $\lambda > 0$ llançar un raig en qualsevol direcció

intersec. en impar parell \rightarrow P dins

intersec. en parell \rightarrow P. fora

* Problema punt - interior - poliedre \rightarrow Igual que en 2D

front - # back $> 0 \Rightarrow P \in SV$

Passos a seguir

1. Dibuixa escena al z-buffer \rightarrow suma zones amb sombra

2. Dibuixa al stencil les cares frontals del volum (SV)

3. Dibuixa al stencil les cares posteriors del volum (SV)
 \rightarrow resta zones redadera de sombra

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

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

6. Restaurar estat inicial

SHADOW MAPPING

Receptor i ombra arbitraris. Llum puntual.

Shadow mapping: situem càmera al punt de la llum. Tots els fragments que no es vegin des d'aquesta càmera estan a la ombra.

Passos a seguir:

1) Dibuixar l'escena (des de la càmera a la pos. de la llum)

Guardar depth buffer \rightarrow Shadowmap,
textura
`glClear()`

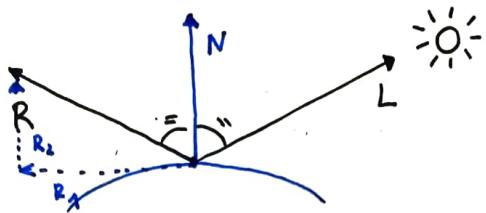
2) Dibuixar l'escena

Shadow mapping

$\left\{ \begin{array}{l} \text{VS: } T(0 s) \cdot S(0 s) \cdot P \cdot V \cdot M \\ \text{FS: shadowMap } (s/q, t/q) \Rightarrow storedDepth \\ \quad p/q \Rightarrow trueDepth \\ \quad trueD \approx storedD \Rightarrow llum \\ \quad trueD \neq storedD \Rightarrow ombra \end{array} \right.$	$\begin{matrix} \text{light} & \downarrow & \text{obj} & \downarrow \\ \downarrow & & \downarrow & \\ \text{P.V.} & & \text{M.} & \\ \end{matrix}$ $\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} s \\ t \\ p \\ q \end{bmatrix}$	$\rightarrow q: \text{dir. persp}$ $(s/q, t/q)$ En un punt a , tenim guardat za . $p/q \rightarrow z$ en W Space $za \approx p/q$ (si està il·luminat)
---	--	---

* cal incloure offsets, si no es produeixen artifacts.

TEMA 8. REFLEXIONS ESPECULARS

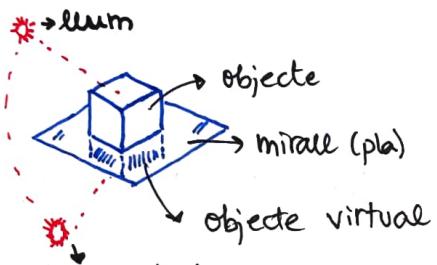


$$\left. \begin{aligned} R &= R_1 + R_2 \\ R_1 &= -L + R_2 \\ R &= 2R_2 - L \\ R_2 &= (N \cdot L) \cdot N \end{aligned} \right\} R = 2(N \cdot L) \cdot N - L$$

Mètodes per calcular reflexions:

- Ray tracing
- Objectes virtuals → Mirall pla. Reflexió exacta
- Environment mapping → Mirall arbitrari. Reflexió aprox.

REFLEXIONS BASADES EN OBJECTES VIRTUALS



(al canviar el comportament
de g_CULLF en el pas 1
i el 2 i 3.

També cal reflexar la llum en relació al pla del mirall.

- Dibuixar l'objecte 2 vegades.
 1 → 1 vegada rotat respecte l'eix del
mirall (mult. -1 → inversa + translació)
 2 → Dibuixar el mirall semitransparent
 3 → Dibuixar objecte(s) en posició real

Limitacions

- S'assumeix que no hi ha objectes a l'altra banda del mirall.
 ↳ 1 solució : Utilitzar stencil test (una māscara)

saboteas a tu propia persona? cómo?? escríbelo **aquí** y táchalo

manual de instrucciones: escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)



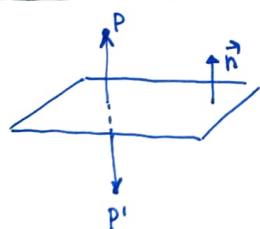
REFLEXIÓ AMB STENCIL TEST

		Stencil	Test z	Esat
<u>Pas 1</u>	Dibuixar mirall al stencil buffer	op 1	True	X X
<u>Pas 2</u>	Dibuixar objectes virtuels	$\neq 1$	✓	✓
<u>Pas 3</u>	Dibuixar mirall semi transparent		✓	✓
<u>Pas 4</u>	Dibuixar objectes reals		✓	✓

REFLEXIÓ AMB TEXTURES DINÀMIQUES

- Pas 1 Dibuixar escena pos. virtual
- Pas 2 Crear textura amb color buffer
- Pas 3 Dibuixar el mirall amb la textura del pas 1
- Pas 4 Dibuixar l'objecte real

MATRÍU DE REFLEXIÓ



$$ax + by + cz + d = 0$$

$$d = \text{dist}(P, \text{plano}) = ap_x + bp_y + cp_z + d$$

$$P' = P - 2 \cdot d \cdot \vec{n}$$

$$P'_x = p_x - 2 \cdot (ap_x + bp_y + cp_z + d) \cdot a$$

$$P'_y = (1 - 2a^2) \cdot p_x, -2ab \cdot p_y, -2ac \cdot p_z - 2ad$$

$$\begin{bmatrix} 1 - 2a^2 & -2ba & -2ca & -2da \\ -2ba & 1 - 2b^2 & -2cb & -2db \\ -2ca & -2cb & 1 - 2c^2 & -2dc \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix}$$

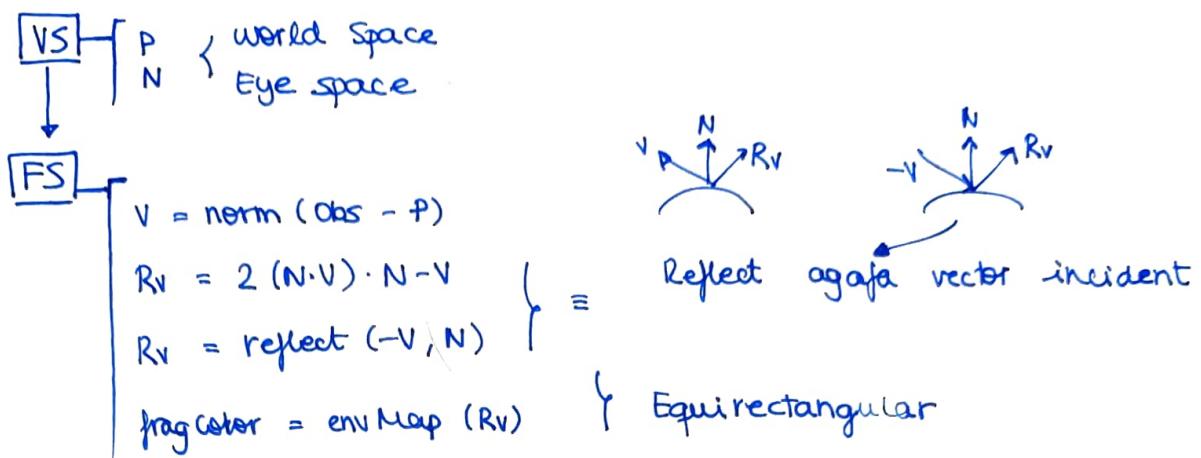
$$\hookrightarrow I_4 - 2 \cdot \begin{bmatrix} a & b & c \\ 0 & 0 & 0 \end{bmatrix} \cdot [a \ b \ c \ d]$$

ENVIRONMENT

MAPPING

Environment map: Dada una dirección R , retorna un color.

Textura representada en: Equirectangular, Sphericalmap, Cubemap,...



$$R = \text{norm}(P - \text{obs})$$

$$\text{fragColor} = \text{envMap}(R)$$

Sphere Map

$$R \rightarrow (\mu, v) \rightarrow (s, t)$$

\Leftarrow

$$V = (0, 0, 1)$$

$$R_v = 2(N \cdot V)N - V$$

$$R_v = 2n_z(n_x, n_y, n_z) - (0, 0, 1) = \left(\frac{r_x}{2n_z}, \frac{r_y}{2n_z}, \frac{r_z}{2n_z^2 - 1} \right)$$

$$P = (p_x, p_y, p_z) = N = (n_x, n_y, n_z)$$

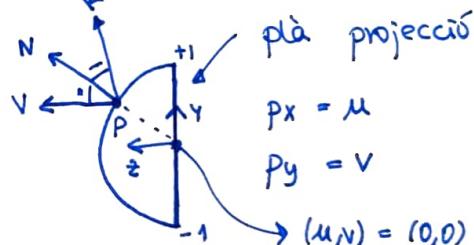
$$p_x = \frac{r_x}{2n_z} \rightarrow \boxed{\mu} \quad p_y = \frac{r_y}{2n_z} \rightarrow \boxed{v} \quad p_z = \sqrt{\frac{r_z + 1}{2}}$$

$$(\mu, v) \rightarrow (s, t)$$

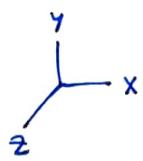
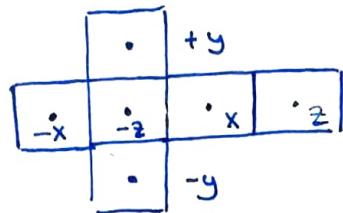
Ver al video: minut 55.

$$[-1, 1] \rightarrow [0, 1]$$

$$\hookrightarrow +1 : 2 \rightarrow$$



Cube mapping



GL_TEXTURE_CUBE_MAP

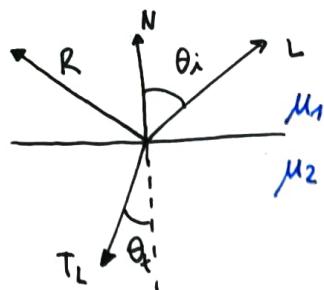
Es creen 6 textures

uniform samplerCube samplerC;

vec3 R;

fragcolor = textureCube(samplerC, R);

TEMA 9: OBJECTES TRANSLÚCIDS



$$\theta_t = f(\theta_i, \mu_1, \mu_2)$$

↳ Depèn també de la longitud d'ona.

Llei d'Snell:

$$\sin(\theta_t) = \frac{\mu_1}{\mu_2} \cdot \sin(\theta_i)$$

Angle crític: $1 = \frac{\mu_1}{\mu_2} \cdot \sin(\theta_i) \rightarrow \theta_i = \theta_{\text{critic}} = \arcsin\left(\frac{\mu_2}{\mu_1}\right)$

Equacions de Fresnel:

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

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

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

Com més angle d'entrada, més reflexió i menys transmissió.

Aproximació de Schlick

$$R = f + (1-f)(1-LN)^5 \quad f = \frac{(1-\mu)^2}{(1+\mu)^2}$$

Útil perquè no s'ha de calcular ni el cos ni el sin.

Ordre de pintat dels polígons

Ordenat: Back-to-Front \rightarrow primer dibuixar els objectes $\rightarrow (rs, gs, bs, as)$

del fons.

Alpha blending

$$\text{color} = s\text{factor} \cdot \text{src-color} + d\text{factor} \cdot \text{dst-color}$$

glEnable(GL_BLEND);

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)

sfactor
as

dfactor

(1-as)

GL_ONE

Es pot prescindir del z-buffer.

$$\text{color} = as \cdot \text{src-color} + dst-color$$

saboteas a tu propia persona? cómo?? escríbelo **aquí** y táchalo

manual de instrucciones: escribe sin filtros

y una vez acabes, táchalo (si lo compartes en redes mencionándonos, te llevas 10 coins por tu cara bonita)

Options

- ① Dibuir tot ordenat Back-to-Front, depth-test → indiferent
no és eficient

② Dibuir opacs sense ordenar } depth test ✓
Dibuir transparents ordenats } depth write ✓

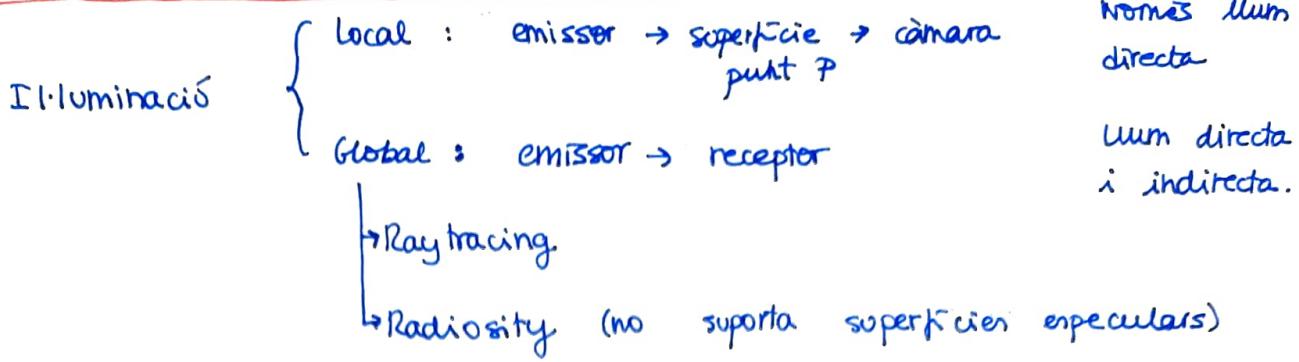
③ Dibuir opacs sense ordenar } depth test ✓
Dibuir transp sense ordenar } depth write ~ indif.

④ → correcte, no eficient

⑤ → correcte, més eficient que ④

⑥ → més eficient, depenen de l'ordre no serà 100% correcte.

TEMA 10. IL·LUMINACIÓ GLOBAL



RADIOMETRIA I FOTOMETRIA

Flux radiant

Quantitat d'energia emesa en una unitat de temps

Unitats: W o lm (lúmens)

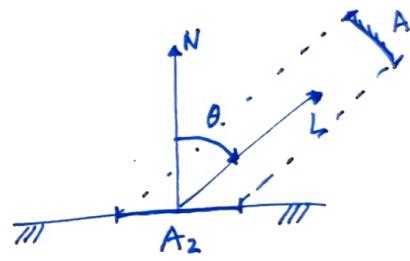
Us: E total que emet una llum.

Irradiància

Densitat de flux = Flux per àrea

Unitats: W/m² lm/m²

Us: llum que incideix en un punt



$$A_2 = \frac{A}{N \cdot L} \rightarrow E = \frac{\Phi}{A} \cdot (N \cdot L)$$

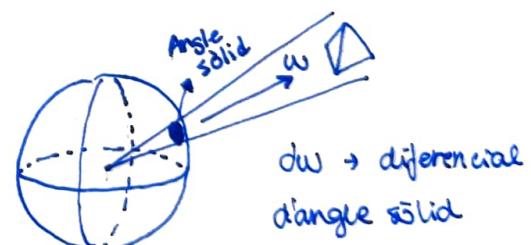
$$N \cdot L = \cos(\theta)$$

Intensitat

Flux / unitat d'angle sòlid

Unitats: $\frac{W}{sr}$, $\frac{lm}{sr} = cd$ \rightarrow sr [0, 4π]

Us: Distribució direccional d'una llum puntual.



$d\omega \rightarrow$ diferencial d'angle sòlid

Radiància

Flux per unitat d'àrea i unitat d'angle sòlid

Unitats: $\frac{W}{sr \cdot m^2}$ $\frac{lm}{sr \cdot m^2}$

$$L = \frac{d^2\Phi}{d\omega dA}$$

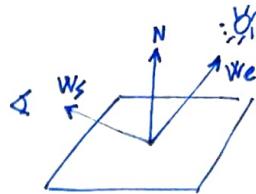
Us: energia que travessa un punt en una determinada direcció.

Tipus: Emissió $\rightarrow L_e(p, w)$; Incident $\rightarrow L_i(p, w)$; Saliente $\rightarrow L_o(p, w)$



BRDF

$f(p, w_o, w_i)$



Donat p , w_o , w_i , quina és la quantitat de llum que en veu.

- $f_r(p, w_o, w_i) \geq 0$
- $f_r(p, w_o, w_i) = f_r(p, w_i, w_o)$
- $\int_{\Omega} f_r(p, w_o, w') \cdot \cos \theta \cdot d\omega' \leq 1$ (Conservació d'energia)

EQUACIÓ GENERAL DEL RENDERING

Kajiya 1986,

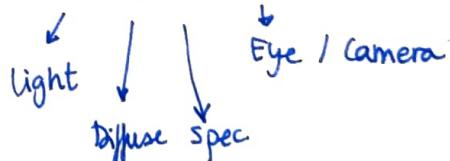
$$L_o(p, w_o) = L_e(p, w_o) + \int_{S^2} \underbrace{f(p, w_o, w_i)}_{\text{rad. sortida}} \cdot \underbrace{\underbrace{L_i(p, w_i) \cdot \cos \theta \cdot d\omega_i}_{\text{Radiància}}}_{\text{rad. emesa}} \rightarrow \begin{matrix} R \\ T \end{matrix} \xrightarrow{\text{BSDF}} \text{Reflexió + Transmissió}$$

LIGHT PATHS

Interessen els camins del

tipus $L(D|S)^*$

Serveixen per donar una solució al problema de la il·luminació global



Phong \rightarrow considerem només $LDE + LSE$ (només 1 nivell, llum directa)
LE no la volem

TEMA II. RAY TRACING

RAY - CASTING IL·L. local

Per cada pixel, calcula el color (sense il·luminació global) llançant un raig. Aquesta versió no és recursiva.

VARIANTS DE RAY TRACING IL·L. global

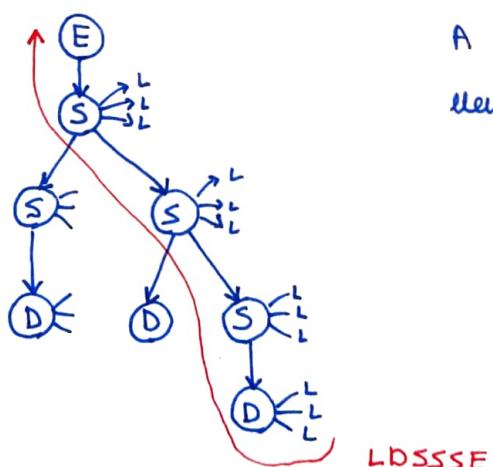
Ray Tracing Clàssic

Rajos llançats de l'obs a escena (oposat a sentit de la llum)

Impactes en superfície : hits

Recursiu ?

- Material difús : No. Il·luminació local en el punt
Light-Ray / Shadow-Ray a cada font de llum.
- Material espejular : Si, fins arribar a cas base (difús)
Poden ser translúcids o miralls.



A cada pas també cal llançar els light/shadow-rays

Camins que suporta RT : LDS^*E

**TOMARSE UN
CAFELITO YA HECHO
FEELS LIKE...**

**HACERSE CHULETAS
EN LA CALCULADORA.**

Path Tracing

Per cada pixel en llancen "r" raiges. A cada rebot



Camins que supera: $L(DIS)^* E$

cal limitar la profunditat.

A cada hit es calculen "x" rebots.

TE problemen de seroll degut a la variància dels raiges rebotats.

Distributed ray-tracing

Quan una sup és especular, calculem rans raiges per cada punt.

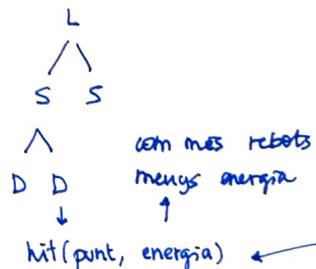
En una sup difusa ja no fa mes càlculs recursius.

Camins: $L D S^* E$

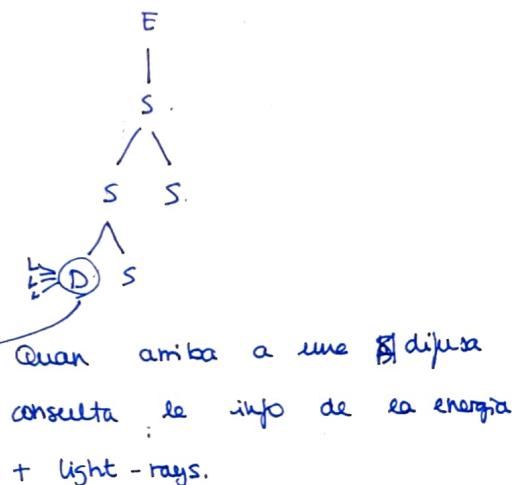
Two-pass ray-tracing

Pas 1 - Light pass

Raiges dellum dar de light fins component difusa:



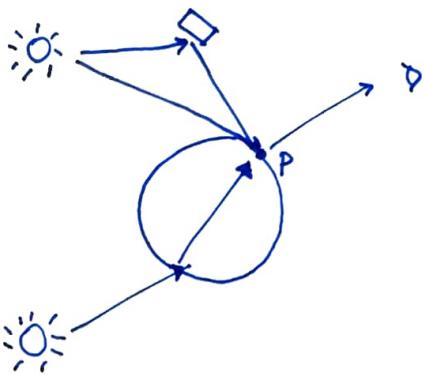
Pas 2 - Eye pass



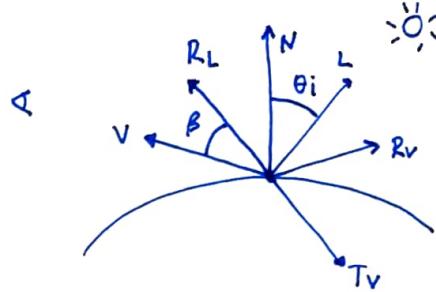
100% INGREDIENTES NATURALES



RAY TRACING CLASSIC



$$I(P) = I_{\text{dir}}(P) + I_{\text{ref}}(P) + I_{\text{tra}}(P)$$



$$I_D(P) = K_a \cdot I_a + K_d \cdot \sum I_L \cdot \underbrace{\cos(\theta_i)}_{N \cdot L} + K_s \cdot \underbrace{\cos^n(\beta)}_{(R_L \cdot V)^n}$$

$$I_R(P) = K_R \cdot L_R$$

↳ calcular recursivamente

$$I_T(P) = K_T \cdot L_T$$

INTERSECCIÓN RAYO - TRIÁNGULO

Input:

$$\begin{cases} \text{Rayo: } p + \lambda \cdot \vec{v} \\ \text{Triángulo: } A, B, C \in \mathbb{R}^3 \end{cases}$$

$$\vec{n} = (a, b, c)$$

$$\text{plano: } ax + by + cz + d = 0$$

① $I \in \text{rayo} \rightarrow$

$$I = p + \lambda \cdot \vec{v} \rightarrow$$

$$ax + by + cz + d = 0$$

depende de λ :

$$\lambda = \frac{-\text{dist}(p, \text{plano})}{\vec{n} \cdot \vec{v}}$$

Output:

$$\begin{cases} \text{hit} \\ \text{punto int: } I \in \mathbb{R}^3 \\ \text{atrib en } I \end{cases}$$

② Determinar $I \in \Delta A, B, C$

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

$$\alpha = \frac{\text{area}(I, B, C)}{\text{area}(A, B, C)}$$

$$\beta = \dots$$

$$\gamma = \dots$$

$$\begin{aligned} &\text{Pertenece si } \alpha \geq 0 \\ &\quad \beta \geq 0 \\ &\quad \gamma \geq 0 \end{aligned}$$

$\vec{n} \cdot \vec{v} \neq 0 \rightarrow \text{calculo } \lambda \rightarrow$
hay int entre recta y
plano \rightarrow comprobar que
rayo intersecte ②

$\vec{n} \cdot \vec{v} = 0 \rightarrow$ recta y planos
son paralelos
 $\left. \begin{array}{l} \text{rayo en plano} \\ \text{no hay intersec.} \end{array} \right\}$