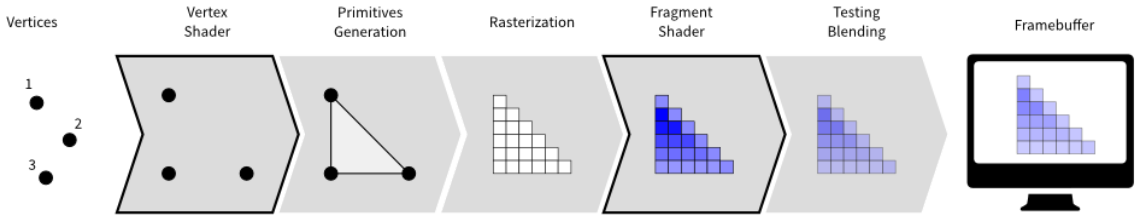
S1-VS-FS-Viewer

José Luis Pontón jose.luis.ponton@upc.edu

Se ejecuta el VS una vez por cada vértice



Espai → Menu
Shift + Click → Zoom
SHADERS
in

Atributs definits pel viewer (cal declarar-los al VS):

```
layout(location= 0) in vec3 vertex;    // object space
layout(location= 1) in vec3 normal;    // object space; unitària
layout(location= 2) in vec3 color;     // color en RGB; valors en [0,1]
layout(location= 3) in vec2 texCoord;  // coordenades de textura (s,t)
```

out

Output variables (pel VS són de sortida; pel FS són d'entrada):

- **out vec4 gl_Position** (predeclarada; habitualment en clip space)
- Qualsevol altre definida per l'usuari. Exemples: color, coordenades del vèrtex, normal, coordenades de textura...

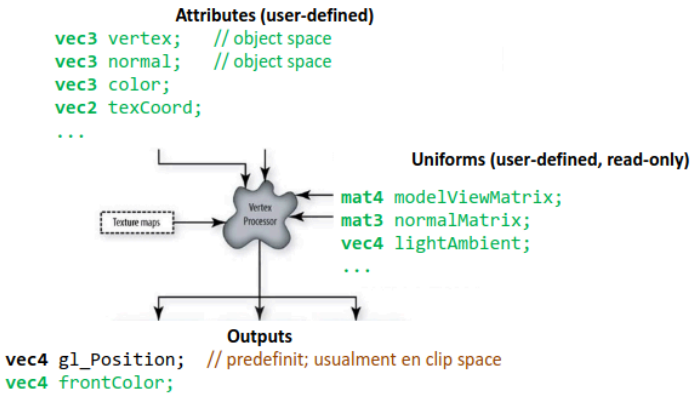
uniform

Variables uniform que envia el viewer (cal declarar-les):

```
uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;
uniform mat4 modelViewMatrix;
uniform mat4 modelViewProjectionMatrix;

uniform mat4 modelMatrixInverse;
uniform mat4 viewMatrixInverse;
uniform mat4 projectionMatrixInverse;
uniform mat4 modelViewMatrixInverse;
uniform mat4 modelViewProjectionMatrixInverse;

uniform mat3 normalMatrix;
```

n'hi ha molts….

VS (3.3 core profile)



Vertex Shader:

```
void main()
{
        vec3 N = normalize(normalMatrix * normal);
        frontColor = vec4(color,1.0) * N.z;
        vtexCoord = texCoord;
        gl_Position = modelViewProjectionMatrix * vec4(vertex, 1.0);
}
```

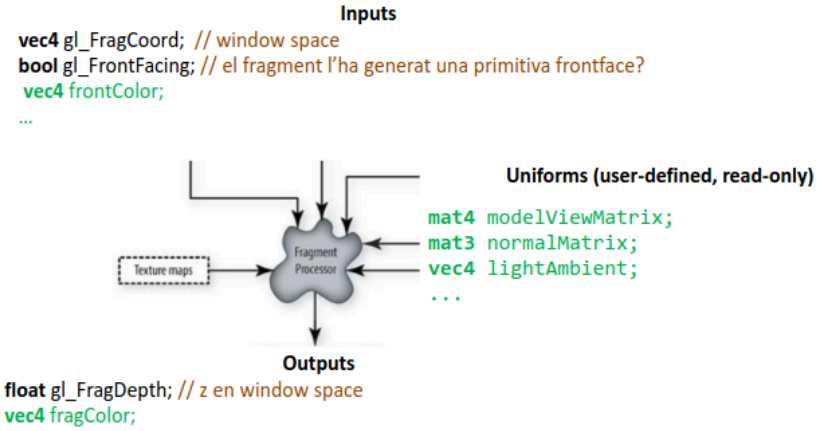
Passar d'object a eye space:

```
vec3 N = normalize(normalMatrix * normal);
```

Eye space a clip space:

```
gl_Position = modelViewProjectionMatrix * vec4(vertex, 1.0); //punto 1.0, vector 0.0
```

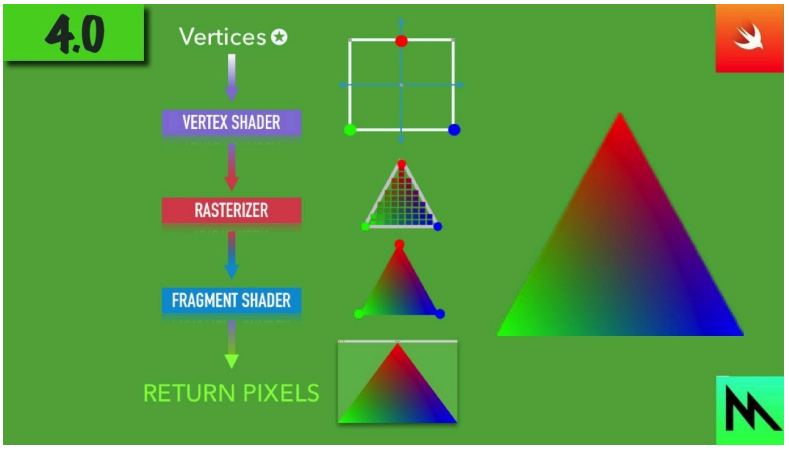
FS (3.3 core profile)



Fragment Shader:

```
#version 330 core

in vec4 frontColor;
out vec4 fragColor;
```

```
void main()
{
	fragColor = frontColor;
}
```



GLSL

Tipus bàsics

Escalars

`int, float, bool`

Vectorials

`vec2, vec3, vec4, mat2, mat3, mat4, ivec3, bvec4,...`

Constructors

Hi ha *arrays*: `mat2 mats[3];`
i també *structs*:

```
1    struct light{
2      vec3 color;
3      vec3 pos;
4    };
```

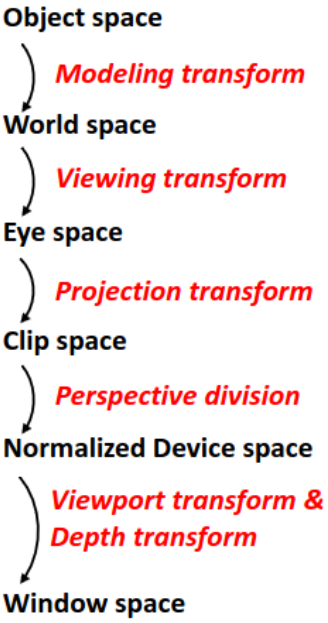
que defineixen implícitament constructors: `light l1(col,p);`

Fer servir funcions:

N'hi ha moltes, especialment en les àrees que poden interessar quan tractem geometria o volem dibuixar. Per exemple, `radians()`, `degrees()`, `sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()` (amb un o amb dos paràmetres), `pow()`, `log()`, `exp()`, `abs()`, `sign()`, `floor()`, `min()`, `max()`, `length()`, `distance()`, `dot()`, `cross()`, `normalize()`, `noise1()`, `noise2()`, ...

Bíblia (funcions en lila)
https://www.cs.upc.edu/~virtual/G/2.%20Laboratori/Part%201%20-%20Shaders/OpenGL-3-2-quick-reference-card.pdf

Sistemes de coordenades i matrius




```
uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;
uniform mat4 modelViewMatrix;
uniform mat4 modelViewProjectionMatrix;

uniform mat4 modelMatrixInverse;
uniform mat4 viewMatrixInverse;
uniform mat4 projectionMatrixInverse;
uniform mat4 modelViewMatrixInverse;
uniform mat4 modelViewProjectionMatrixInverse;

uniform mat3 normalMatrix;
```

Clip space -> NDS (es divideix per la w de clip space)

Modeling transforms

translate(t_x,t_y,t_z) $$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

scale(s_x,s_y,s_z) $$T = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotate(a,x,y,z) $$T = \begin{bmatrix} x^2d + c & xyd - zs & xzd + ys & 0 \\ yxd + zs & y^2d + c & yzd - xs & 0 \\ xzd - ys & yzd + xs & z^2d + c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

c=cos(a), s=sin(a), d=1-cos(a)

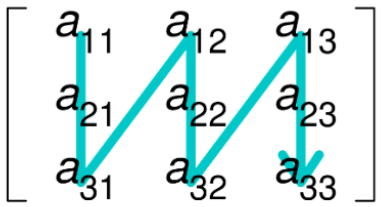
(per moure un punt podem sumar un vector, o per escalar multiplicar, no cal usar sempre matrius).

glRotate*$(a, 1, 0, 0)$: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos a & -\sin a & 0 \\ 0 & \sin a & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

glRotate*$(a, 0, 1, 0)$: $\begin{bmatrix} \cos a & 0 & \sin a & 0 \\ 0 & 1 & 0 & 0 \\ -\sin a & 0 & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

glRotate*$(a, 0, 0, 1)$: $\begin{bmatrix} \cos a & -\sin a & 0 & 0 \\ \sin a & \cos a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

mat3 m = **mat3**(**vec3**(1,0,0), **vec3**(0,1,0), **vec3**(0,0,1));
// els tres vectors són les **columnes** de la matriu



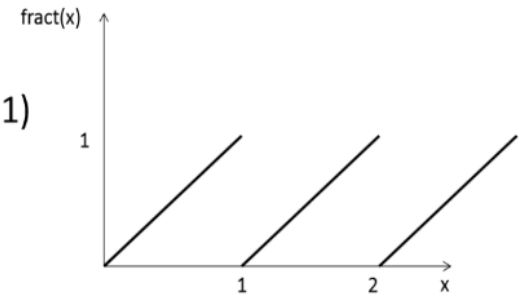
funcions

Funcions GLSL – **fract(x)**

Retorna la part fraccionària de **x**, calculada com

x - floor(x)

- Domini: R^n
- Recorregut: [0, 1)
- Període: 1

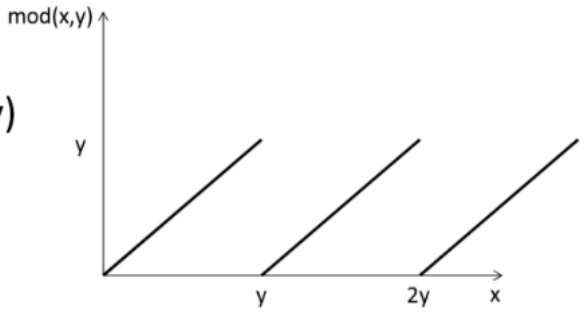


Funcions GLSL – **mod(x,y)**

Retorna **x mòdul y**, calculat com

$$x - y * \text{floor}(x/y)$$

- Domini: R^n
- Recorregut: [0, y)
- Període: y



IMPORTANT (Interpolació lineal entre 2 vectors)

Funcions GLSL – **mix(a,b,t)**

Retorna la interpolació linial entre a i b ponderada per t:

$$a(1-t)+bt \;=\; a + t(b-a)$$

- Habitualment **t** és un escalar en [0,1].
- Els paràmetres **a**, **b** poden ser vectorials (en aquest cas la interpolació es fa per components):

  mix(vec3(1,0,0), vec3(0,1,0), 0.5) → retorna vec3(0.5,0.5,0)

Funcions GLSL – **sin(x)**

Retorna el sinus de x (en radians).

És freqüent usar sinusoïdals de la forma:

$$A * \sin(2\pi * f * t + \Theta)$$

A = amplitud

f = freqüència; el factor 2π apareix només si volem que freq estigui en Hz

t = temps (en segons)

Θ = fase; si per exemple Θ = {0, π/2, 3π/2}, llavors per t=0 la sinusoïdal serà {0, A, -A}

const float PI 3.141592;

Animacions als shaders

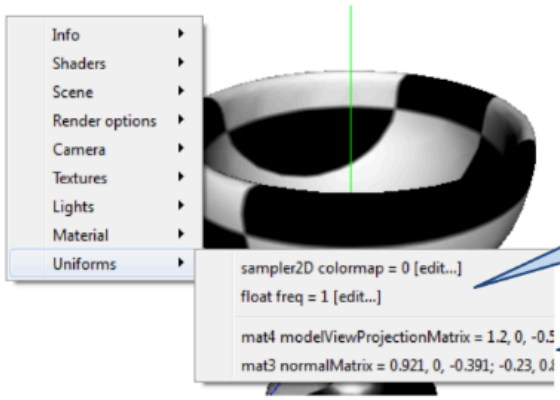
```
uniform float time;
const float PI = 3.141592;

void main()
{
 fragColor = vec4(0.5*(sin(2*PI*time)+1.0));
}
```

Temps (segons) des de la darrera compilació.

User-defined uniforms

```
uniform float freq=2.0; // frequencia en Hz
void main()
{
 fragColor=vec4(.5*(sin(2*PI*freq*time)+1.0));
}
```

Uniform definit per l'usuari; convé donar-li un valor.



Uniforms definits per l'usuari: el viewer permet editar-los (actualment limitat a bool, int, float)

Uniforms definits pel viewer (el menu no en permet l'edició)

Test, última columna es podria ignorar. Tenir la penúltima blanca no implica que estigui bé.

Exercicis:
https://www.dropbox.com/scl/fo/mfw15m5ywzwfqyaxza6ei/AD8gwLepAvER9T4tPMYbuRA?rlkey=xy6e18vmqtop2cmhja4nwser4&e=1&dl=0

EQUACIO ONA

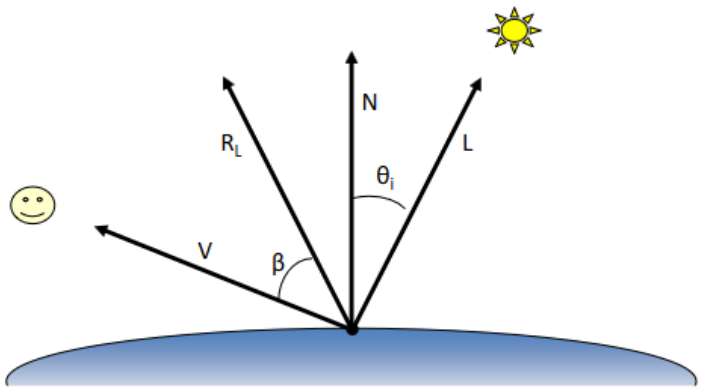
$$\delta = amplitude * sin(2 \cdot \pi \cdot f \cdot t + fase)$$

ELEMENTS USATS

- mix (interpolació lineal)
- smoothstep (interpolació suau amb S-curve)

S2-Il·luminacio

PHONG



$$K_e + K_a(M_a + I_a) + K_d I_d(N \cdot L) + K_s I_s(\mathbf{R} \cdot \mathbf{V})^s$$

Només si N·L>0 Només si N·L>0

- K_* = material
- I_* = llum

Emissió de l'objecte → $K_e = 0$

AMBIENT $K_a(M_a+I_a)$ → **Llum difusa general a l'escena**

K_a →Coeficient de reflectància ambiental de la superficie del material.
$M_a = 0$ → Llum ambiental present a l'escena.
I_a → intensitat de la llum ambiental que arriba a l'objecte.

DIFÚS $K_d I_d(N \cdot L)$ → **Llum que incideix sobre la superfície i es dispersa**

K_d →Coeficient de reflectància difusa del model per reflectir la llum de manera dispersa.
I_d → Intensitat de la llum difusa que arriba a la superfície
N → Normal del vèrtex
L → Vector del punt a la llum
$N \cdot L$ →Producte escalar . Determina quina quantitat de llum es reflecteix en funció de l'angle entre la superfície i la direcció de la llum. Màxim en superfícies perpendicular llum.

ESPECULAR $K_s I_s (R \cdot V)^S$ **→Reflex**

K_s →Coeficient de reflectància especular del model, intensitat.

I_s → Intensitat de la llum especular que arriba a la superfície

R → Reflexió de la llum sobre la superfície (reflex del vector L respecte N)

V → Vector del punt a l'observador

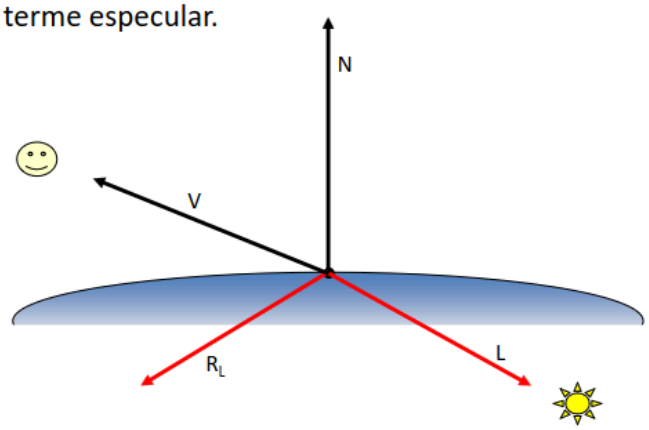
$(R \cdot V)^S$ →Intensitat especular de la taca especular.

S → Shiness del material

Notació

Si N·L < 0:

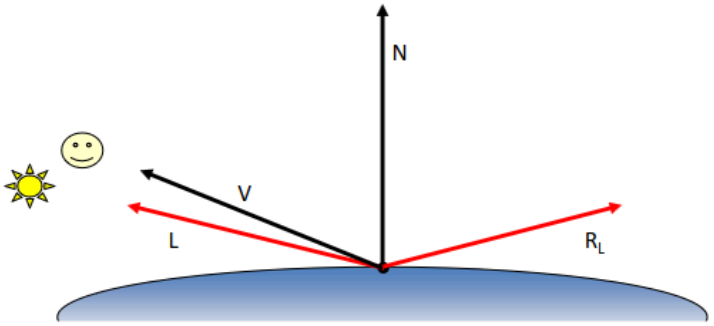
- Cal evitar que la contribució difosa "resti" llum. Useu per exemple max(0, ...)
- Cal ignorar el terme especular.



Notació

Si R·V < 0:

- Cal evitar que la contribució especular "resti" llum. Useu per exemple max(0, ...)



EXERCICIS:
https://www.dropbox.com/scl/fo/ekyp70sa7rge94a2x1g6s/APsdetuNLNe04BXVqI7XxWA?rlkey=5yocedhi9fz17c8k7bw105dr9&e=1&dl=0

Obligatoris

Lighting 2

Lighting 4
Lighting 5
Color Gradient 1
Color Gradient 2
Reverse Z 1
Reverse Z 2
Zoom
Uncover

Opcionals:

Nlights

S3-Textures

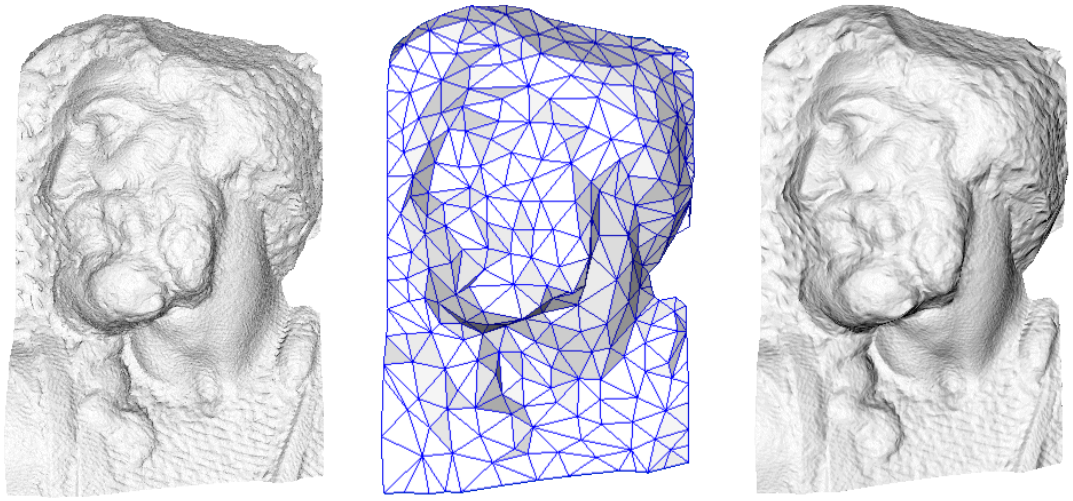
- Basic Texture
- Animate Texture
- Tiling
- Explosion
- UV unfold

> Observeu que el punt que ha d'escriure el VS a gl_Position no dependrà ni de la matriu **modelViewProjection**, ni de **vertex**, sinó de **texCoord**.

- Texture splatting

Opcionals:
- lightchange (textura de prova: Combo-lite.png, a la mateixa carpeta de l'enunciat)
- Gioconda (*) L'enunciat original tenia un error. Ara al dropbox teniu la versió correcta.

S4-Textures Procedurals



original mesh
4M triangles

simplified mesh
500 triangles

simplified mesh
and normal mapping
500 triangles

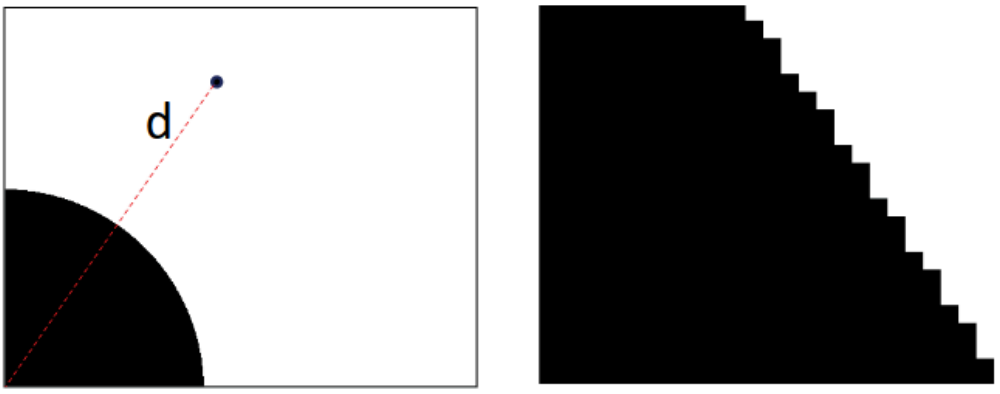
Texturas procedurales (SIEMPRE ENTRA EN EL EXAMEN)→ No tenemos imagen en memoria, sino que de forma matemática modificamos el modelo a partir de texturas.

Manera alternativa de fer una esfera

STEP → Brusc

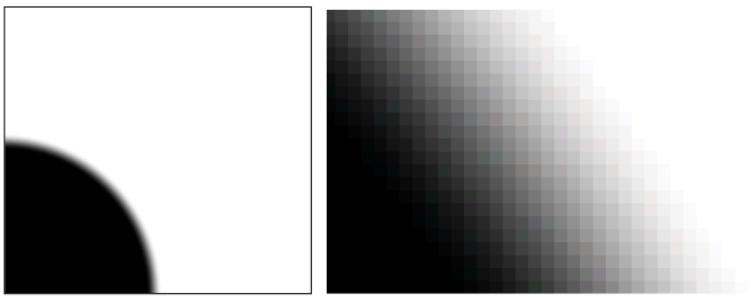
Exemple - step

```
void main() {
  float d = length(gl_FragCoord.xy);
  gl_FragColor = vec4(step(200, d));
}
```



Alternativament, suavitzant “Antialiasing”

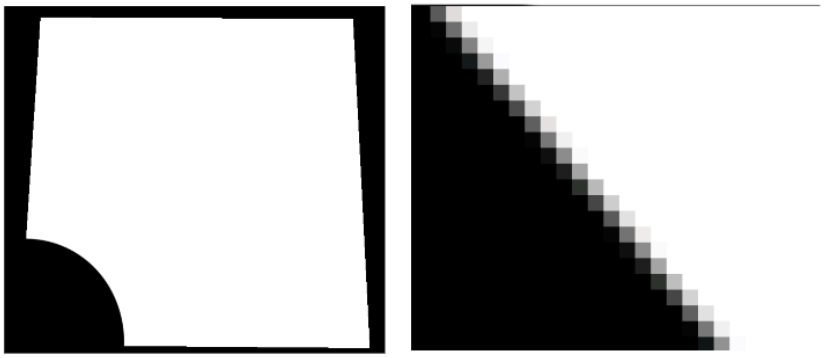
```
void main() {
  float d = length(gl_FragCoord.xy);
  gl_FragColor = vec4(smoothtep(200-10,200+10, d));
}
```



La derivada nos dice como cambia una variable en un píxel

Exemple 2 – smoothstep + dFdx,dFdy

```
float width = 0.5*length(vec2(dFdx(d), dFdy(d)));
gl_FragColor=vec4(smoothstep(r-width, r+width, d));
```



Entre 1 y 1.4

aastep (*)

```
float aastep(float threshold, float x)
{
  float width = 0.7*length(vec2(dFdx(x), dFdy(x)));
  return smoothstep(threshold-width, threshold+width, x);
}
```

CRT Display
Checkerboard 1
Checkerboard 2
Hinomaru
Hinomaru (implementeu també una versió amb anti-aliasing, usant smoothstep; no patiu pel test)
Hinomaru 2
Pacman 2
Aigua
Opcionals:
Players
Halloween 2

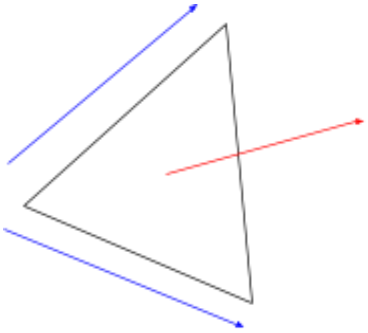
S5 - Més textures

La textura de normals serveix per tenir bona il·luminació en un pla amb textura de pedres, per exemple .

Bump Mapping → Només normals no es veu bé de costat

Displacement mapping → Modificar vèrtexs al fs

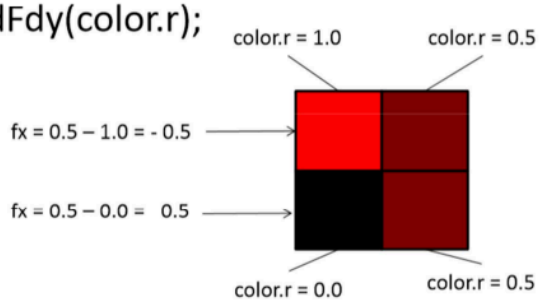
No podem usar la normal, però podem usar la derivada al FS



Per calcular la normal podríem fer el producte vectorial dels vectors que van d'un vèrtex a un altre.

dFdx, dFdy - exemple

```
float fx = dFdx(color.r);
float fy = dFdy(color.r);
```



fy = 1.0 – 0.0 = 1.0 fy = 0.5 – 0.5 = 0.0

S6 - Repàs

Setmana que ve comencem Geometry Shaders

Enunciats dels exercicis:

- Hunter
- Invaders
- Spring

Opcional:

- xrays (no feu cas del test)

S7 - Geometry Shaders

VS -> GS -> FS