

Laboratori de G

Professors de G

2021

Laboratori

- Primera meitat: shaders (GLSL)
- Segona meitat: shaders (GLSL) + aplicacions (C++, Qt, OpenGL)
- Usarem un visualitzador propi: viewer (aka GLArena)
- Metodologia bàsica:
 - Professor: explicacions (curtes excepte primera sessió de cada meitat)
 - Estudiants: implementació exercicis proposats (“obligatoris”/“opcionals”)
- Material: <https://www.cs.upc.edu/~virtual/G/index.php?dir=2.%20Laboratori/>
- Avaluació: preguntes als exàmens + dos controls de lab.
- Plagiarisme: a tots els actes evaluatius de G, el que lliureu (respostes, codi font...) ha de ser d'**autoria pròpia**. Altrament (còpia d'un altre estudiant, de repositoris externs, d'acadèmies, autoria compartida...) es considera frau.

Vertex shaders i Fragment shaders

Entorn per a desenvolupar shaders (viewer)

Professors de Gràfics

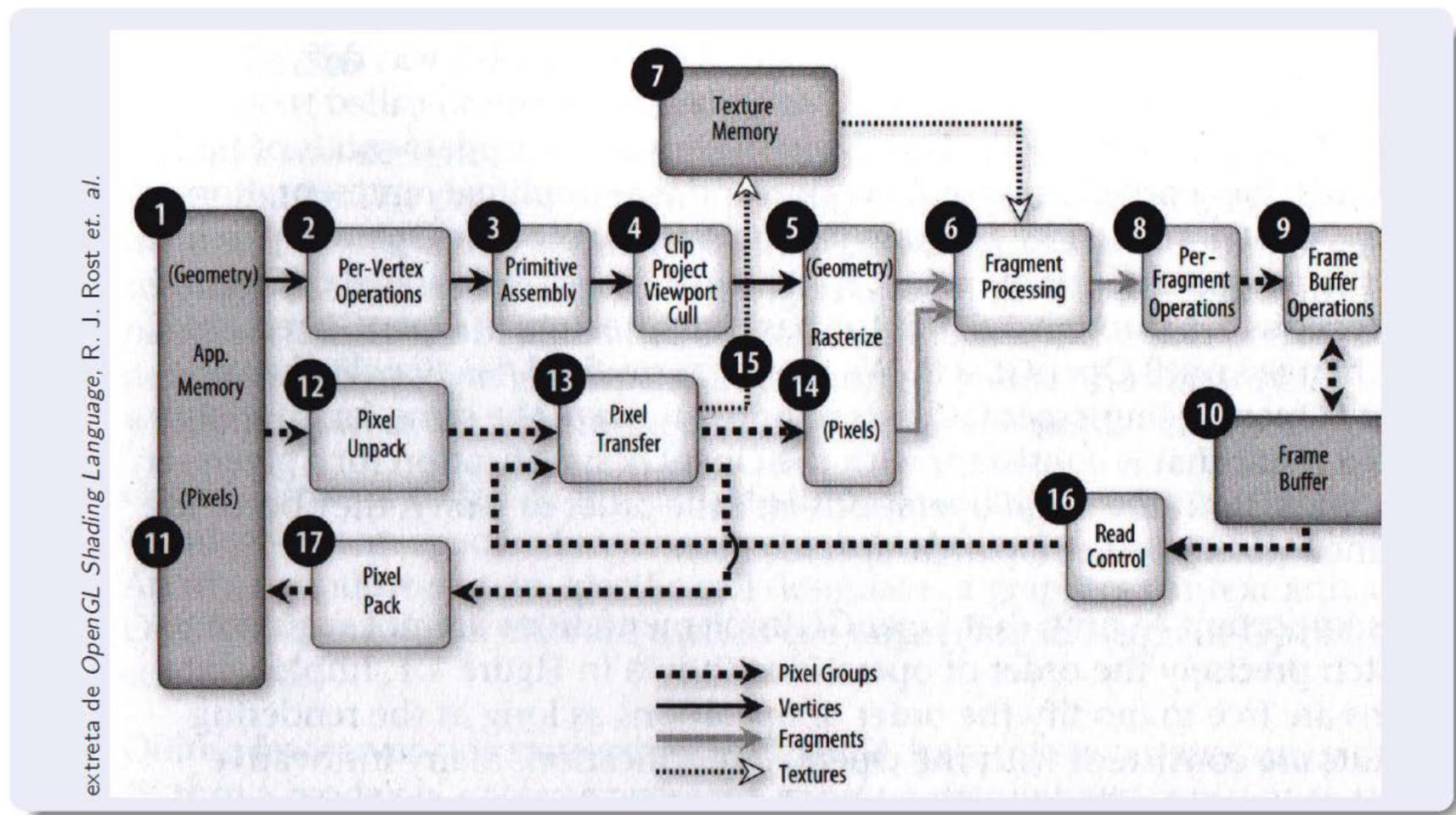
2021

Versions i pipelines d'OpenGL

OpenGL

1.0	(1992)	Primera versió
	...	
2.0	(2004)	VS + FS
	...	
3.0	(2008)	Core profile / Compatibility profile
3.1	(2009)	Millors GLSL
3.2	(2009)	Geometry shaders (GS)
3.3	(2010)	Usada al lab de gràfics
4.0	(2010)	Tessellation shaders (TCS + TES)
	...	
4.3	(2012)	Compute shaders (CS)
	...	
4.6	(2017)	Darrera versió publicada

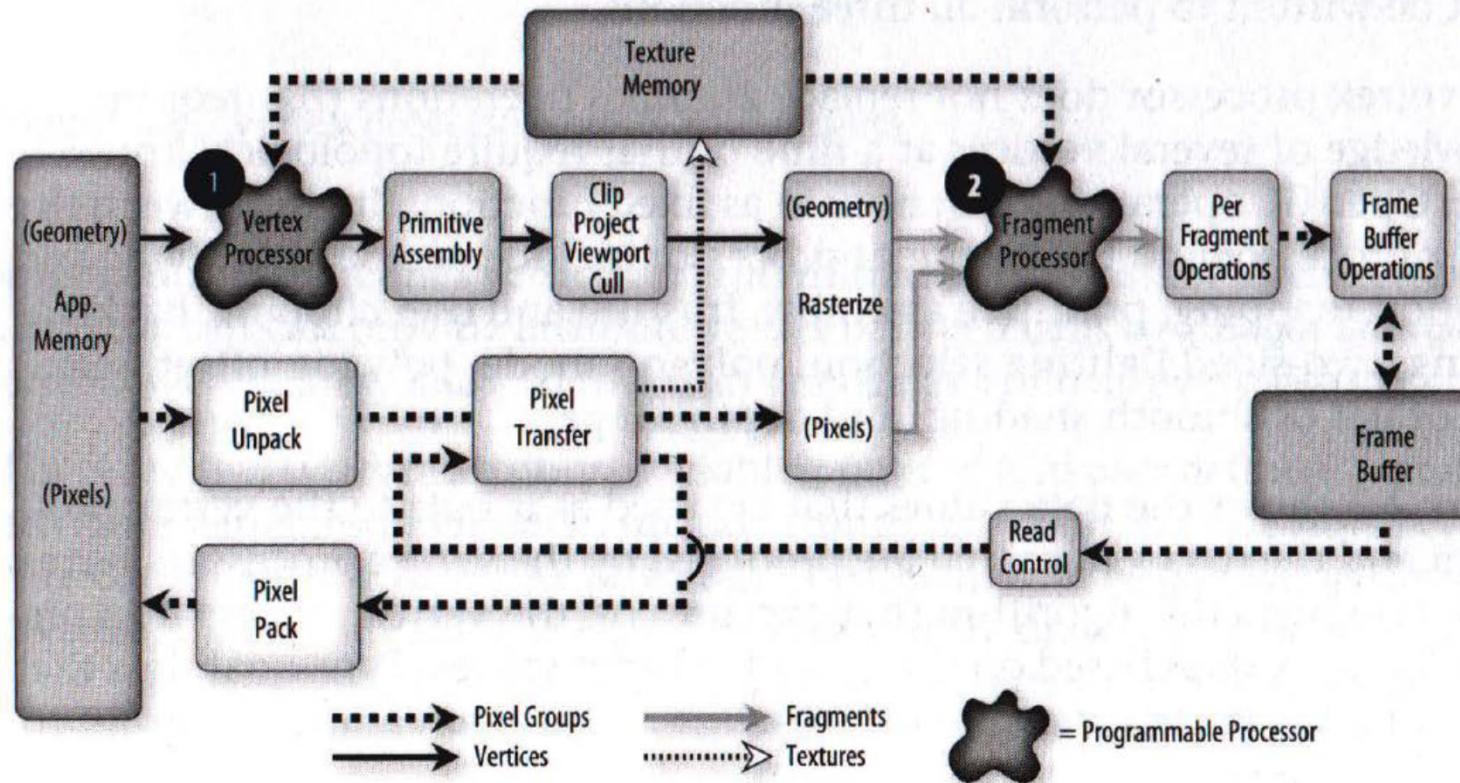
Pipeline fix



Pipeline programmable (VS + FS)

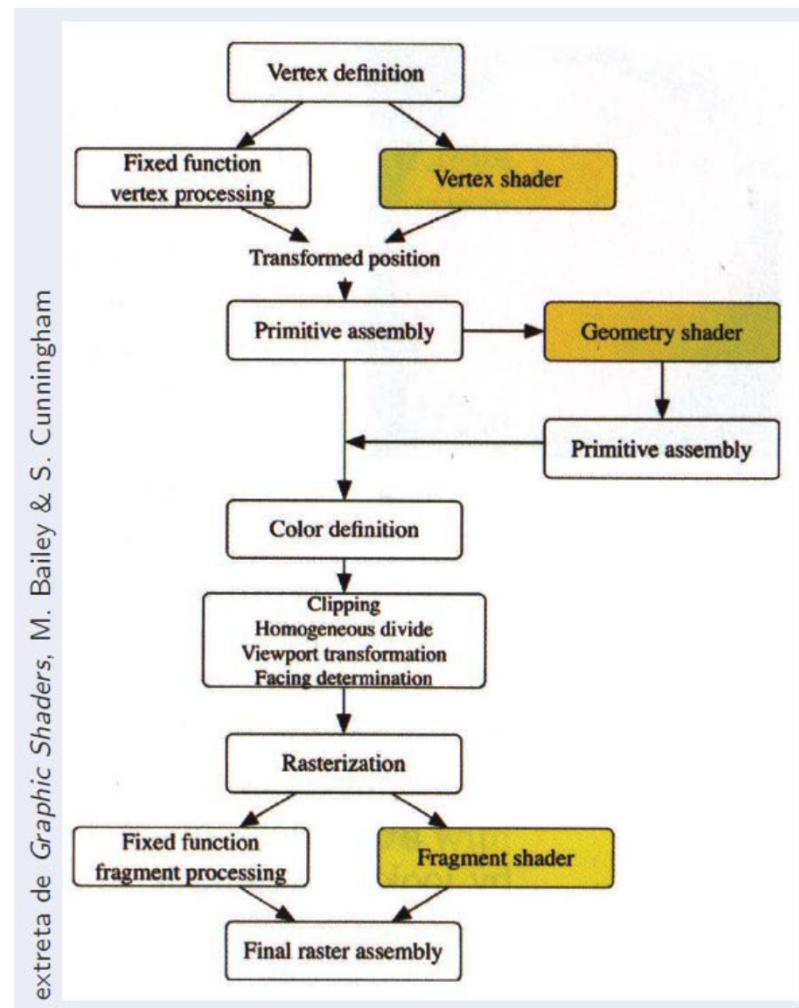
2.0+

extreta de OpenGL Shading Language, R. J. Rost et. al.

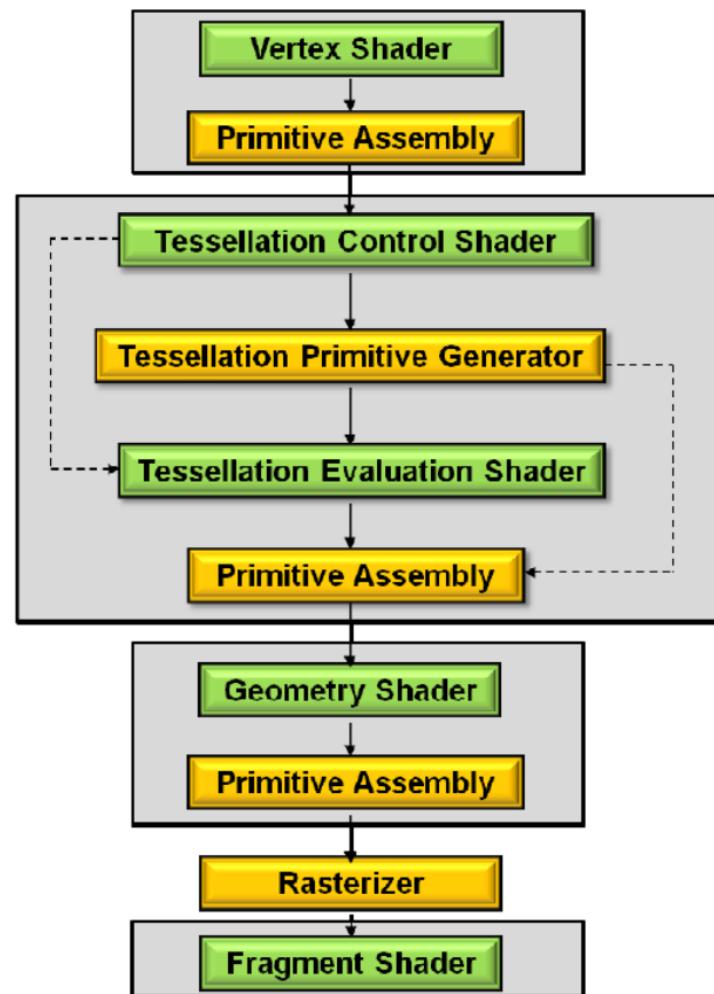


Pipeline programmable (VS + GS + FS)

3.2+

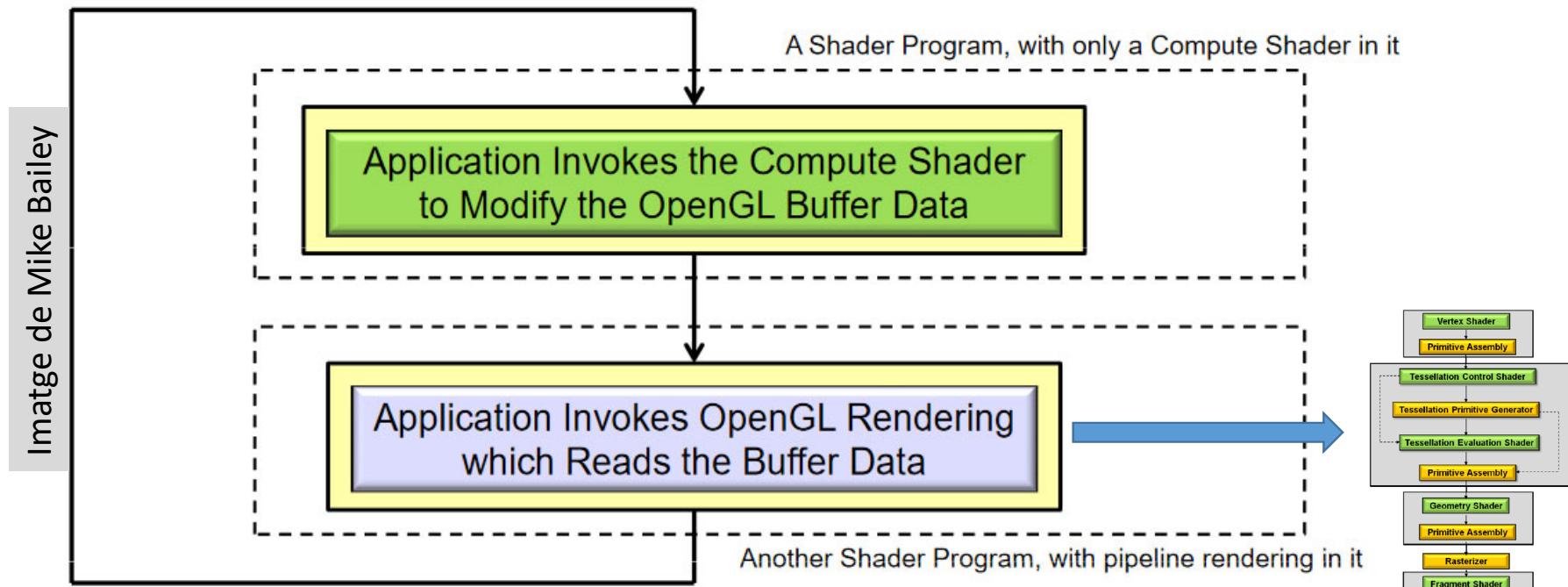


Pipeline programmable (VS + TCS + TES + GS + FS)



4.0+

Pipeline programmable (4.3+)

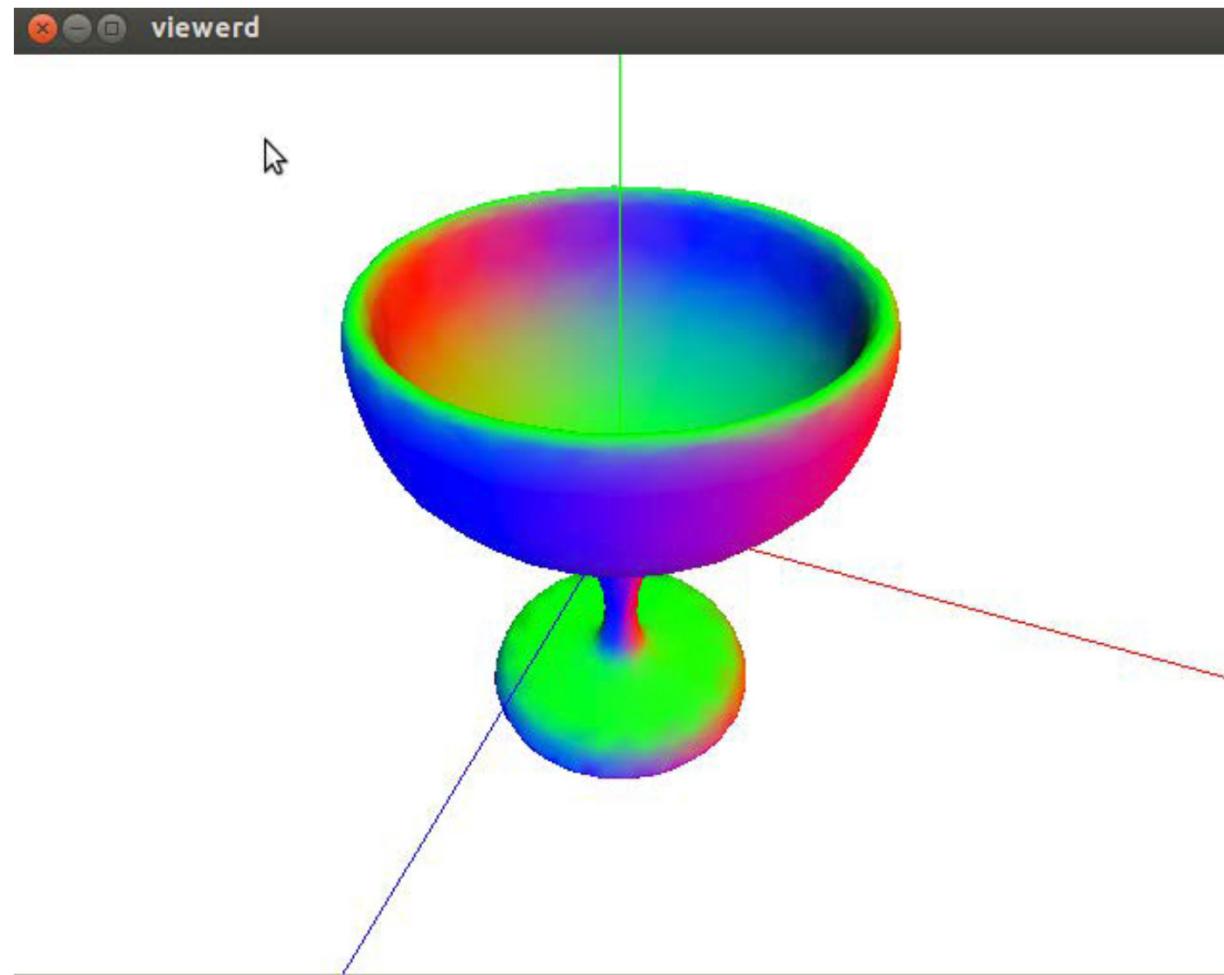


OpenGL

1.0	(1992)	Primera versió
	...	
2.0	(2004)	VS + FS
	...	
3.0	(2008)	Core profile / Compatibility profile
3.1	(2009)	Millors GLSL
3.2	(2009)	Geometry shaders (GS)
3.3	(2010)	Usada al lab de gràfics (VS+FS, VS+GS+FS)
4.0	(2010)	Tessellation shaders (TCS + TES)
	...	
4.3	(2012)	Compute shaders (CS)
	...	
4.6	(2017)	Darrera versió publicada

Viewer

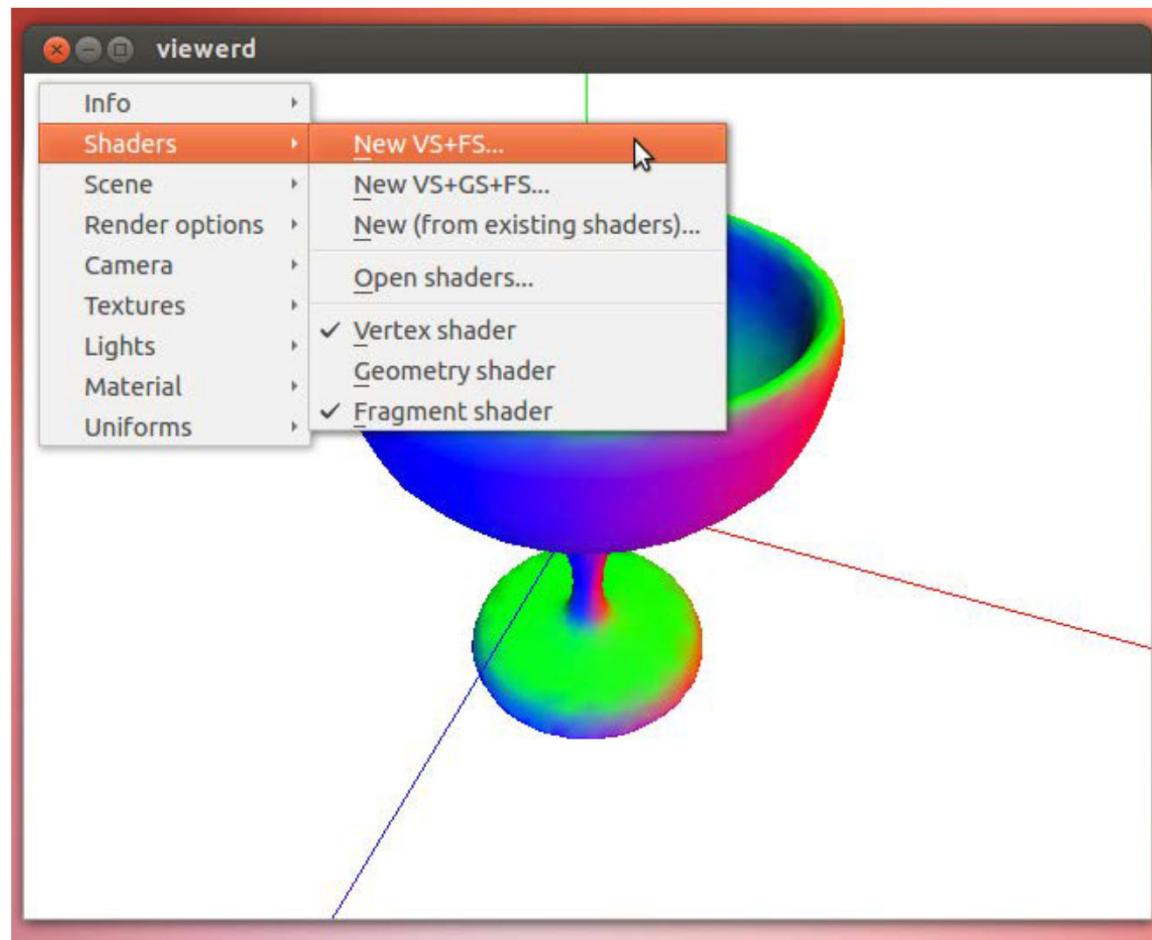
Viewer



Viewer als labs de la FIB

- Funciona en linux
- És recomanable crear una carpeta amb els shaders que anireu creant:
`mkdir shaders` (on vulgueu)
`cd shaders/`
`~/assig/grau-g/Viewer/GLarenaSL`
Premeu [SPACE] per accedir al menú

Viewer



Viewer

The image shows a dual-pane application interface for developing OpenGL shaders and viewing 3D scenes.

Left Panel (Code Editor):

- Title:** myShader.vert (~/NewViewer) - gedit
- Toolbar:** Includes standard file operations (Abrir, Guardar, Deshacer, Redo, Print, Find, Replace).
- Documents:** Two files are open: "myShader.vert" and "myShader.frag".
- Code Content (myShader.vert):**

```
1 #version 330 core
2
3 layout (location = 0) in vec3 vertex;
4 layout (location = 1) in vec3 normal;
5 layout (location = 2) in vec3 color;
6 layout (location = 3) in vec2 texCoord;
7
8 out vec4 frontColor;
9 out vec2 vtexCoord;
10
11 uniform mat4 modelViewProjectionMatrix;
12 uniform mat3 normalMatrix;
13
14 void main()
15 {
16     vec3 N = normalize(normalMatrix * normal);
17     frontColor = vec4(color, 1.0) * N.z;
18     vtexCoord = texCoord;
19     gl_Position = modelViewProjectionMatrix * vec4(vertex.xyz, 1.0);
20 }
```

- Status Bar:** GLSL 3.30, Ancho de la tabulación: 4, Ln 1, Col 1, INS.

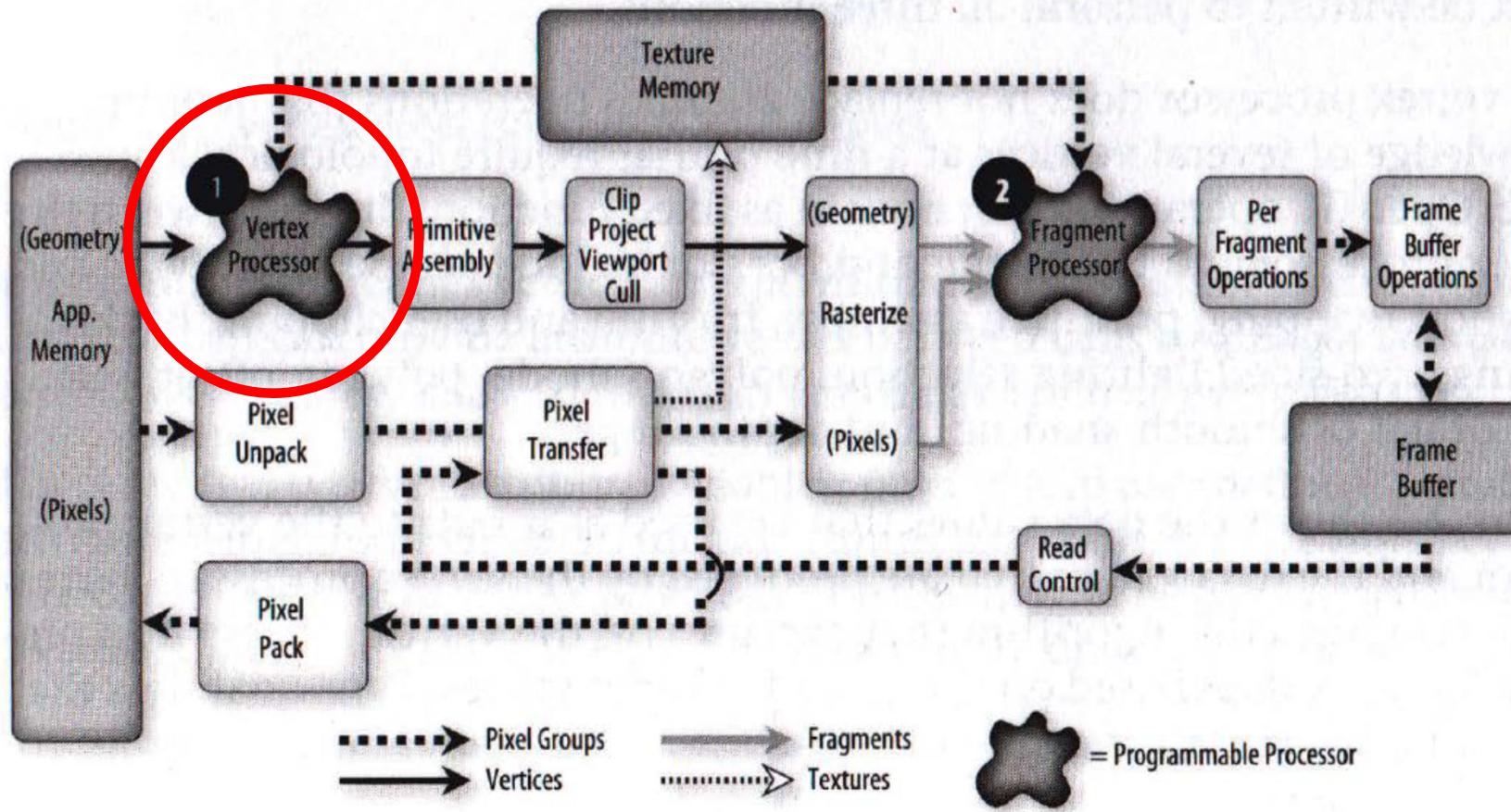
Right Panel (3D Viewer):

- Title:** viewerd
- Toolbar:** Standard window controls (close, minimize, maximize).
- Menu:** Info, Shaders, Scene, Render options, Camera, Textures, Lights, Material, Uniforms.
- Info Submenu:** A tooltip displays system information:
 - Current path is /home/andujar/NewViewer
 - VS: myShader.vert [COMPILE: OK]
 - GS: myShader.geom [DISABLED]
 - FS: myShader.frag [COMPILE: OK]
 - PROGRAM LINK: OK
- Scene:** A 3D rendering of a teapot with a gradient color map (blue at the base, red at the top).

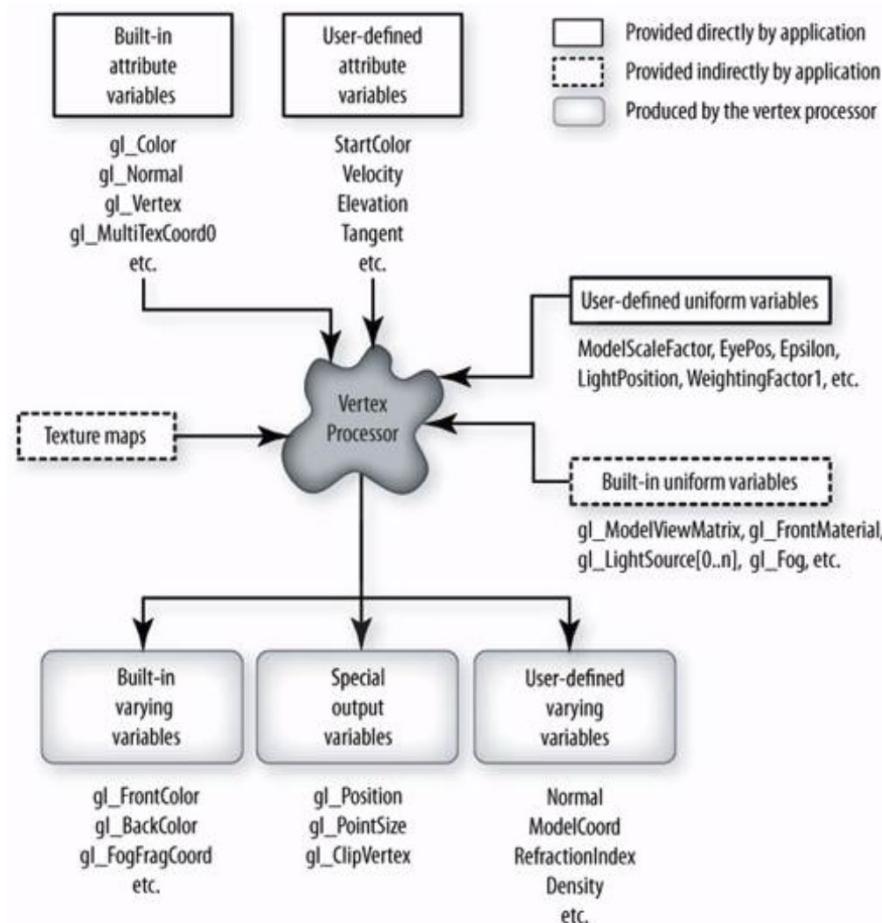
Configuració de gedit als labs de la FIB

- Activar syntax highlighting per GLSL 3.30:
`~/assig/grau-g/gedit-config`
- Activar el plugin “snippets” del gedit (**Preferences-Plugins-Snippets**)
Fa que **defs[TAB]** s’expandeixi a les declaracions de tots els uniforms que envia el viewer

Vertex shaders



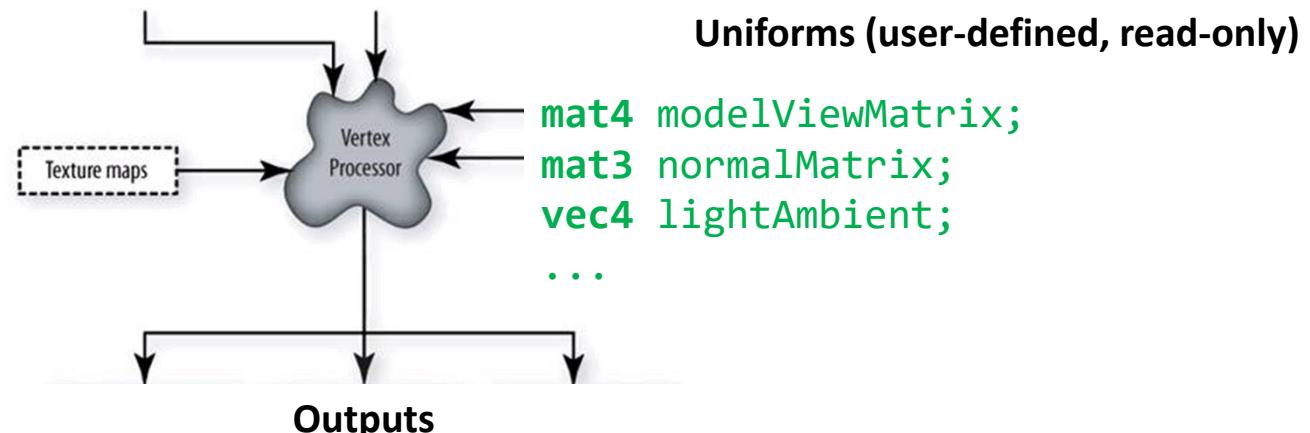
VS (compatibility profile)



VS (3.3 core profile)

Attributes (user-defined)

```
vec3 vertex;    // object space  
vec3 normal;   // object space  
vec3 color;  
vec2 texCoord;  
...
```



```
vec4 gl_Position; // predefinit; usualment en clip space  
vec4 frontColor;
```

VS: entrades

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)
```

VS: uniforms

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;
```

VS: uniforms

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

uniform vec4 lightAmbient;

uniform vec4 lightDiffuse;

uniform vec4 lightSpecular;

uniform vec4 lightPosition; // (sempre estarà en eye space)

uniform vec4 matAmbient;

uniform vec4 matDiffuse;

uniform vec4 matSpecular;

uniform float matShininess;

VS: uniforms

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

uniform vec3 boundingBoxMin; // cantonada de la capsa englobant

uniform vec3 boundingBoxMax; // cantonada de la capsa englobant

uniform vec2 mousePosition; // pos del cursor (window space); origen BL

uniform float time; // temps (segons) des de la darrera compilació

VS: sortides

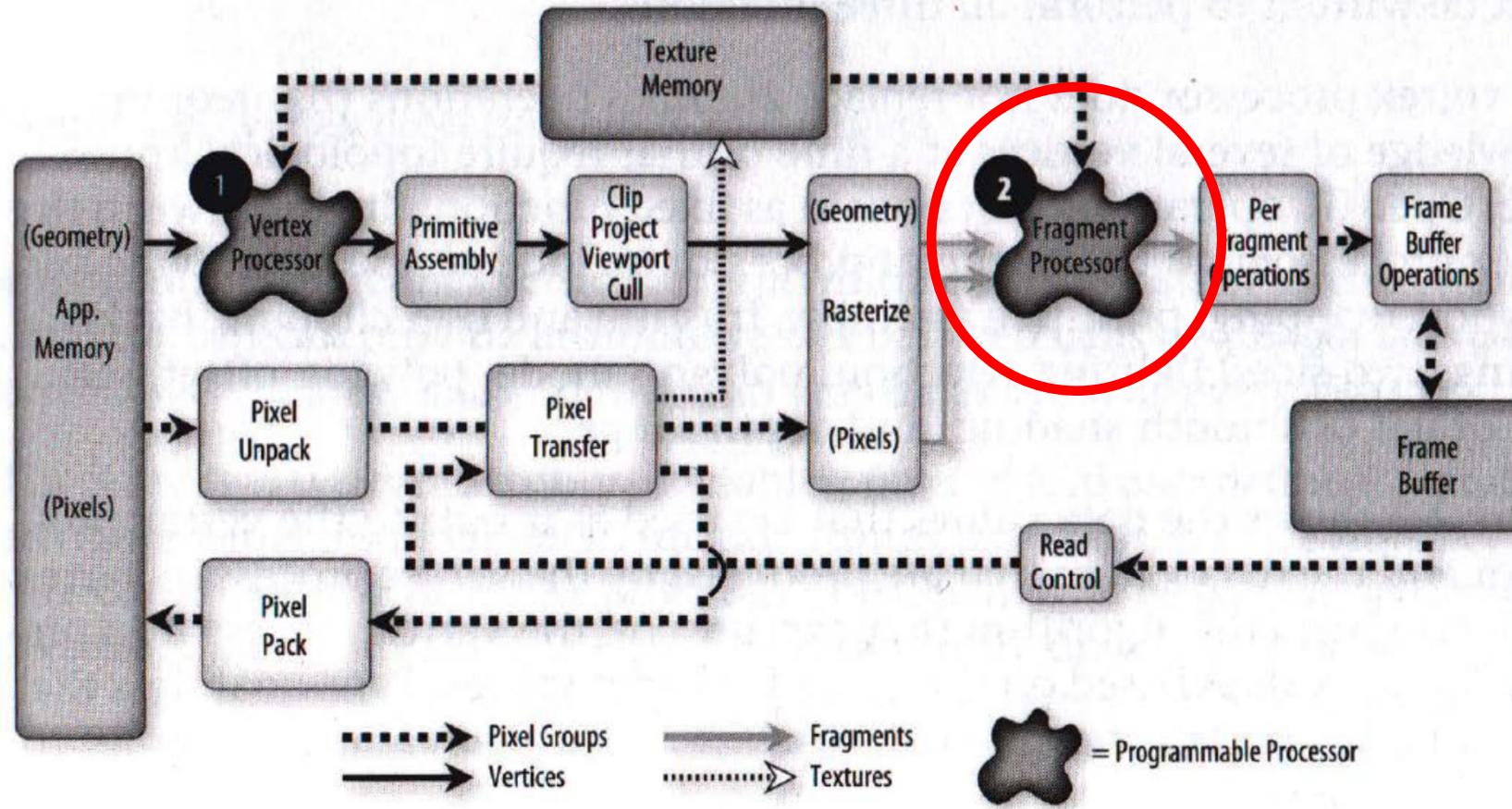
Output variables (pel VS són de sortida; pel FS són d'entrada):

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

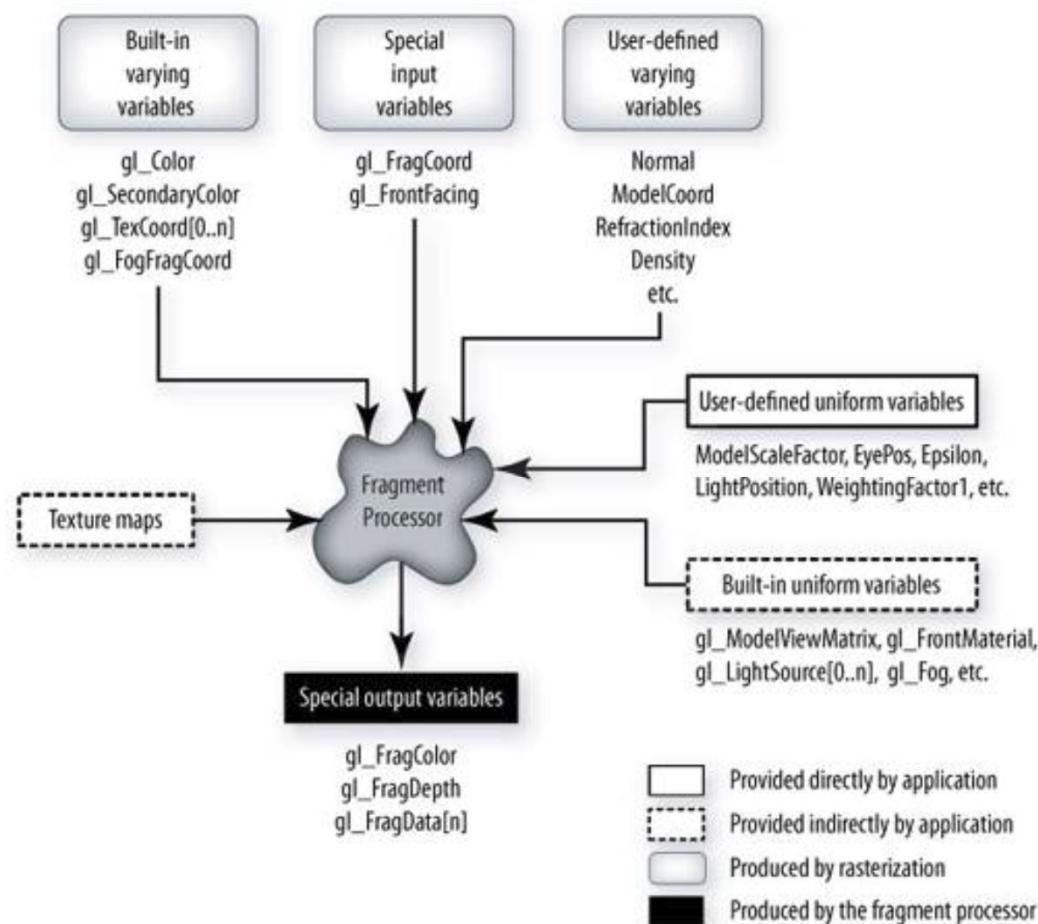
VS

- El VS s'executa per cada vèrtex que s'envia a OpenGL.
- Les tasques habituals d'un VS són:
 - Transformar el vèrtex (object space → clip space)
 - Transformar i normalitzar la normal (eye space)
 - Calcular la il·luminació del vèrtex
 - Generar o passar les coords de textura pel vèrtex

Fragment shaders



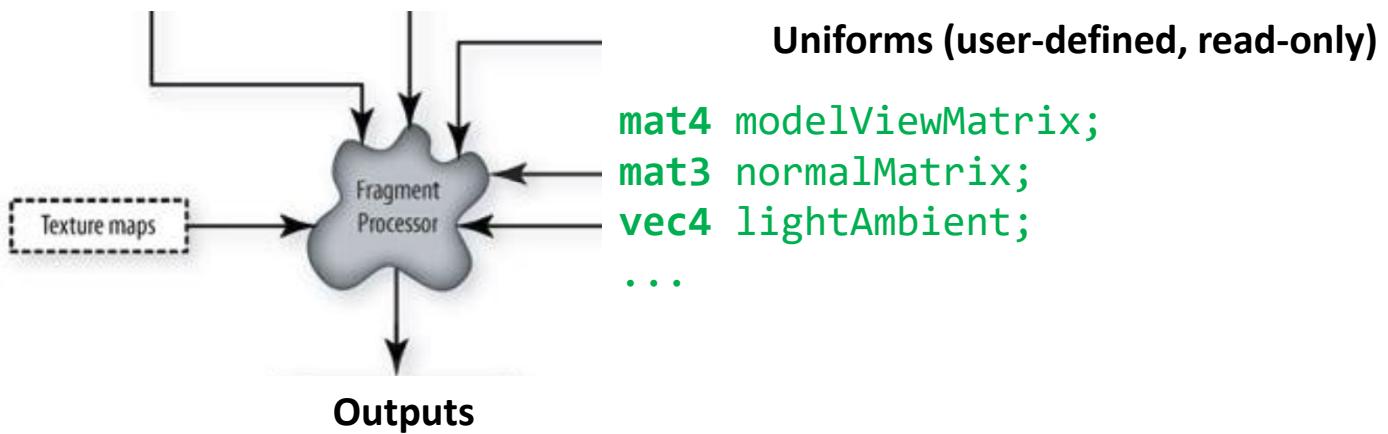
FS (compatibility profile)



FS (3.3 core profile)

Inputs

```
vec4 gl_FragCoord; // window space  
bool gl_FrontFacing; // el fragment l'ha generat una primitiva frontface?  
vec4 frontColor;  
...  
...
```



Outputs

```
float gl_FragDepth; // z en window space  
vec4 fragColor;
```

FS

- El FS s'executa per cada fragment que produeix una primitiva.
- Les tasques habituals d'un FS són:
 - Calcular la il·luminació
 - Usar textures per a afegir detall.
- I el que no pot fer un fragment shader:
 - Canviar les coordenades del fragment (sí pot canviar `gl_FragDepth`)
 - Accedir a informació d'altres fragments (tret de `dFdx`, `dFdy`)

Llenguatge GLSL

Elements del llenguatge

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);

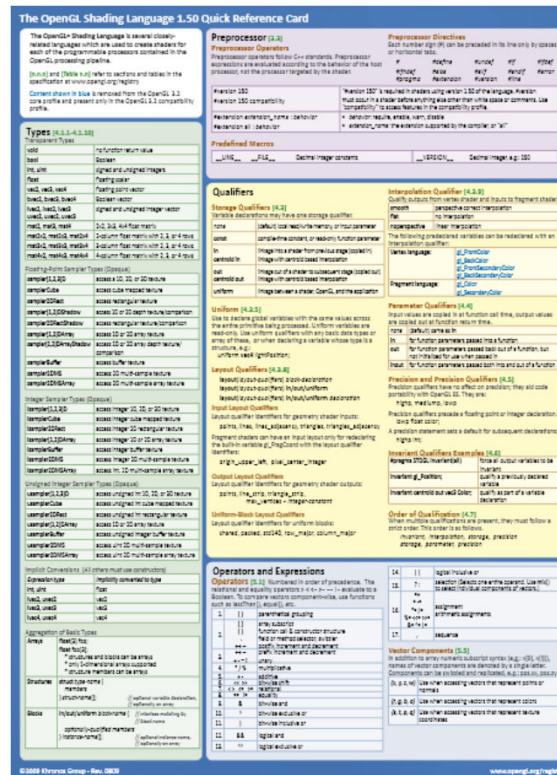
Elements del llenguatge

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(), ...

OpenGL Quick Reference card

<https://www.khronos.org/files/opengl-quick-reference-card.pdf>



Exemple: Phong Shading

VS (1/3)

```
#version 330 core
layout(location= 0) in vec3 vertex;
layout(location= 1) in vec3 normal;
layout(location= 2) in vec3 color;
layout(location= 3) in vec2 texCoord;
out vec4 frontColor;

uniform mat4 modelViewProjectionMatrix, modelViewMatrix;
uniform mat3 normalMatrix;
uniform vec4 matAmbient, matDiffuse, matSpecular;
uniform float matShininess;
uniform vec4 lightAmbient, lightDiffuse, lightSpecular
uniform vec4 lightPosition;
```

VS (2/3)

```
vec4 light(vec3 N, vec3 V, vec3 L)
{
    vec3 R = normalize( 2.0*dot(N, L)*N - L );
    float NdotL= max( 0.0, dot(N, L) );
    float RdotV= max( 0.0, dot(R, V) );
    float Idiff= NdotL;
    float Ispec= 0;
    if (NdotL>0) Ispec=pow(RdotV, matShininess);
    return matAmbient * lightAmbient +
           matDiffuse * lightDiffuse * Idiff +
           matSpecular * lightSpecular * Ispec;
}
```

VS (3/3)

```
void main()
{
    vec3 P = (modelViewMatrix * vec4(vertex, 1.0)).xyz;
    vec3 N = normalize(normalMatrix* normal);
    vec3 V = normalize(-P);
    vec3 L = normalize(lightPosition.xyz - P);
    frontColor= light(N, V, L);
    gl_Position= modelViewProjectionMatrix * vec4(vertex, 1.);
}
```

FS

```
#version 330 core
in vec4 frontColor;
out vec4 fragColor;

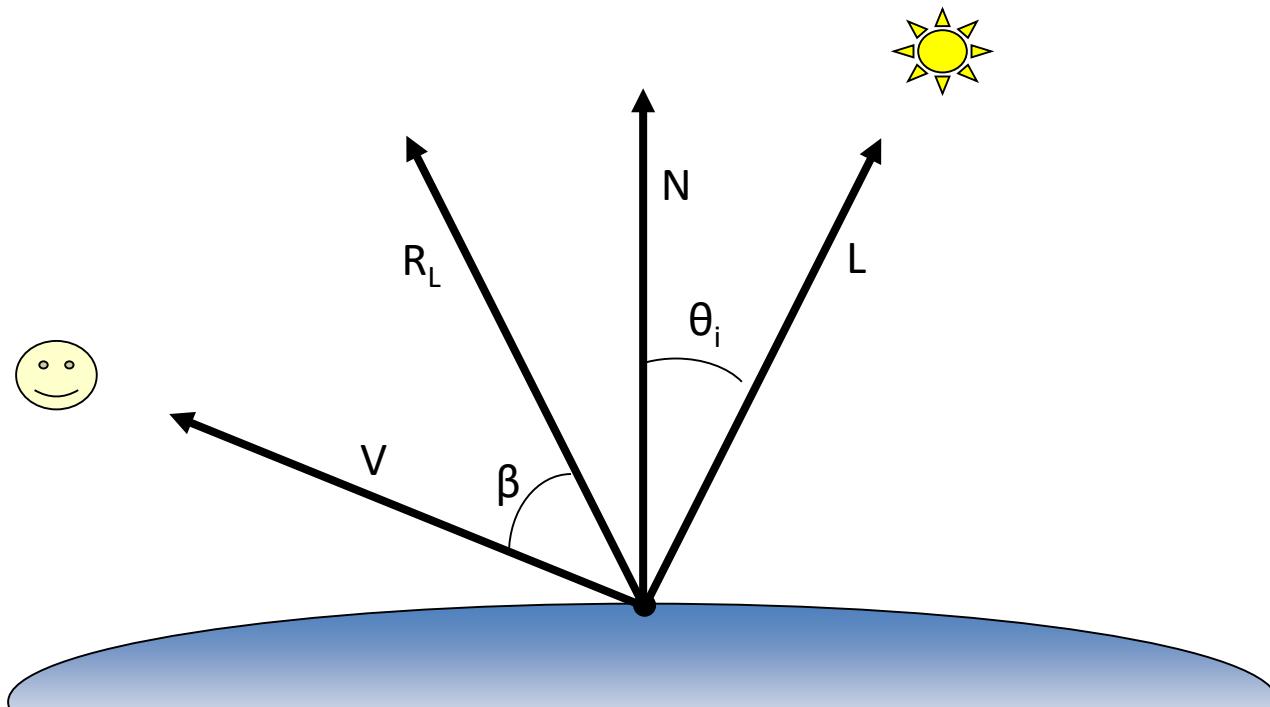
void main()
{
    fragColor= frontColor;
}
```

Guia exercicis

Il·luminació

2020

Notació



Model de Phong

$$K_e + K_a(M_a + I_a) + \underbrace{K_d I_d (N \cdot L)}_{\text{Només si } N \cdot L > 0} + \underbrace{K_s I_s (R \cdot V)^s}_{\text{Només si } N \cdot L > 0}$$

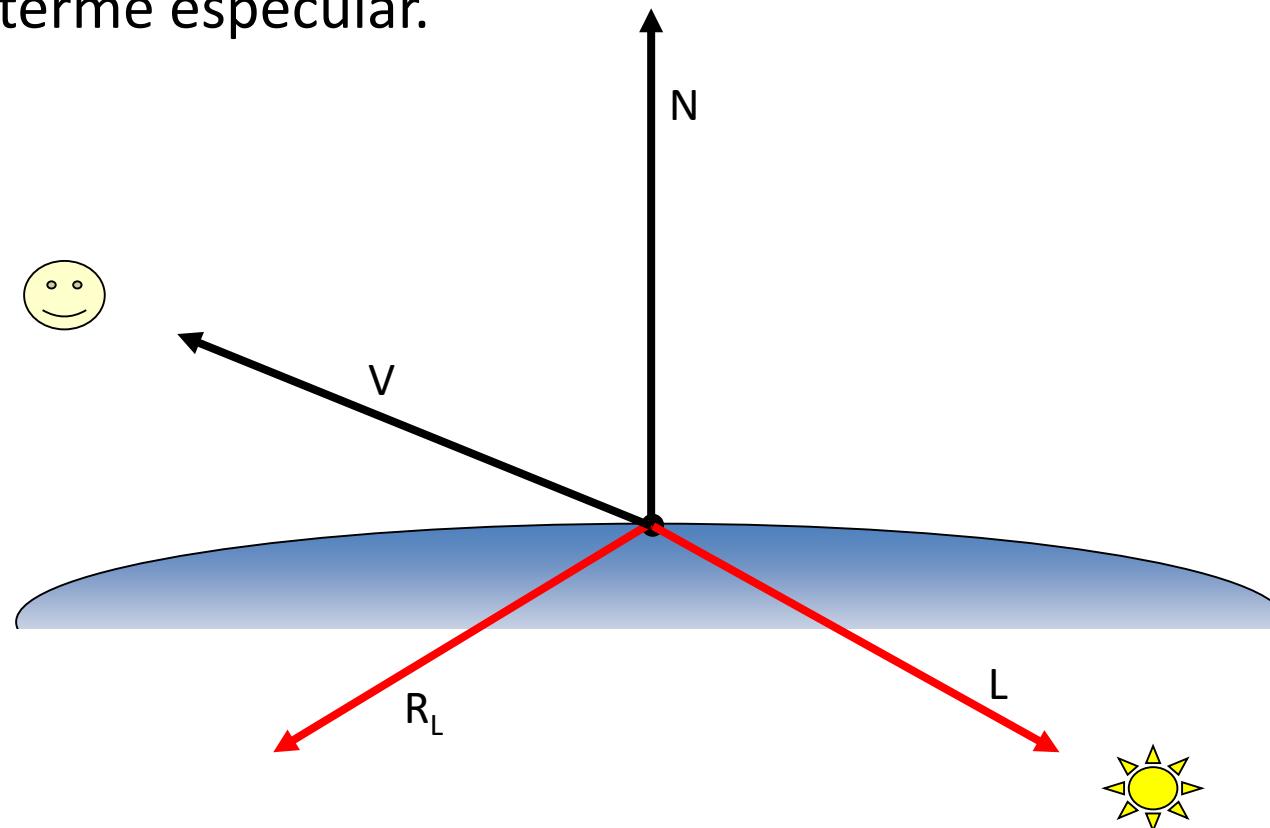
- K_* = material

- I_* = Illum

Notació

Si $N \cdot L < 0$:

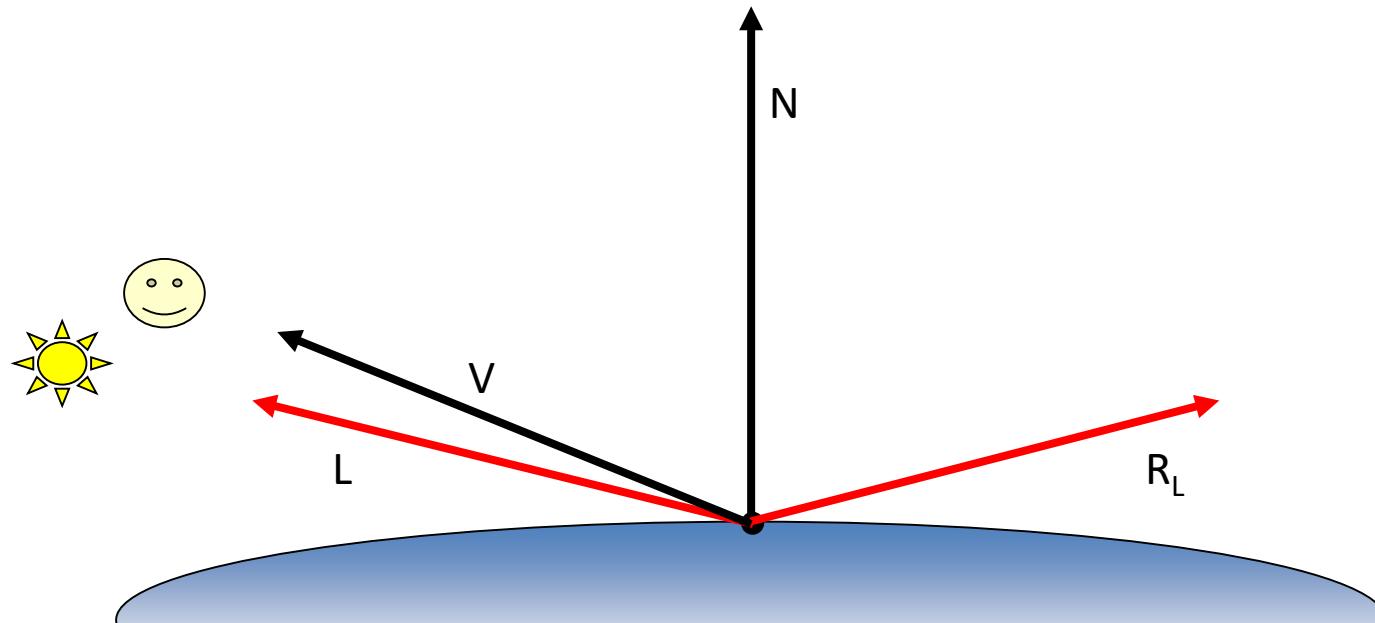
- Cal evitar que la contribució difosa “resti” llum. Useu per exemple $\max(0, \dots)$
- Cal ignorar el terme espectral.



Notació

Si $R \cdot V < 0$:

- Cal evitar que la contribució especular “resti” llum. Useu per exemple $\max(0, \dots)$



Exemple senzill

- K_* = material
- I_* = llum

$$K_e + K_a(M_a + I_a) + \underbrace{K_d I_d (N \cdot L)}_{\text{Només si } N \cdot L > 0} + K_s I_s (R \cdot V)^s$$

Quan normalitzar

- Els vectors (N , L , R , V) que apareixen a les eqüacions d'il·luminació han de ser unitaris (cal normalitzar abans)
- En general, la longitud d'un vector no es preserva:
 - Quan es multiplica per una matriu (`normalMatrix * normal`)
 - Quan s'interpola línialment (ex. `out vec3 N`)
- On normalitzar? immediatament abans de fer els càculs que assumeixen que el vector és unitari: al VS si il·lum per vèrtex, al FS si il·lum per fragment.

Laboratori Gràfics

Shaders Sessió 4

Shaders i textures (1)

VS

```
...
layout (location = 3) in vec2 texCoord;
out vec2 vtexCoord;

void main() {
    vtexCoord = texCoord; // o similar...
    ...
}
```

FS

```
...
uniform sampler2D myMap;
in vec2 vtexCoord;

void main() {
    gl_FragColor = texture(myMap, vtexCoord);
}
```

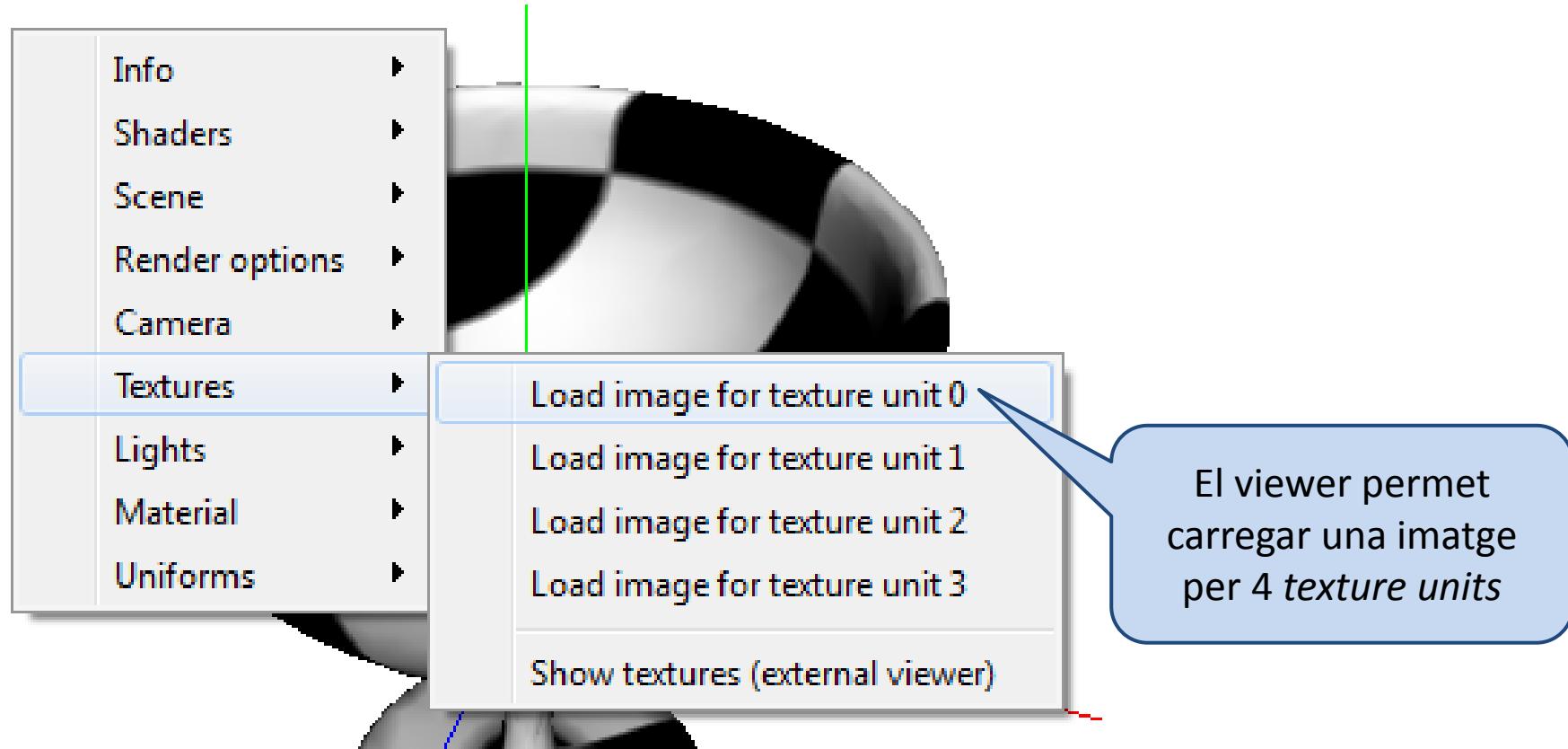
Un **sampler2D** és
una textura 2D

Retorna un **vec4**
amb el color RGBA

vec2 amb les
coordenades de
textura

Shaders i textures (2)

Pas 2: obrir els fitxers (.png...) amb les imatges



Shaders i textures (3)

El viewer associa cada sampler amb una texture unit basant-se en el darrer caràcter del nom:

```
uniform sampler2D colorMap; // no digit → unit 0 ←→  
uniform sampler2D normMap1; // ends with '1' → unit 1 ←→  
uniform sampler2D noise3;   // ends with '3' → unit 3 ←→
```

Load image for texture unit 0
Load image for texture unit 1
Load image for texture unit 2
Load image for texture unit 3

Laboratori de Gràfics

Sessió 5

Interpolació per fragment

- Tot el que s'interpolà per cada fragment (coords x,y,z, coords de textura s,t, out's definits per l'usuari) es calcula al **centre del pixel** corresponent. Per tant:

`fract(glFragCoord.x)` serà 0.5

`fract(glFragCoord.y)` serà 0.5

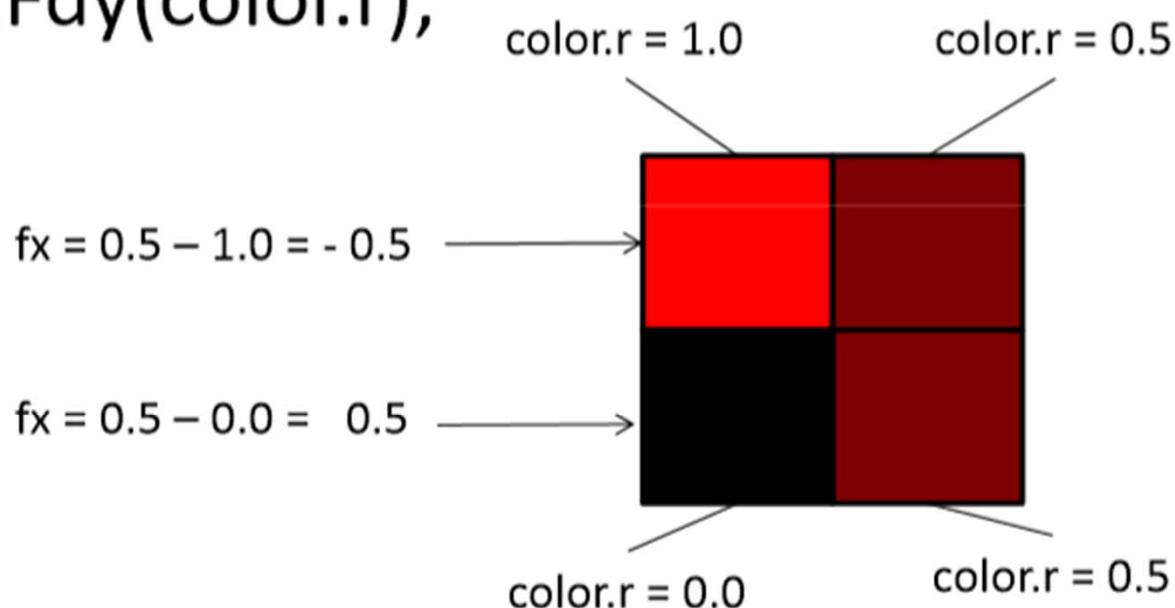
- En algunes versions de GLSL, és pot eliminar aquest offset redeclarant `gl_FragCoord`

```
layout(pixel_center_integer) in vec4 gl_FragCoord;
```

$dFdx$, $dFdy$ - exemple

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

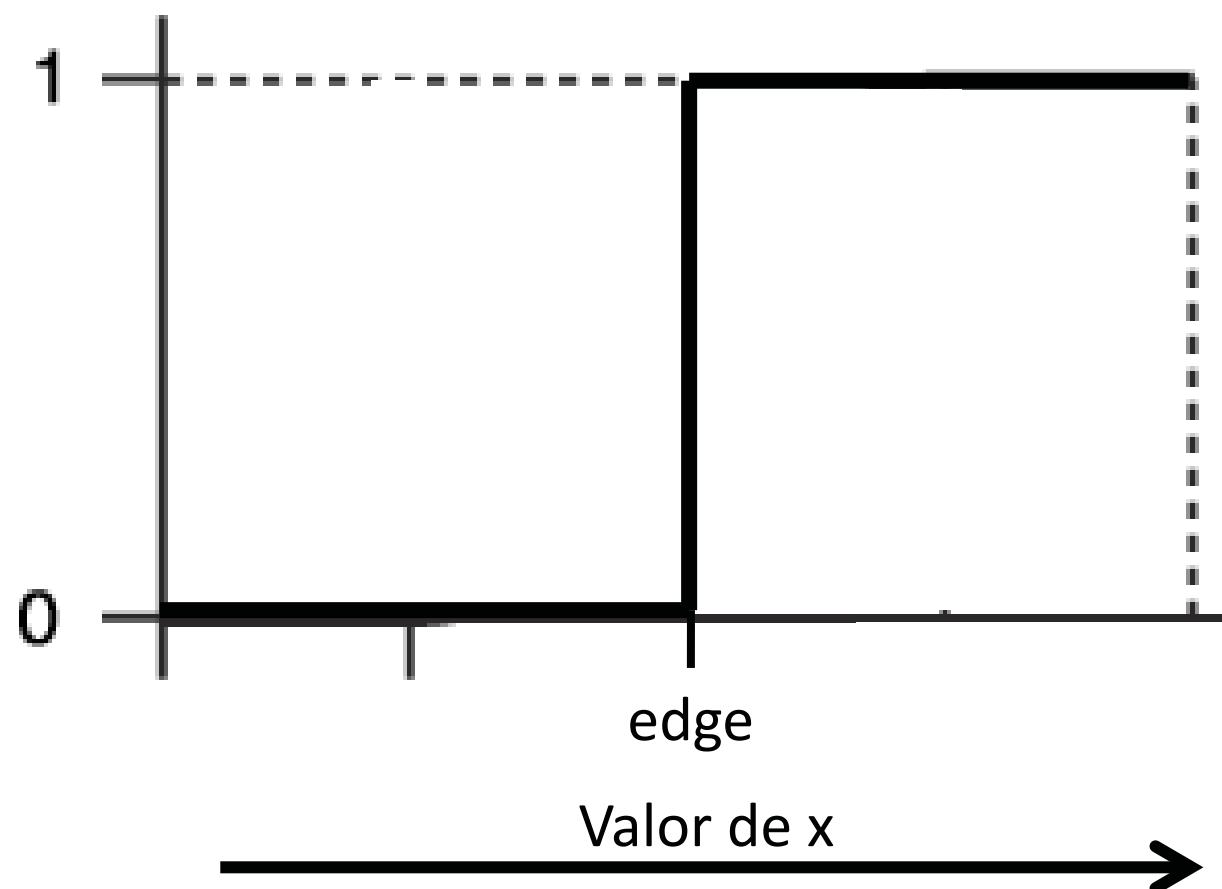
```
float fy = dFdy(color.r);
```



Funcions step, smoothstep

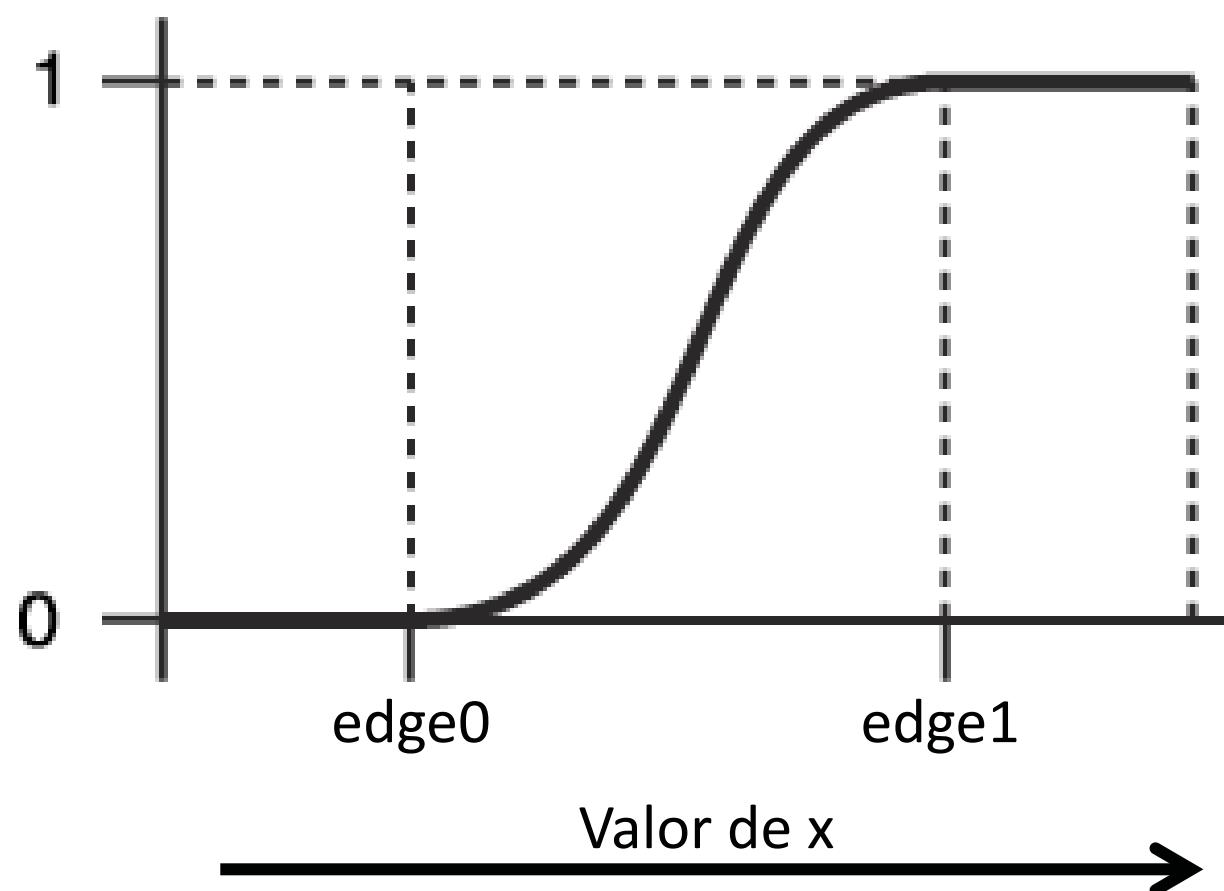
float step(float edge, float x)

$$\begin{cases} 0 & \text{if } x < \text{edge} \\ 1 & \text{otherwise} \end{cases}$$



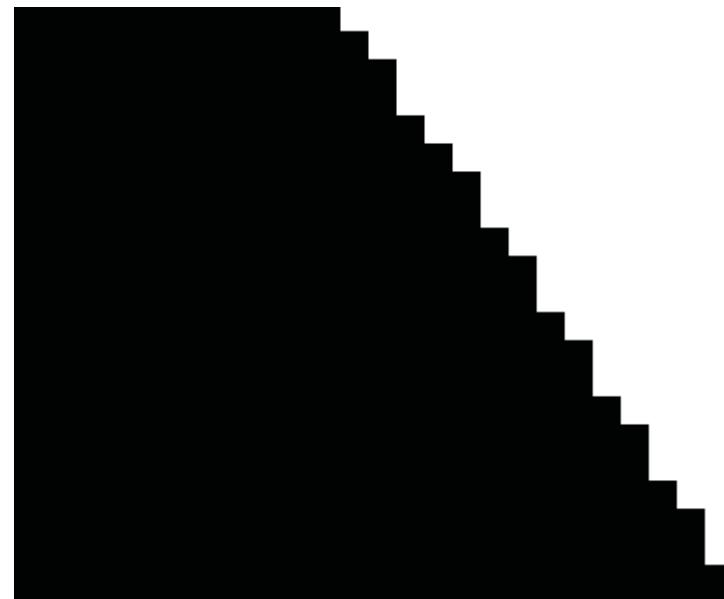
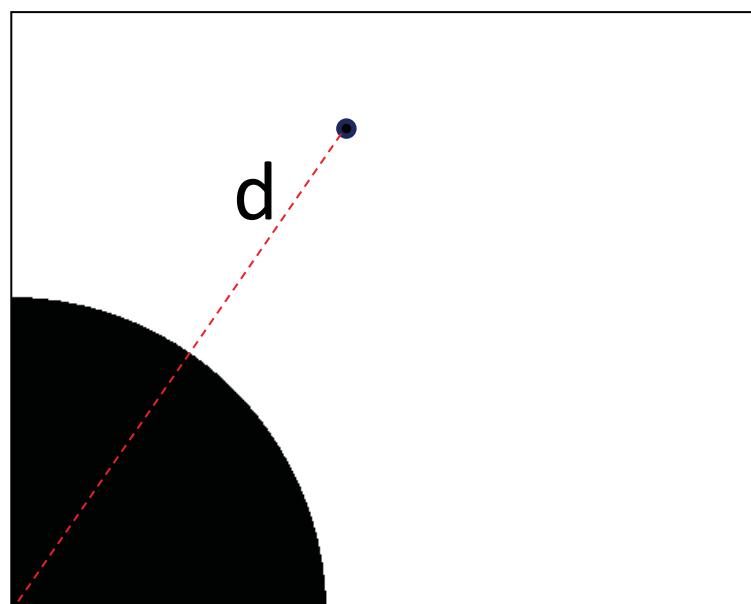
Funcions step, smoothstep

float smoothstep(float edge0, float edge1, float x)



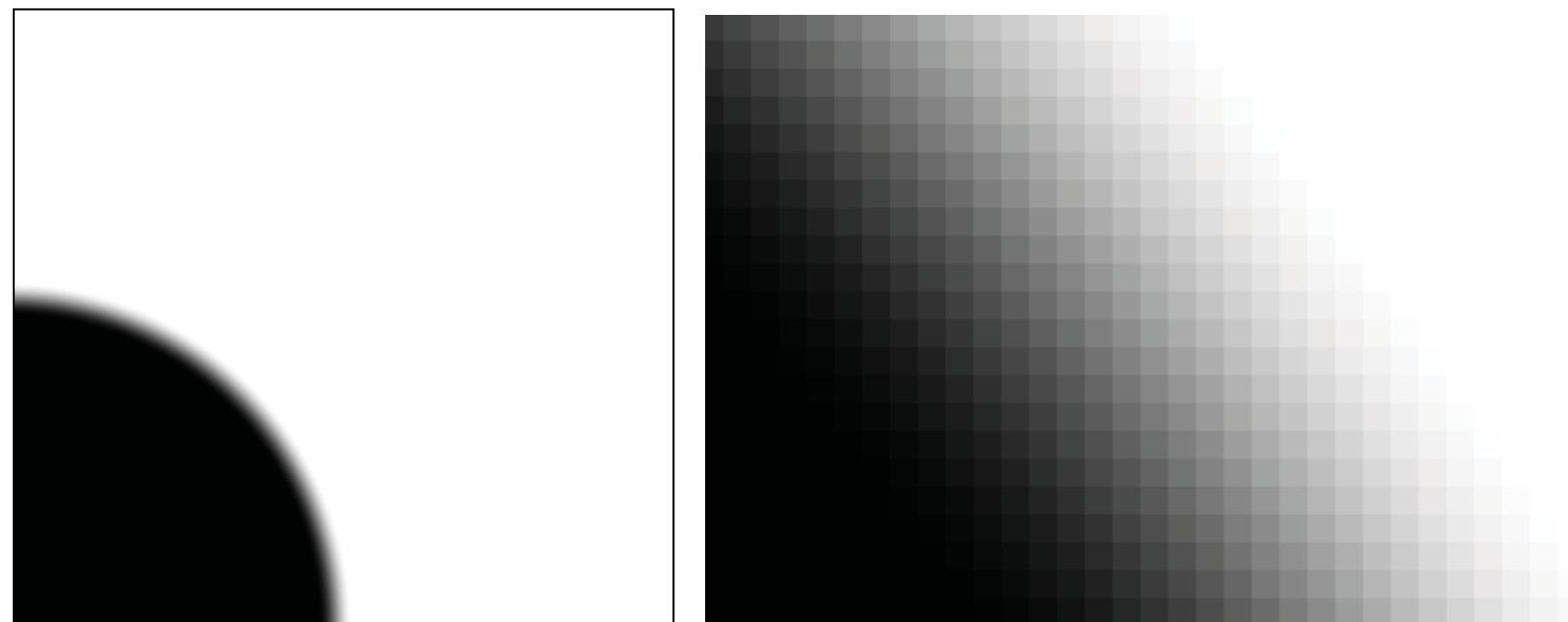
Exemple - step

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



Exemple - step

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



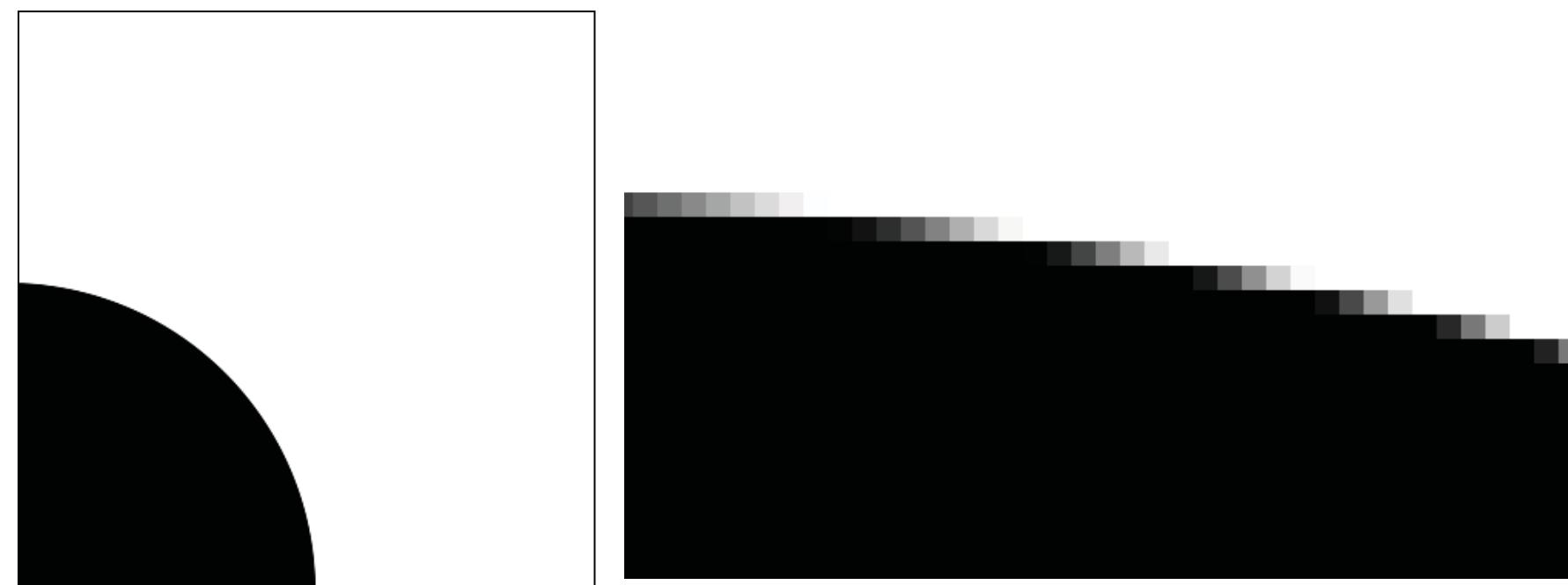
Exemple - smoothstep

```
void main() {  
    float d = length(gl_FragCoord.xy);  
    gl_FragColor = vec4(smoothstep(200-1,200+1, d));  
}
```



Exemple - smoothstep

```
void main() {  
    float d = length(gl_FragCoord.xy);  
    gl_FragColor = vec4(smoothstep(200-0.5,200+0.5, d));  
}
```



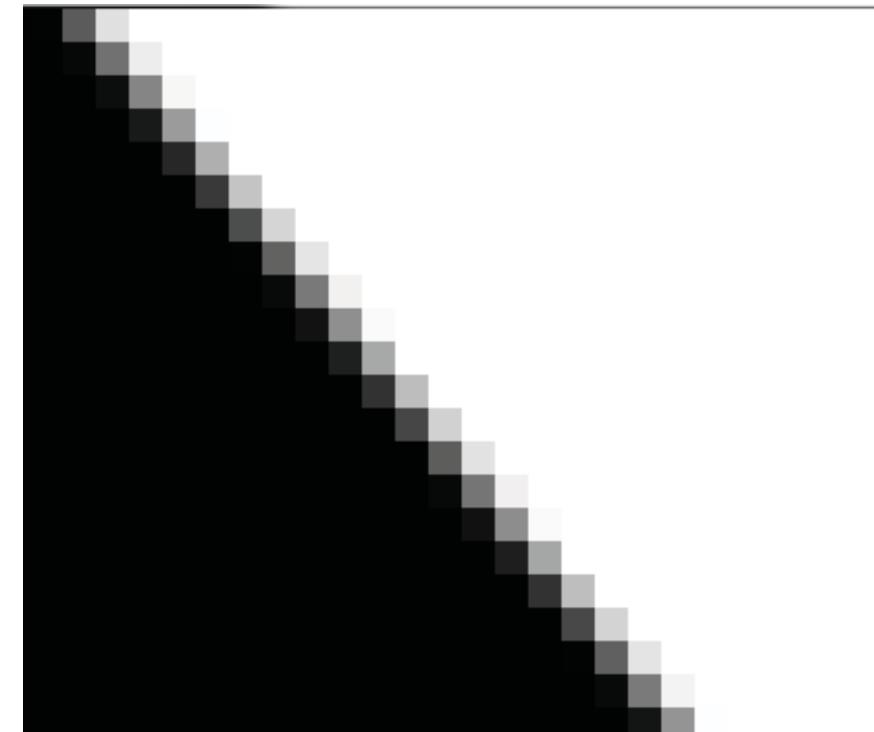
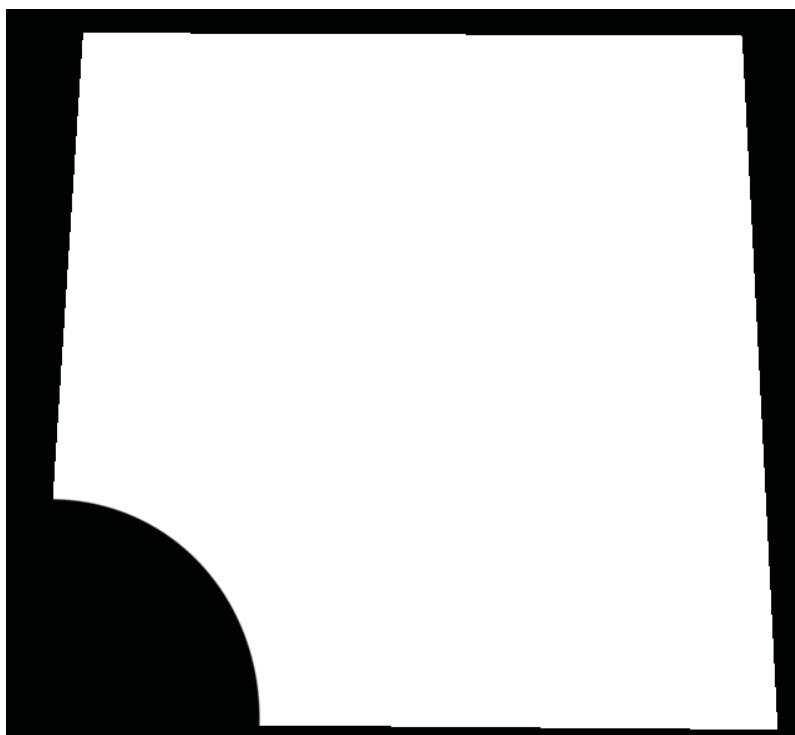
Exemple 2 - smoothstep

```
void main() {  
    float d = length(vtexCoord);  
    const float r = 0.3;  
    gl_FragColor = vec4(smoothstep(r-0.5, r+0.5, d));  
}
```



Exemple 2 – smoothstep + dFdx,dFdy

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



aastep (*)

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

(*) Patrick Cozzi, Christophe Riccio (Eds.) *OpenGL Insights*, CRC Press, 2012

Textures amb OpenGL

© Professors de VA

Grup MOVING – Dep. LSI – UPC

Ús de textures

- Tres etapes:
 - Creació de la textura:
 - Creació: glGenTexture, glBindTexture, glTexImage
 - Definició paràmetres: glTexParameter
 - Dibuix de les primitives texturades
 - Activació: glEnable i glBindTexture
 - Definició funció texturació: glTexEnvi
 - Generació coordenades: glTexCoord o automàtiques
 - Destrucció textures: glDeleteTextures

Ús de textures

// 1. Activar el texture mapping desitjat

// Només pot estar activat un mode: GL_TEXTURE_1D, 2D o 3D
glEnable(GL_TEXTURE_2D);

// 2. Activar el texture object corresponent

glBindTexture(GL_TEXTURE_2D, id);

// 3. Establir la funció de texturació

glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

// 4. Dibuixar la primitiva

glBegin(GL_POLYGON);
glTexCoord2d(0, 0);
 glVertex3d(...);
...

// o utilitzant coordenades automàtiques

Creació de l'objecte textura

- Generar un nou nom:
 - `void glGenTextures(1, &texName);`
 - Crea una textura (1) i emmagatzema el seu identificador a *texName*
- Activar la textura
 - `void glBindTexture(GL_TEXTURE_2D, texName);`
 - Les següents operacions de textures actuaran sobre *texName*.

Creació de l'objecte textura

- Introducció de les dades:
 - `void glTexImage2D(GLenum objective, GLint level,
GLint internalFormat, GLsizei width, GLsizei height,
GLint border, GLenum format, GLenum type, GLvoid*
pixels);`
 - *objective*: `GL_TEXTURE_2D`
 - *level*: 0 (nivells de mip mapping)
 - *internalFormat* i *format*: `GL_RGB` o `GL_RGBA`
 - *width* i *height*: de la forma 2^m+2^b (mín 64x64)
 - *border*: 0 o 1
 - *type*: de les dades que passem a *pixels* (`GL_BYTE`,
`GLfloat...`)
 - *pixels*: Array de bytes amb valors del tipus *tipus*

Creació de l'objecte textura

- Altres formes de posar les dades. A partir de la informació generada:
 - `void glCopyTexImage2D(GLenum target, GLint level, GLenum internalFormat, GLint x, GLint y, GLsizei width, GLsizei height, GLint border);`
 - Defineix la textura a partir d'una regió rectangular del `GL_READ_BUFFER` actiu (com el `glCopyPixels` però els `pixels` van a memòria de textura en comptes del `framebuffer`).
 - `void glCopyTexSubImage2D(GLenum target, GLint level, GLint xoffset, GLint yoffset, GLint x, GLint y, GLsizei width, GLsizei height);`
 - Substitueix una regió rectangular d'una textura ja definida per una regió rectangular.

Funcions de textura

- Definir el comportament en filtrat:
 - void **glTexParameteri(GL_TEXTURE_2D,**
filtre, *filtrat*);
 - *filtre*: ampliació (GL_TEXTURE_MAG_FILTER) o
reducció (GL_TEXTURE_MIN_FILTER)
 - *filtrat*: agafar el més proper (GL_NEAREST) o una
interpolació (GL_LINEAR)
 - Si no es defineixen els filters pot no veure's
res!!!

Dibuixat escena

- Comportament més enllà de [0.0, 1.0]:
 - `void glTexParameteri(GL_TEXTURE_2D, param, tipus);`
 - *param*: s (`GL_TEXTURE_WRAP_S`) o t (`GL_TEXTURE_WRAP_T`)
 - *tipus*: repetir (`GL_REPEAT`) o tallar (`GL_CLAMP`)

Laboratori de Gràfics, part 2.

À. Vinacua, C. Andújar i professors de Gràfics

5 de novembre de 2019

Segona part del laboratori



Segona part del laboratori

Objectius

Extendrem el viewer que hem fet servir per a programar shaders, aprenent programació més avançada en OpenGL

Implementarem en OpenGL efectes per augmentar el realisme, com ombres, reflexions, transparències, . . .



Segona part del laboratori

Objectius

Extendrem el viewer que hem fet servir per a programar shaders, aprenent programació més avançada en OpenGL

Implementarem en OpenGL altres efectes per augmentar el realisme, com **ombres, reflexions, transparències, . . .**



Eines

C++

Qt5 (però no caldran gaires coneixements específics)

OpenGL (Core) + GLSL



Visualitzador i plugins

Us proporcionem un visualitzador senzill que haureu de completar via *plugins*.

Cada exercici de la llista consisteix a implementar un *plugin* (i potser shaders).



Avaluació

El control final de laboratori inclourà:

Exercicis de shaders pel visualitzador (fins ara heu fet servir un plugin específic: *shaderloader*.

Exercicis de plugins pel visualitzador

Els vostres plugins hauran de funcionar sobre el visualitzador original. Per tant, **no feu canvis al codi del nucli que us passem**



Avaluació

El control final de laboratori inclourà:

Exercicis de shaders pel visualitzador (fins ara heu fet servir un plugin específic: *shaderloader*.

Exercicis de plugins pel visualitzador

Els vostres plugins hauran de funcionar sobre el visualitzador original. Per tant, **no feu canvis al codi del nucli que us passem**



Estructura de directorios



Codi de partida del Visualitzador

```
Viewer/ ← Directori arrel de l'aplicació
├── all.pro
├── GLarena
├── GLarenaPL
└── GLarenaSL
├── plugins/
└── viewer/
```



Codi de partida del Visualitzador

```
Viewer/ ← Directori arrel de l'aplicació
|   all.pro ← arxiu pel qmake recursiu
|   GLarena
|   GLarenaPL
|
|   GLarenaSL
|   plugins/
|   viewer/
```



Codi de partida del Visualitzador

```
Viewer/ ← Directori arrel de l'aplicació
|   all.pro ← arxiu pel qmake recursiu
|   GLarena
|   GLarenaPL ← scripts per a engegar
|                 l'aplicació
|   GLarenaSL
|   plugins/
|   viewer/
```



Codi de partida del Visualitzador

```
Viewer/ ← Directori arrel de l'aplicació
└── all.pro ← arxiu pel qmake recursiu
└── GLarena
└── GLarenaPL ← scripts per a engegar
    l'aplicació
└── GLarenaSL
└── plugins/ ← fonts dels plugins
└── viewer/
```



Codi de partida del Visualitzador

```
Viewer/ ← Directori arrel de l'aplicació
└── all.pro ← arxiu pel qmake recursiu
└── GLarena
└── GLarenaPL ← scripts per a engegar
    l'aplicació
└── GLarenaSL
└── plugins/ ← fonts dels plugins
└── viewer/ ← fonts del nucli del Viewer
```



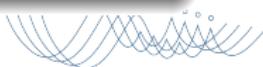
Codi de partida del Visualitzador

```
viewer/ ←D'aquí no heu de canviar res...
└── bin/
└── app/
    ├── app.pro
    └── main.cpp
└── core/
    ├── core.pro
    ├── include/
    └── src/
└── glwidget/
    ├── glwidget.pro
    ├── include/
    └── src/
└── interfaces/
    └── plugin.h
```



Codi de partida del Visualitzador

```
plugins/
└── bin/
└── common.pro
plugins.pro ← Cal editar-lo per afegir nous
                  plugins "permanentment"
alphablending/
└── alphablending.pro
    └── alphablending.h
    └── alphablending.cpp
navigate-default/
└── ...
...  
...
```



Codi de partida del Visualitzador

```
plugins/
└── bin/
└── common.pro
plugins.pro ← Cal editar-lo per afegir nous
                  plugins ‘‘permanentment’’

alphablending/ ← Un directori per cada
                  plugin
└── alphablending.pro
    └── alphablending.h
    └── alphablending.cpp
navigate-default/
└── ...
...  
...
```



Codi de partida del Visualitzador

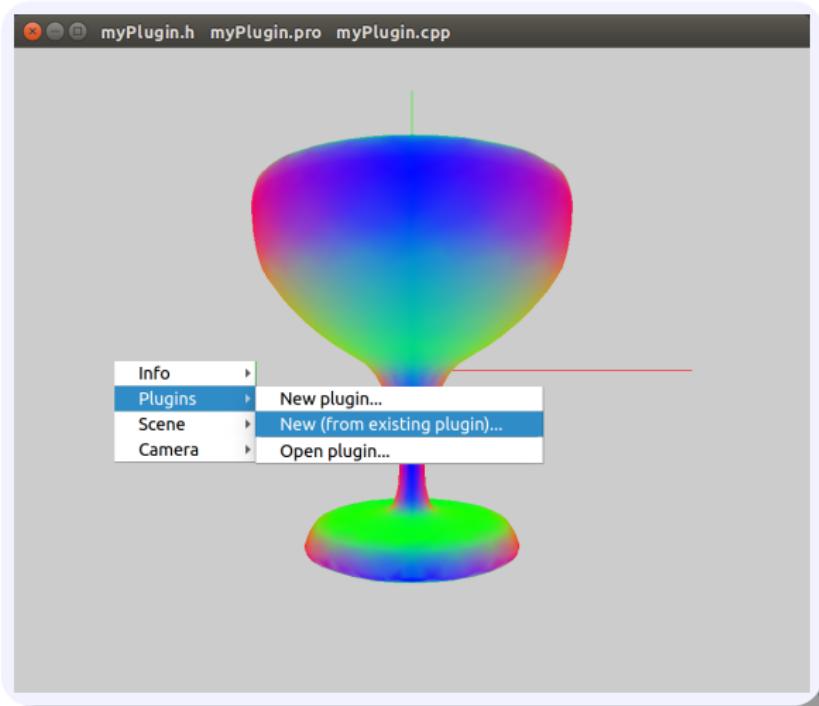
```
plugins/
└── bin/
└── common.pro
plugins.pro ← Cal editar-lo per afegir nous
                  plugins ‘‘permanentment’’

alphablending/ ← Un directori per cada
                  plugin
└── alphablending.pro ← S’ha de dir igual que
                          el directori
└── alphablending.h
└── alphablending.cpp
navigate-default/
└── ...
...
...
```



pluginLoader

Un plugin similar a shaderLoader, per a programar plugins



Algunes restriccions del pluginLoader

- No feu servir caràcters que no siguin alfanumèrics, llevat de la ratlla baixa '_', en els noms dels plugins
- pluginLoader no sap de shaders. Si en feu servir, haureu de gestionar aquells arxius vosaltres mateixos.
- Si heu de fer servir paths relatius, penseu que el vostre plugin serà executat, quan feu servir el pluginLoader, des del mateix directori del plugin.



Compilació i Execució



Procediment per a obtenir els binaris (viewer + plugins)

Seguiu les instruccions del racó. Resum:

```
tar -xzvf NewViewer_52d9d92.tgz  
cd NewViewer_52d9d92  
/opt/Qt/5.9.6/gcc_64/bin/qmake  
make -j
```

Executar el viewer:

```
./GLarenaSL (per provar shaders)  
./GLarenaPL (per provar plugins)
```



Adaptació a l'entorn

Per defecte, Viewer buscarà una sèrie de recursos en els directoris en què estan al laboratori, és a dir sota /assig/grau-g/... o en el seu directori arrel (el que conté GLarena*).

Podeu modificar aquest comportament definint variables d'entorn:

VIMAGE defineix l'executable a fer servir per mostrar imatges

VEDITOR l'editor que voleu fer servir per a editar shaders (si carregueu el shaderloader)

VMODELS el directori on trobar models

VTEXTURES el directori on trobar les textures

VPLUGINS els plugins a carregar en engegar.



Com afegir un Plugin



Crear nous plugins (manualment; no ho farem així)

Procediment per afegir un plugin 'MyEffect'

Crear el directori `plugins/my-effect` (eviteu usar espais)

Dins d'aquest directori:

 Editar el fitxer `my-effect.pro`

 Editar el fitxer `my-effect.h`

 Editar el fitxer `my-effect.cpp`

Afegiu una línia a `plugins/plugins.pro`

`SUBDIRS += my-effect`

[qmake +] make (des del directori viewer)



Amb pluginLoader...

Cal tenir tot el viewer correctament compilat a la màquina en què s'hi treballa

No cal preocupar-se de cap pas dels mencionats anteriorment, però convé ser conscient d'algunes particularitats:

- per restriccions en la descàrrega de plugins, pluginLoader afegirà un suffix al nom de la llibreria
- pluginLoader automàticament carregarà la nova versió del plugin cada cop que el recompili amb èxit.



Tipus de plugins

(es tracta d'una distinció semàntica: tant sols hi ha una interfície, comuna a tots els “tipus”)



(Algunes) Mètodes virtuals de la classe base dels plugins:

```
1      void onPluginLoad();
2      void onObjectAdd();
3      void onSceneClear();
4      void preFrame();
5      void postFrame();
6      bool drawScene();
7      bool drawObject(int);
8      bool paint();
9      void keyPressEvent(QKeyEvent *);
10     void mouseMoveEvent(QMouseEvent *);
```

11 . . .



Mètodes de la classe Plugin per accedir a altres components:

```
1 Scene* scene();  
2 Camera* camera();  
3 Plugin* drawPlugin();  
4 OpenGLWidget* glwidget();
```



Tipus de plugins

Effect Plugins

Canvien l'estat d'OpenGL abans i/o després de que es pinta l'escena.

Exemples: activar shaders, configurar textures, alpha blending...

Draw Plugins (sols un serà actiu)

Recorren els objectes per pintar les primitives de l'escena.

Exemples: dibuixar amb vertex arrays...

Action Plugins

Executen accions arbitràries en resposta a events (mouse, teclat).

Exemples: selecció d'objectes, control de la càmera virtual...

Render Plugins (sols un serà actiu)

Dibuixar un frame amb un o més passos de rendering.

Exemples: múltiples passos de rendering, shadow mapping...



Plugins per defecte

Per tal de ser utilitzable d'entrada, el viewer porta uns plugins per defecte, que podeu substituir per d'altres si és el cas:

render-default: un *render plugin* bàsic; sols esborra els buffers, crida al `drawPlugin` si està carregat, i afegeix els eixos coordenats.

drawvbong: un *draw plugin* que construeix VBOs/VAOs per cada objecte de l'escena, i ofereix un mètode `drawScene()` que recorre l'escena i dibuixa cada objecte fent-los servir.

navigate-default: un *action plugin* que implementa mecanismes bàsics per a navegar l'escena: rotació, zoom, pan.



Sessió 1: Effect plugins



Effect plugins

Mètodes típicament redefinits en els effect plugins (no necessàriament tots):

```
virtual void preFrame();  
virtual void postFrame();  
virtual void onPluginLoad();  
virtual void onObjectAdd();
```

Accés a les dades de l'aplicació:

```
GLWidget* glwidget();  
Scene* scene();  
Camera* camera();
```



Exemples d'accés als objectes de l'aplicació

scene()->objects().size() // num objectes

camera()->getObs() // pos de l'observador

glwidget()->defaultProgram()



Exemples d'accés als objectes de l'aplicació

```
scene()->objects().size() // num objectes  
camera()->getObs() // pos de l'observador  
glwidget()->defaultProgram()
```



Exemples d'accés als objectes de l'aplicació

```
scene()->objects().size() // num objectes  
camera()->getObs() // pos de l'observador  
glwidget()->defaultProgram()
```



Exemples d'effect plugins: 1/3



alphablending

alphablending.pro

```
1 TARGET      = $$qtLibraryTarget(alphablending)
2 include(../common.pro)
```



alphablending.h

```
1 #ifndef _ALPHABLENDING_H
2 #define _ALPHABLENDING_H
3 #include "plugin.h"
4
5 class AlphaBlending: public QObject, public Plugin
6 {
7     Q_OBJECT
8     Q_PLUGIN_METADATA(IID "Plugin")
9     Q_INTERFACES(Plugin)
10
11 public:
12     void preFrame();
13     void postFrame();
14 };
15 #endif
```



alphablending.cpp

```
1 #include "alphablending.h"
2 #include "glwidget.h"
3
4 void AlphaBlending::preFrame() {
5     glDisable(GL_DEPTH_TEST);
6     glBlendEquation(GL_FUNC_ADD);
7     glBlendFunc(GL_SRC_ALPHA, GL_ONE);
8     glEnable(GL_CULL_FACE);
9     glEnable(GL_BLEND);
10 }
11
12 void AlphaBlending::postFrame() {
13     glEnable(GL_DEPTH_TEST);
14     glDisable(GL_BLEND);
15 }
```



Exemples d'effect plugins: 2/3



effect-crt

effect-crt.pro

```
1 TARGET      = $$qtLibraryTarget(effect-crt)
2 include(../common.pro)
```



effectcrt.h

```
1 #ifndef _EFFECTCRT_H
2 #define _EFFECTCRT_H
3 #include "plugin.h"
4 #include <QOpenGLShader>
5 #include <QOpenGLShaderProgram>
6 class EffectCRT : public QObject, public Plugin
7 {
8     Q_OBJECT
9     Q_PLUGIN_METADATA(IID "Plugin")
10    Q_INTERFACES(Plugin)
11 public:
12     void onPluginLoad();
13     void preFrame();
14     void postFrame();
15 private:
16     QOpenGLShaderProgram* program;
17     QOpenGLShader *fs, *vs;
18 };
```



effectcrt.cpp

```
1 #include "effectcrt.h"
2
3 void EffectCRT::onPluginLoad() {
4     glwidget()->makeCurrent(); // !!!
5     QString vs_src =
6         "#version 330 core\n"
7         "uniform mat4 modelViewProjectionMatrix;" +
8         "in vec3 vertex;" +
9         "in vec3 color;" +
10        "out vec4 col;" +
11        "void main() {"
12        "    gl_Position = modelViewProjectionMatrix *"
13        "                  vec4(vertex,1.0);"
14        "    col=vec4(color,1.0);"
15        "}";
16     vs = new QOpenGLShader(QOpenGLShader::Vertex, this);
17     vs->compileSourceCode(vs_src);
18     cout << "VS log:" << vs->log().toStdString() << endl;
```

```
19 QString fs_src =  
20     "#version 330 core\n"  
21     "out vec4 fragColor;"  
22     "in vec4 col;"  
23     "uniform int n;"  
24     "void main() {"  
25     "    if (mod((gl_FragCoord.y-0.5), float(n)) > 0.0) dis  
26     "        fragColor=col;"  
27     "}";  
28     fs = new QOpenGLShader(QOpenGLShader::Fragment, this);  
29     fs->compileSourceCode(fs_src);  
30     cout << "FS log:" << fs->log().toStdString() << endl;  
31     program = new QOpenGLShaderProgram(this);  
32     program->addShader(vs); program->addShader(fs);  
33     program->link();  
34     cout << "Link log:" << program->log().toStdString() <<  
35 }
```



effect-crt.cpp...

```
36 void EffectCRT::preFrame()
37 {
38     // bind shader and define uniforms
39     program->bind();
40     program->setUniformValue("n", 6);
41     QMatrix4x4 MVP = camera()->projectionMatrix() *
42                         camera()->viewMatrix();
43     program->setUniformValue(
44             "modelViewProjectionMatrix", MVP);
45 }
46
47 void EffectCRT::postFrame()
48 {
49     // unbind shader
50     program->release();
51 }
```



Exemples d'effect plugins: 3/3



showHelpNg

showHelpNg.pro

```
1 TARGET      = $$qtLibraryTarget(showHelpNg)
2 include(../common.pro)
```



showHelpNg.h

```
1 #ifndef _SHOWHELPNG_H
2 #define _SHOWHELPNG_H
3
4 #include "plugin.h"
5 #include <QPainter>
6
7 class ShowHelpNg : public QObject, Plugin
8 {
9     Q_OBJECT
10    Q_PLUGIN_METADATA(IID "Plugin")
11    Q_INTERFACES(Plugin)
12
13 public:
14     void postFrame() Q_DECL_OVERRIDE;
15 private:
16     QPainter painter;
17 };
18 #endif
```



part of showHelpNg.cpp

```
1 #include "showHelpNg.h"
2 #include "glwidget.h"
3
4 void ShowHelpNg::postFrame()
5 {
6     QFont font;
7     font.setPixelSize(32);
8     painter.begin(glwidget());
9     painter.setFont(font);
10    int x = 15;
11    int y = 40;
12    painter.drawText(x, y, QString("L - Load object"
13                                "          A - Add plugin"));
14    painter.end();
15 }
```



Flux de control

Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`

Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats

Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats

Els events de ratolí i teclat (`keyPressEvent()`...
`mouseMoveEvent()`...) es propaguen a tots els plugins carregats

`GLWidget::paint()` crida:

`bind()` dels shaders per defecte

`setUniformValue()` pels uniforms que fan servir els shaders per defecte

`preFrame()` de tots els plugins

`paint()` del **darrer plugin carregat que l'implementi**

`postFrame()` de tots els plugins



Classes de core/



Classes

Als directoris viewer/core/{include,src}

box: Caixes englobants

camera: Un embolcall per a una càmera rudimentària

face: Cares d'un model

object: objecte (inclou codi per a carregar .obj)

point: Punts. Alias de QVector3D amb operador d'escriptura per a missatges de debug, etc.

scene: Model simple d'escena usat pel GLWidget.

vector: Altre alias de QVector3D amb operador d'escriptura.

vertex: Model de vèrtex usat a les demés classes.



Classes

Per a representar l'escena:

Als directoris `viewer/core/{include,src}`

box: Caixes englobants

camera: Un embolcall per a una càmera rudimentària

face: Cares d'un model

object: objecte (inclou codi per a carregar .obj)

point: Punts. Alias de QVector3D amb operador d'escriptura per a missatges de debug, etc.

scene: Model simple d'escena usat pel GLWidget.

vector: Altre alias de QVector3D amb operador d'escriptura.

vertex: Model de vèrtex usat a les demés classes.



Classes

Support a la geometria:

Als directoris `viewer/core/{include,src}`

box: Caixes englobants

camera: Un embolcall per a una càmera rudimentària

face: Cares d'un model

object: objecte (inclou codi per a carregar .obj)

point: Punts. Alias de QVector3D amb operador d'escriptura per a missatges de debug, etc.

scene: Model simple d'escena usat pel GLWidget.

vector: Altre alias de QVector3D amb operador d'escriptura.

vertex: Model de vèrtex usat a les demés classes.



Vector, Punt

Vector

```
Vector ( qreal xpos, qreal ypos, qreal zpos )  
qreal length () const  
void normalize ()  
Point normalized () const  
void setX ( qreal x )  
void setY ( qreal y )  
void setZ ( qreal z )  
qreal x () const  
qreal y () const  
qreal z () const  
Vector crossProduct ( const QVector3D & v1, const QVector3D & v2 )  
qreal dotProduct ( const QVector3D & v1, const QVector3D & v2 )  
const Vector operator* ( const QVector3D & vector, qreal factor )
```



Vector, Point

Vector

```
1     Vector v(1.0, 0.0, 0.0);
2     float l = v.length();
3     v.normalize();
4     Vector w = v.normalized();
5     v.setX(2.0);
6     v.setY(-3.0);
7     v.setZ(1.0);
8     cout << "[" << v << "]" << endl;
9     Vector u = QVector3D::crossProduct(v,w);
10    float dot = QVector3D::dotProduct(v,w);
11    Vector u = v + 2.5*w;
```



Vector, Point

Point

```
1 Point p(1.0, 0.0, 0.0);
2 p.setX(0.0);
3 p.setY(0.0);
4 p.setZ(1.0);
5 cout << "(" << p << ")" << endl;
6 // point substraction (returns a Vector)
7 Vector v = p - q;
8 // barycentric combination:
9 Point r = 0.4*p + 0.6*q;
```



Box

```
1 class Box
2 {
3 public:
4     Box(const Point& point=Point());
5     Box(const Point& minimum, const Point& maximum);
6
7     void expand(const Point& p); // incloure un punt
8     void expand(const Box& p); // incloure una caps
9
10    void render(); // dibuixa en filferros
11    Point center() const; // centre de la caps
12    float radius() const; // meitat de la diagonal
13    Point min() const;
14    Point max() const;
15 ...};
```



Scene

Scene té una col·lecció d'objectes 3D

```
1 class Scene
2 {
3     public:
4     Scene();
5
6     const vector<Object>& objects() const;
7     vector<Object>& objects();
8     void addObject(Object &);
9     void clear();
10
11    int selectedObject() const;
12    void setSelectedObject(int index);
13    void computeBoundingBox();
14    Box boundingBox() const;
15 };
```

Object

Object té un vector de cares i un vector de vèrtexs

```
1 class Object {  
2     public:  
3         ...  
4         Box boundingBox() const;  
5         const vector<Face>& faces() const;  
6         const vector<Vertex>& vertices() const;  
7         void computeNormals();           // normals *per-cara*  
8         void computeBoundingBox();  
9         void applyGT(const QMatrix4x4& mat);  
10  
11    private:  
12        vector<Vertex> pvertices;  
13        vector<Face> pfaces;  
14        Box pboundingBox;  
15};
```

Face

Face té una seqüència ordenada de 3 o més índexs a vèrtex

```
1 class Face
2 {
3     public:
4         ...
5         int numVertices() const;
6         int vertexIndex(int i) const;
7         Vector normal() const;
8         void addVertexIndex(int i);
9         void computeNormal(const vector<Vertex> &);
10    private:
11        Vector pnormal;
12        vector<int> pvertices; // indexes dels vertexs
13 };
```



Vertex

Simplement les coordonées d'un point

```
1 class Vertex
2 {
3     Vertex(const Point&);
4     Point coord() const;
5     void setCoord(const Point& coord);
6
7 private:
8     Point pcoord;
9 };
```



APIs per treballar amb shaders



Support per a shaders a Qt

Podeu fer servir `QOpenGLShader` i `QOpenGLShaderProgram`

```
1 QOpenGLShader shader(QOpenGLShader::Vertex);  
2 shader.compileSourceCode(code);  
3 shader.compileSourceFile(filename);  
4 ...  
5 QOpenGLShaderProgram *program = new QOpenGLShaderProgram();  
6 program->addShader(shader);  
7 ...  
8 program->link();  
9 ...  
10 program->bind();  
11 ...  
12 program->release();
```



Alguns mètodes de QOpenGLShaderProgram

Atributs i Uniforms

```
1 int attributeLocation(const char * name ) const;  
2 void setAttributeValue(int location, T value);  
3  
4 int uniformLocation(const char * name ) const;  
5 void setUniformValue(int location, T value);
```

Molts altres mètodes útils

```
1 bool isLinked() const;  
2 QString log() const;  
3 void setGeometryOutputType(GLenum outputType);
```



QOpenGLShader és semblant

Interfície semblant:

```
1 bool isCompiled() const;  
2 QString log() const;
```



Vertex Array Objects (VAOs)

C. Andujar, A. Vinacua

Nov 2019

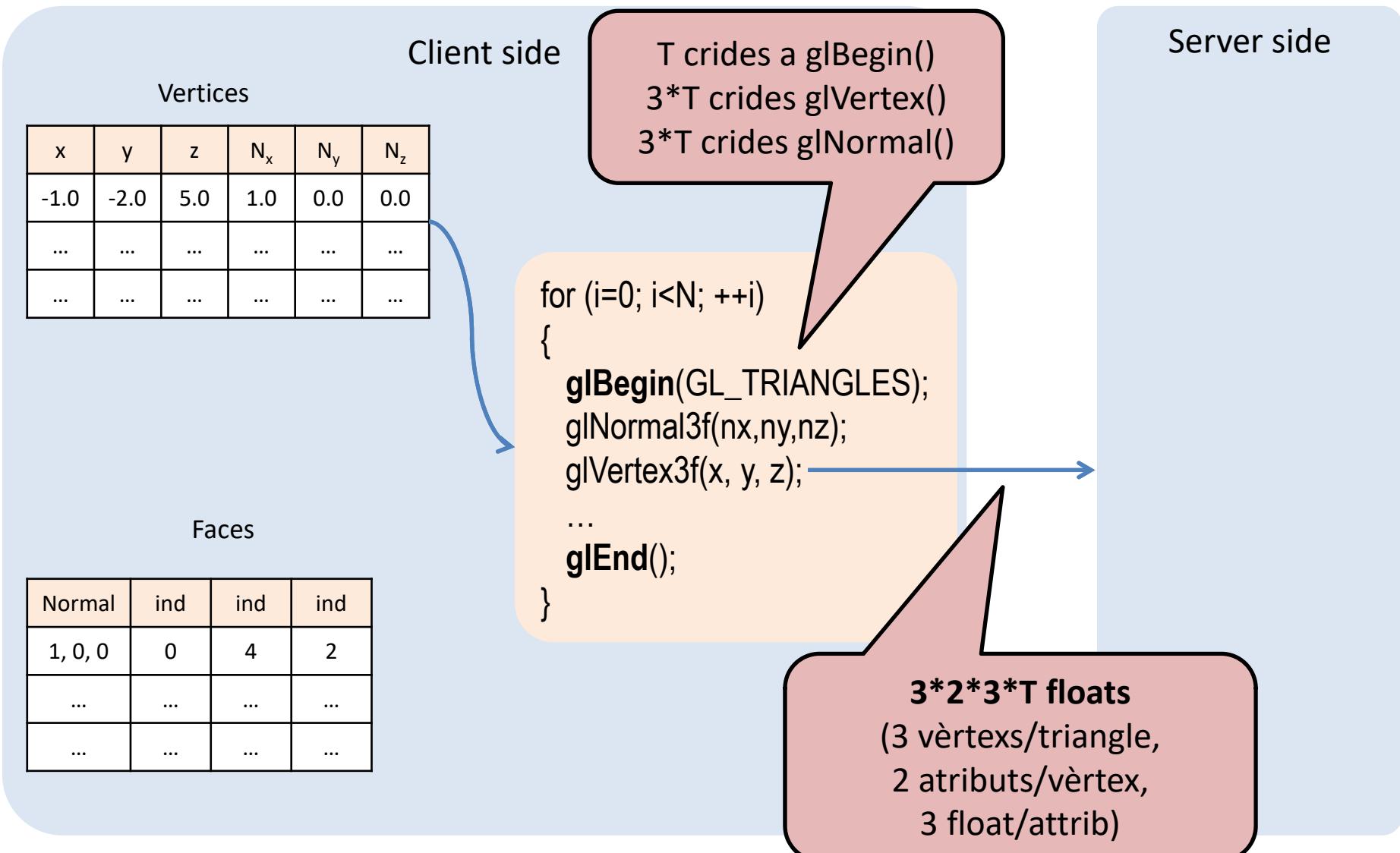
Formes de pintar geometria

- Mode immediat (`glBegin`,`glEnd`) (**Compatibility**)
- Usant Vertex Arrays (VAs) (**Compatibility, Core**)
- Usant Vertex Array Object (VAOs) (**Compatibility, Core**)

Mode immédiat

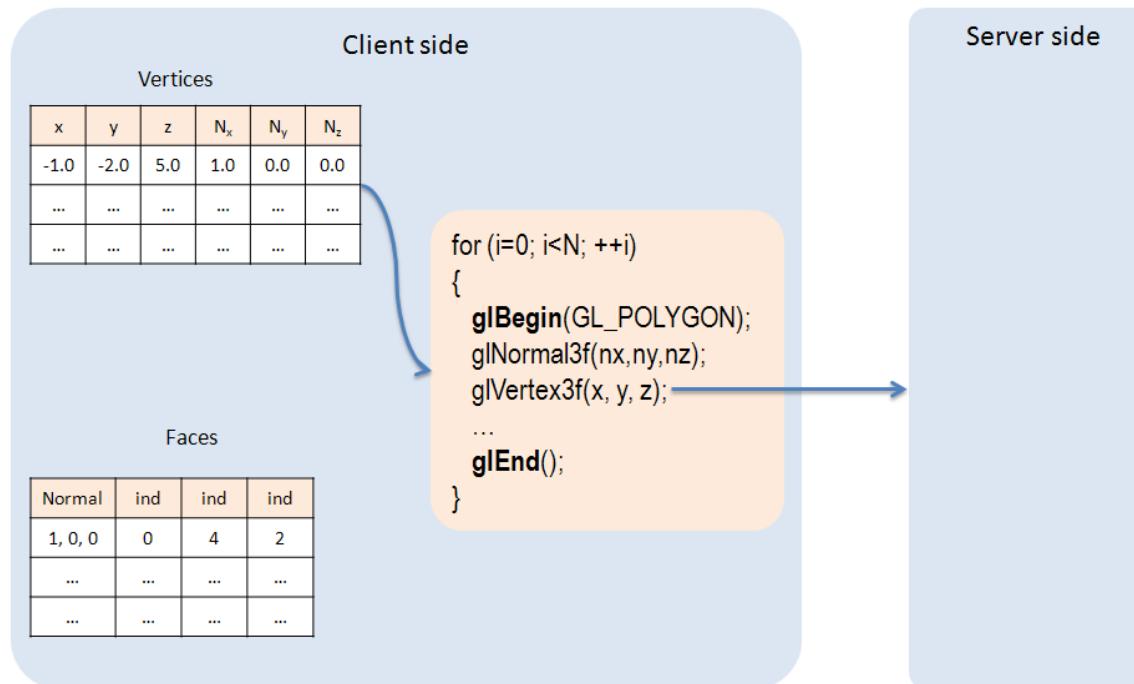
```
for (i=0; i<T; ++i) {  
    glBegin(GL_TRIANGLES);  
    glNormal3f(...);  
    glVertex3f(...);  
  
    glNormal3f(...);  
    glVertex3f(...);  
  
    glNormal3f(...);  
    glVertex3f(...);  
    glEnd();  
}
```

Mode immédiat



Mode immediat

- Senzill, fàcil de depurar, flexible...
- Moltes crides a funcions
- Cal transferir totes les dades cada frame

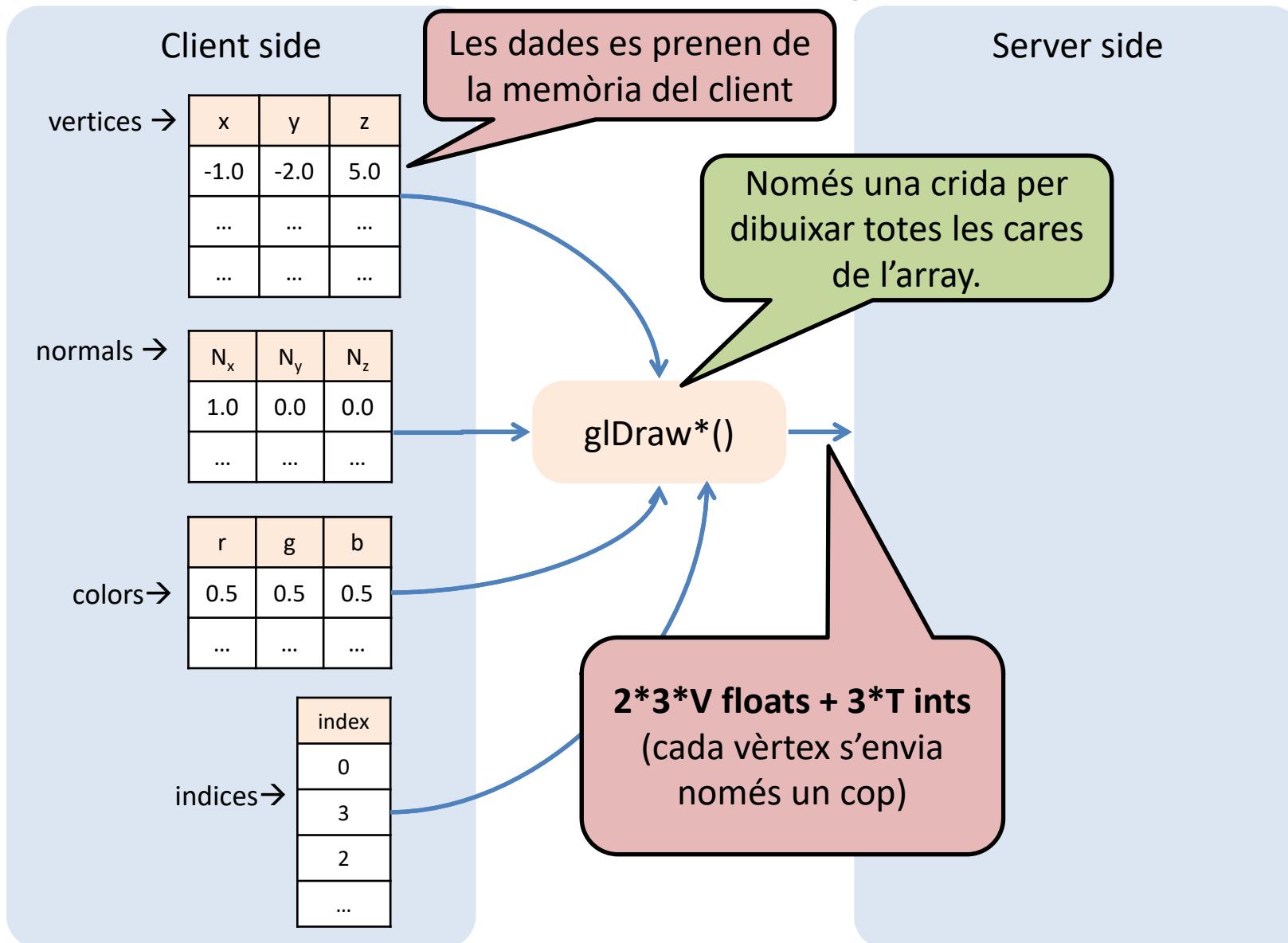


Vertex Arrays

Objectius:

- Reduir crides a OpenGL
- Enviar un cop cada vèrtex

Vertex Arrays

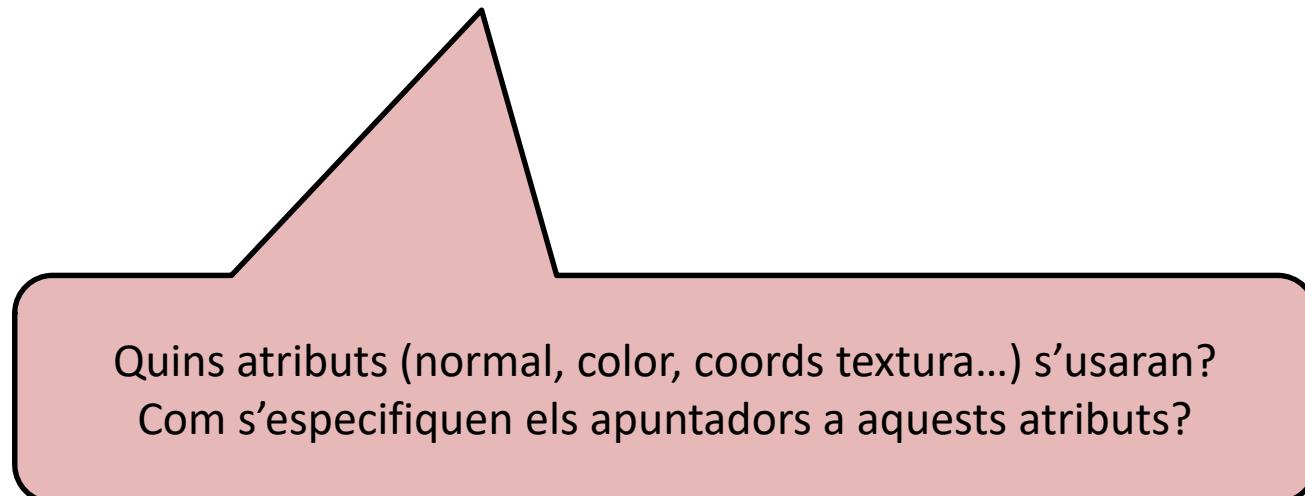


Vertex Arrays

```
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, indices)
```

① ② ③ ④

- ① És la primitiva: GL_TRIANGLES, GL_QUADS ...
- ② És el número d'índexos a l'array (ex. 12 triangles → $12 \times 3 = 36$)
- ③ És el tipus dels índexs (normalment GL_UNSIGNED_INT)
- ④ És l'apuntador a l'array amb els índexs (que haurem definit previament)



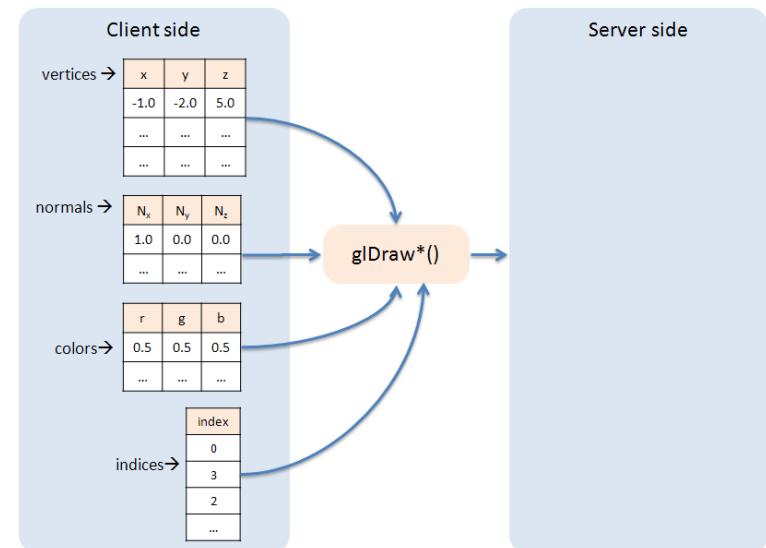
Vertex Arrays

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0,  
(GLvoid*)verts);  
glEnableVertexAttribArray(0);
```

```
void glVertexAttribPointer(  
    GLuint index,           // VS: layout (location = 0) in vec3 vertex;  
    GLint size,             // Num de coordenades (1,2,3,4)  
    GLenum type,            // Tipus de cada coordenada: GL_FLOAT ...  
    GLboolean normalized,   // Per convertir valors a [0,1]  
    GLsizei stride,         // Normalment 0 (un array per cada atribut)  
    const GLvoid* pointer); // Apuntador a les dades
```

Vertex Arrays - resum

- Una única crida a funció (per model 3D)
- Els vèrtexs s'envien un cop
- Menys flexible que el mode immediat
- Encara cal transferir moltes dades cada frame



Vertex buffer object

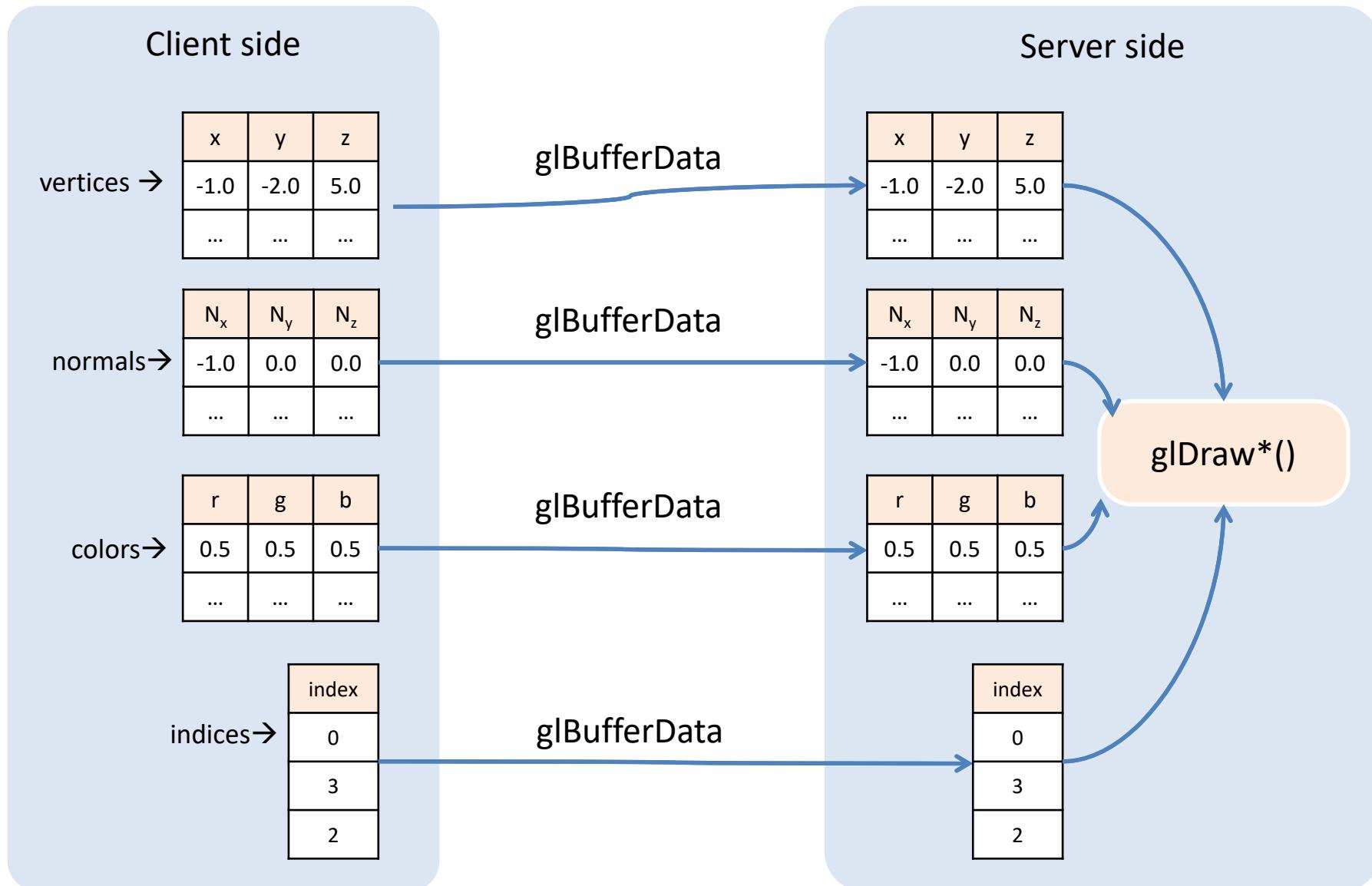
Objectiu:

- Evitar transferir les dades cada frame

Idea:

- Emmagatzemar les dades del VA al servidor!

Vertex buffer object



EXEMPLE 1 – USANT INDEXOS

Setup 1/3

```
// Step 1: Create and fill STL vectors(coords, normals...)
vector<float> vertices;    // (x,y,z)
vector<float> normals;     // (nx,ny,nz)
vector<float> colors;      // (r,g,b)
vector<float> texCoords;   // (s,t)
vector<unsigned int> indices; // i0,i1,i2, i3,i4,i5...
for(...) {
    vertices.push_back(x);
    vertices.push_back(y);
    vertices.push_back(z);
for(...) {
    indices.push_back(index);
```

Setup 2/3

```
// Step 2: Create VAO & empty VBOs
GLuint VAO;
g.glGenVertexArrays(1,&VAO);

GLuint coordBufferID;
g glGenBuffers(1, &coordBufferID);

GLuint normalBufferID;
g glGenBuffers(1, &normalBufferID);

...
GLuint indexBufferID;
g glGenBuffers(1, &indexBufferID);
```

Setup 3/3

```
// Step 3: Define VBO data (coords,normals,indices)
g glBindVertexArray(VAO);
g glBindBuffer(GL_ARRAY_BUFFER, coordBufferID);
g glBufferData(GL_ARRAY_BUFFER, sizeof(float)*vertices.size(), &vertices[0],
    GL_STATIC_DRAW);
g glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
g glEnableVertexAttribArray(0);

g glBindBuffer(GL_ARRAY_BUFFER, normalBufferID);
g glBufferData(GL_ARRAY_BUFFER, sizeof(float)*normals.size(), &normals[0],
    GL_STATIC_DRAW);
g glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, 0);
g glEnableVertexAttribArray(1);

...
g glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexBuffersID);
g glBufferData(GL_ELEMENT_ARRAY_BUFFER,
    sizeof(int)*indices.size(), &indices[0], GL_STATIC_DRAW);

g glBindVertexArray(0);
```

Draw (amb índexos)

```
// Draw a single instance of the 3D model
g glBindVertexArray(VAO);
g glDrawElements(GL_TRIANGLES, numIndices, GL_UNSIGNED_INT, (GLvoid*)0);
//numIndices=indices.size()
g glBindVertexArray(0);
```

```
// Draw multiple instances of the same 3D model
g glBindVertexArray(VAO);
g glDrawElementsInstanced(GL_TRIANGLES, numIndices, GL_UNSIGNED_INT,
(GLvoid*)0, numInstances);
g glBindVertexArray(0);
```

VS: int gl_InstanceID → instance number (0...numInstances-1)

Clean up

```
// Clean up  
g.glDeleteBuffers(1, &coordBufferID);  
g.glDeleteBuffers(1, &normalBufferID);  
...  
g.glDeleteBuffers(1, &indexBufferID);  
  
g.glDeleteVertexArrays(1, &VAO);
```

EXEMPLE 2 – SENSE USAR INDEXOS

Setup 1/3

```
// Step 1: Create and fill STL vectors(coords, normals...)
vector<float> vertices;      // (x,y,z)
vector<float> normals;        // (nx,ny,nz)
vector<float> colors;         // (r,g,b)
vector<float> texCoords;      // (s,t)
vector<unsigned int> indices; // i0,i1,i2, i3,i4,i5...
for(...) {
    vertices.push_back(x); // vertexs duplicates!
    vertices.push_back(y);
    vertices.push_back(z);
for(...) {
    indices.push_back(index);
```

Setup 2/3

```
// Step 2: Create VAO & empty VBOs
GLuint VAO;
g.glGenVertexArrays(1,&VAO);

GLuint coordBufferID;
g glGenBuffers(1, &coordBufferID);

GLuint normalBufferID;
g glGenBuffers(1, &normalBufferID);

...
GLuint indexBufferID;
g glGenBuffers(1, &indexBufferID);
```

Setup 3/3

```
// Step 3: Define VBO data (coords,normals,indices)
g glBindVertexArray(VAO);
g glBindBuffer(GL_ARRAY_BUFFER, coordBufferID);
g glBufferData(GL_ARRAY_BUFFER, sizeof(float)*vertices.size(), &vertices[0],
    GL_STATIC_DRAW);
g glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
g glEnableVertexAttribArray(0);

g glBindBuffer(GL_ARRAY_BUFFER, normalBufferID);
g glBufferData(GL_ARRAY_BUFFER, sizeof(float)*normals.size(), &normals[0],
    GL_STATIC_DRAW);
g glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, 0);
g glEnableVertexAttribArray(1);

...
g glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexBuffersID);
g glBufferData(GL_ELEMENT_ARRAY_BUFFER,
    sizeof(int)*indices.size(), &indices[0], GL_STATIC_DRAW);

g glBindVertexArray(0);
```

Draw (sense índexos)

```
// Draw a single instance of the 3D model  
g glBindVertexArray(VAO);  
g glDrawArrays(GL_TRIANGLES, 0, numVertices);  
g glBindVertexArray(0);
```

```
// Draw multiple instances of the same 3D model  
g glBindVertexArray(VAO);  
g glDrawArraysInstanced(GL_TRIANGLES, 0, numVertices, numInstances);  
g glBindVertexArray(0);
```

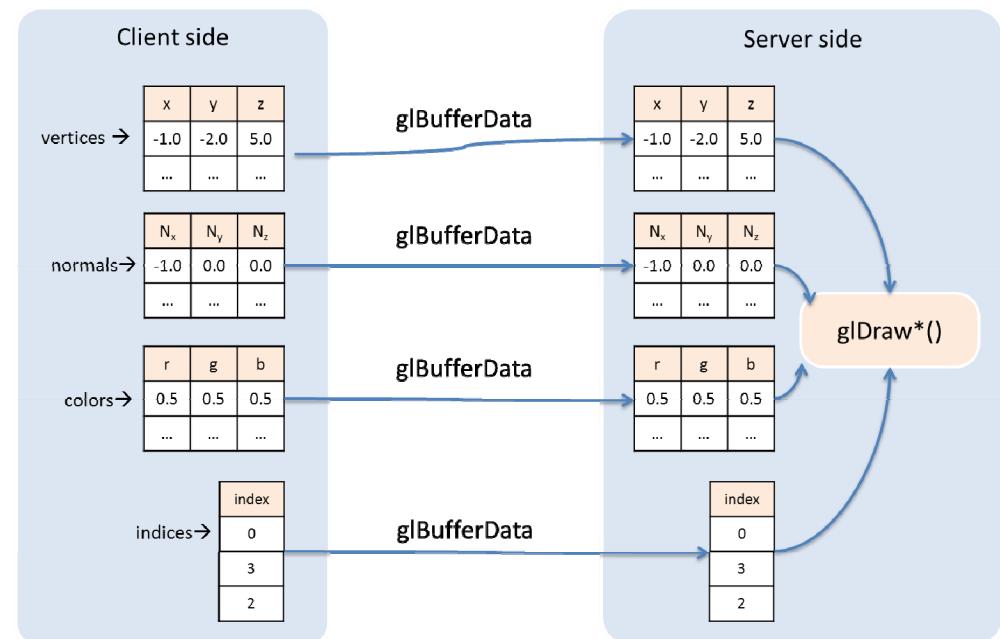
VS: int gl_InstanceID → instance number (0...numInstances-1)

Clean up

```
// Clean up  
g.glDeleteBuffers(1, &coordBufferID);  
g.glDeleteBuffers(1, &normalBufferID);  
...  
g.glDeleteBuffers(1, &indexBufferID);  
g.glDeleteVertexArrays(1, &VAO);
```

Vertex Buffer Objects - resum

- Una única crida a funció
- Els vèrtexs s'envien (i processen) un cop (*)
- Les dades es transfereixen al servidor
- Menys flexible que el mode immediat



Render plugins

À. Vinacua, C. Andújar i professors de Gràfics

novembre de 2019

Tipus de plugins

(es tracta d'una distinció semàntica: tant sols hi ha una interfície, comuna a tots els “tipus”)



Tipus de plugins

- Effect Plugins
 - Canvien l'estat d'OpenGL abans i/o després de que es pinta l'escena.
 - Exemples: activar shaders, configurar textures, alpha blending...
- Draw Plugins (Sols un serà actiu)
 - Recorren els objectes per pintar les primitives de l'escena.
 - Exemples: dibuixar amb vertex arrays...
- Action Plugins
 - Executen accions arbitràries en resposta a events (mouse, teclat).
 - Exemples: selecció d'objectes, control de la càmera virtual...
- Render Plugins (Sols un serà actiu)
 - Dibuixar un frame amb un o més passos de rendering.
 - Exemples: múltiples passos de rendering, shadow mapping...



Fluxe de control

- Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`
- Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats
- Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats
- Els events de ratolí i teclat (`keyPressEvent()`...
`mouseMoveEvent()`...) es propaguen a tots els plugins carregats
- `GLWidget::paintGL()` crida:
 - `bind()` dels shaders per defecte
 - `setUniformValue()` pels uniforms que fan servir els shaders per defecte
 - `preFrame()` de tots els plugins
 - `paintGL()` del darrer plugin carregat que l'implementi
 - `postFrame()` de tots els plugins



Fluxe de control

- Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`
- Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats
- Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats
- Els events de ratolí i teclat (`keyPressEvent()`...
`mouseMoveEvent()`...) es propaguen a tots els plugins carregats
- `GLWidget::paintGL()` crida:
 - `bind()` dels shaders per defecte
 - `setUniformValue()` pels uniforms que fan servir els shaders per defecte
 - `preFrame()` de tots els plugins
 - `paintGL()` del darrer plugin carregat que l'implementi
 - `postFrame()` de tots els plugins



Flux de control

- Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`
- Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats
- Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats
- Els events de ratolí i teclat (`keyPressEvent()`...
`mouseMoveEvent()`...) es propaguen a tots els plugins carregats
- `GLWidget::paintGL()` crida:
 - `bind()` dels shaders per defecte
 - `setUniformValue()` pels uniforms que fan servir els shaders per defecte
 - `preFrame()` de tots els plugins
 - `paintGL()` del darrer plugin carregat que l'implementi
 - `postFrame()` de tots els plugins



Flux de control

- Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`
- Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats
- Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats
- Els events de ratolí i teclat (`keyPressEvent()`...
`mouseMoveEvent()`...) es propaguen a tots els plugins carregats
- `GLWidget::paintGL()` crida:
 - `bind()` dels shaders per defecte
 - `setUniformValue()` pels uniforms que fan servir els shaders per defecte
 - `preFrame()` de tots els plugins
 - `paintGL()` del darrer plugin carregat que l'implementi
 - `postFrame()` de tots els plugins



Flux de control

- Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`
- Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats
- Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats
- Els events de ratolí i teclat (`keyPressEvent()`...
`mouseMoveEvent()`...) es propaguen a tots els plugins carregats
- `GLWidget::paintGL()` crida:
 - `bind()` dels shaders per defecte
 - `setUniformValue()` pels uniforms que fan servir els shaders per defecte
 - `preFrame()` de tots els plugins
 - `paintGL()` del **darrer plugin carregat que l'implementi**
 - `postFrame()` de tots els plugins



Flux de control

- Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`
- Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats
- Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats
- Els events de ratolí i teclat (`keyPressEvent()`...
`mouseMoveEvent()`...) es propaguen a tots els plugins carregats
- `GLWidget::paintGL()` crida:
 - `bind()` dels shaders per defecte
 - `setUniformValue()` pels uniforms que fan servir els shaders per defecte
 - `preFrame()` de tots els plugins
 - `paintGL()` del **darrer plugin carregat que l'implementi**
 - `postFrame()` de tots els plugins



Flux de control

- Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`
- Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats
- Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats
- Els events de ratolí i teclat (`keyPressEvent()`...
`mouseMoveEvent()`...) es propaguen a tots els plugins carregats
- `GLWidget::paintGL()` crida:
 - `bind()` dels shaders per defecte
 - `setUniformValue()` pels uniforms que fan servir els shaders per defecte
 - `preFrame()` de tots els plugins
 - `paintGL()` del **darrer plugin carregat que l'implementi**
 - `postFrame()` de tots els plugins



Flux de control

- Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`
- Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats
- Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats
- Els events de ratolí i teclat (`keyPressEvent()`...
`mouseMoveEvent()`...) es propaguen a tots els plugins carregats
- `GLWidget::paintGL()` crida:
 - `bind()` dels shaders per defecte
 - `setUniformValue()` pels uniforms que fan servir els shaders per defecte
 - `preFrame()` de tots els plugins
 - `paintGL()` del **darrer plugin carregat que l'implementi**
 - `postFrame()` de tots els plugins



Flux de control

- Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`
- Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats
- Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats
- Els events de ratolí i teclat (`keyPressEvent()`...
`mouseMoveEvent()`...) es propaguen a tots els plugins carregats
- `GLWidget::paintGL()` crida:
 - `bind()` dels shaders per defecte
 - `setUniformValue()` pels uniforms que fan servir els shaders per defecte
 - `preFrame()` de tots els plugins
 - `paintGL()` del **darrer plugin carregat que l'implementi**
 - `postFrame()` de tots els plugins



Flux de control

- Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`
- Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats
- Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats
- Els events de ratolí i teclat (`keyPressEvent()`...
`mouseMoveEvent()`...) es propaguen a tots els plugins carregats
- `GLWidget::paintGL()` crida:
 - `bind()` dels shaders per defecte
 - `setUniformValue()` pels uniforms que fan servir els shaders per defecte
 - `preFrame()` de tots els plugins
 - `paintGL()` del **darrer plugin carregat que l'implementi**
 - `postFrame()` de tots els plugins



Laboratori de Gràfics, Action plugins.

À. Vinacua, C. Andújar i professors de Gràfics

23 de novembre de 2015

Tipus de plugins



Tipus de plugins

- Effect Plugins

- Canvien l'estat d'OpenGL abans i/o després de que es pinta l'escena.
- Exemples: activar shaders, configurar textures, alpha blending...

- Draw Plugins

- Recorren els objectes per pintar les primitives de l'escena.
- Exemples: dibuixar amb VBO...

- Action Plugins

- Executen accions arbitràries en resposta a events (mouse, teclat).
- Exemples: selecció d'objectes, control de la càmera virtual...

- Render Plugins

- Dibuixar un frame amb un o més passos de rendering.
- Exemples: múltiples passos de rendering, shadow mapping...



Tipus de plugins

- Effect Plugins
 - Canvien l'estat d'OpenGL abans i/o després de que es pinta l'escena.
 - Exemples: activar shaders, configurar textures, alpha blending...
- Draw Plugins
 - Recorren els objectes per pintar les primitives de l'escena.
 - Exemples: dibuixar amb VBO...
- Action Plugins
 - Executen accions arbitràries en resposta a events (mouse, teclat).
 - Exemples: selecció d'objectes, control de la càmera virtual...
- Render Plugins
 - Dibuixar un frame amb un o més passos de rendering.
 - Exemples: múltiples passos de rendering, shadow mapping...



Tipus de plugins

- Effect Plugins
 - Canvien l'estat d'OpenGL abans i/o després de que es pinta l'escena.
 - Exemples: activar shaders, configurar textures, alpha blending...
- Draw Plugins
 - Recorren els objectes per pintar les primitives de l'escena.
 - Exemples: dibuixar amb VBO...
- Action Plugins
 - Executen accions arbitràries en resposta a events (mouse, teclat).
 - Exemples: selecció d'objectes, control de la càmera virtual...
- Render Plugins
 - Dibuixar un frame amb un o més passos de rendering.
 - Exemples: múltiples passos de rendering, shadow mapping...



Tipus de plugins

- Effect Plugins
 - Canvien l'estat d'OpenGL abans i/o després de que es pinta l'escena.
 - Exemples: activar shaders, configurar textures, alpha blending...
- Draw Plugins
 - Recorren els objectes per pintar les primitives de l'escena.
 - Exemples: dibuixar amb VBO...
- Action Plugins
 - Executen accions arbitràries en resposta a events (mouse, teclat).
 - Exemples: selecció d'objectes, control de la càmera virtual...
- Render Plugins
 - Dibuixar un frame amb un o més passos de rendering.
 - Exemples: múltiples passos de rendering, shadow mapping...



Aquesta sessió: Action plugins



Action plugins

Mètodes propis

- `virtual void keyPressEvent (QKeyEvent *)`
- `virtual void keyReleaseEvent (QKeyEvent *)`
- `virtual void mouseMoveEvent (QMouseEvent *)`
- `virtual void mousePressEvent (QMouseEvent *)`
- `virtual void mouseReleaseEvent (QMouseEvent *)`
- `virtual void wheelEvent (QWheelEvent *)`

Mètodes/Atributs heredats

- `virtual void onPluginLoad()`
- `virtual void onObjectAdd()`
- `GLWidget* glwidget(); // dóna accés a l'escena i la càmera`
- `scene()->objects().size() // num objectes`
- `camera()->getObs() // pos de l'observador`

Action plugins

Mètodes propis

- virtual void keyPressEvent (QKeyEvent *)
- virtual void keyReleaseEvent (QKeyEvent *)
- virtual void mouseMoveEvent (QMouseEvent *)
- virtual void mousePressEvent (QMouseEvent *)
- virtual void mouseReleaseEvent (QMouseEvent *)
- virtual void wheelEvent (QWheelEvent *)

Mètodes/Atributs heredats

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget(); // dóna accés a l'escena i la càmera
- scene()->objects().size() // num objectes
- camera()->getObs() // pos de l'observador

Action plugins

Mètodes propis

- virtual void keyPressEvent (QKeyEvent *)
- virtual void keyReleaseEvent (QKeyEvent *)
- virtual void mouseMoveEvent (QMouseEvent *)
- virtual void mousePressEvent (QMouseEvent *)
- virtual void mouseReleaseEvent (QMouseEvent *)
- virtual void wheelEvent (QWheelEvent *)

Mètodes/Atributs heredats

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget(); // dóna accés a l'escena i la càmera
- scene()->objects().size() // num objectes
- camera()->getObs() // pos de l'observador

Action plugins

Mètodes propis

- virtual void keyPressEvent (QKeyEvent *)
- virtual void keyReleaseEvent (QKeyEvent *)
- virtual void mouseMoveEvent (QMouseEvent *)
- virtual void mousePressEvent (QMouseEvent *)
- virtual void mouseReleaseEvent (QMouseEvent *)
- virtual void wheelEvent (QWheelEvent *)

Mètodes/Atributs heredats

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget(); // dóna accés a l'escena i la càmera
- scene()->objects().size() // num objectes
- camera()->getObs() // pos de l'observador

Action plugins

Mètodes propis

- virtual void keyPressEvent (QKeyEvent *)
- virtual void keyReleaseEvent (QKeyEvent *)
- virtual void mouseMoveEvent (QMouseEvent *)
- virtual void mousePressEvent (QMouseEvent *)
- virtual void mouseReleaseEvent (QMouseEvent *)
- virtual void wheelEvent (QWheelEvent *)

Mètodes/Atributs heredats

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget(); // dóna accés a l'escena i la càmera
- scene()->objects().size() // num objectes
- camera()->getObs() // pos de l'observador

Action plugins

Mètodes propis

- virtual void keyPressEvent (QKeyEvent *)
- virtual void keyReleaseEvent (QKeyEvent *)
- virtual void mouseMoveEvent (QMouseEvent *)
- virtual void mousePressEvent (QMouseEvent *)
- virtual void mouseReleaseEvent (QMouseEvent *)
- virtual void wheelEvent (QWheelEvent *)

Mètodes/Atributs heredats

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget(); // dóna accés a l'escena i la càmera
- scene()->objects().size() // num objectes
- camera()->getObs() // pos de l'observador

Action plugins

Mètodes propis

- virtual void keyPressEvent (QKeyEvent *)
- virtual void keyReleaseEvent (QKeyEvent *)
- virtual void mouseMoveEvent (QMouseEvent *)
- virtual void mousePressEvent (QMouseEvent *)
- virtual void mouseReleaseEvent (QMouseEvent *)
- virtual void wheelEvent (QWheelEvent *)

Mètodes/Atributs heredats

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget(); // dóna accés a l'escena i la càmera
- scene()->objects().size() // num objectes
- camera()->getObs() // pos de l'observador

Action plugins

Mètodes propis

- virtual void keyPressEvent (QKeyEvent *)
- virtual void keyReleaseEvent (QKeyEvent *)
- virtual void mouseMoveEvent (QMouseEvent *)
- virtual void mousePressEvent (QMouseEvent *)
- virtual void mouseReleaseEvent (QMouseEvent *)
- virtual void wheelEvent (QWheelEvent *)

Mètodes/Atributs heredats

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget(); // dóna accés a l'escena i la càmera
- scene()->objects().size() // num objectes
- camera()->getObs() // pos de l'observador

Action plugins

Mètodes propis

- virtual void keyPressEvent (QKeyEvent *)
- virtual void keyReleaseEvent (QKeyEvent *)
- virtual void mouseMoveEvent (QMouseEvent *)
- virtual void mousePressEvent (QMouseEvent *)
- virtual void mouseReleaseEvent (QMouseEvent *)
- virtual void wheelEvent (QWheelEvent *)

Mètodes/Atributs heredats

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget(); // dóna accés a l'escena i la càmera
- scene()->objects().size() // num objectes
- camera()->getObs() // pos de l'observador

Action plugins

Mètodes propis

- virtual void keyPressEvent (QKeyEvent *)
- virtual void keyReleaseEvent (QKeyEvent *)
- virtual void mouseMoveEvent (QMouseEvent *)
- virtual void mousePressEvent (QMouseEvent *)
- virtual void mouseReleaseEvent (QMouseEvent *)
- virtual void wheelEvent (QWheelEvent *)

Mètodes/Atributs heredats

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget(); // dóna accés a l'escena i la càmera
- scene()->objects().size() // num objectes
- camera()->getObs() // pos de l'observador

Action plugins

Mètodes propis

- virtual void keyPressEvent (QKeyEvent *)
- virtual void keyReleaseEvent (QKeyEvent *)
- virtual void mouseMoveEvent (QMouseEvent *)
- virtual void mousePressEvent (QMouseEvent *)
- virtual void mouseReleaseEvent (QMouseEvent *)
- virtual void wheelEvent (QWheelEvent *)

Mètodes/Atributs heredats

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget(); // dóna accés a l'escena i la càmera
- scene()->objects().size() // num objectes
- camera()->getObs() // pos de l'observador

Flux de control

Per cada refresh:

- Si hi ha plugins registrats es crida el mètode preFrame() de cadascun.
- Si hi ha plugins registrats es crida el mètode postFrame() de cadascun.

Tractament d'esdeveniments

Es propaguen als plugins que hi hagi registrats:

```
1 ...
2 void GLWidget::mousePressEvent( QMouseEvent *e)
3 {
4     for (unsigned int i=0; i<plugins.size(); ++i)
5         qobject_cast<BasicPlugin*>
6             (plugins[i]->instance())->mousePressEvent(e);
7 }
8 ...
```



Geometry shaders (GLSL 3.30 core)

C. Andújar (*)

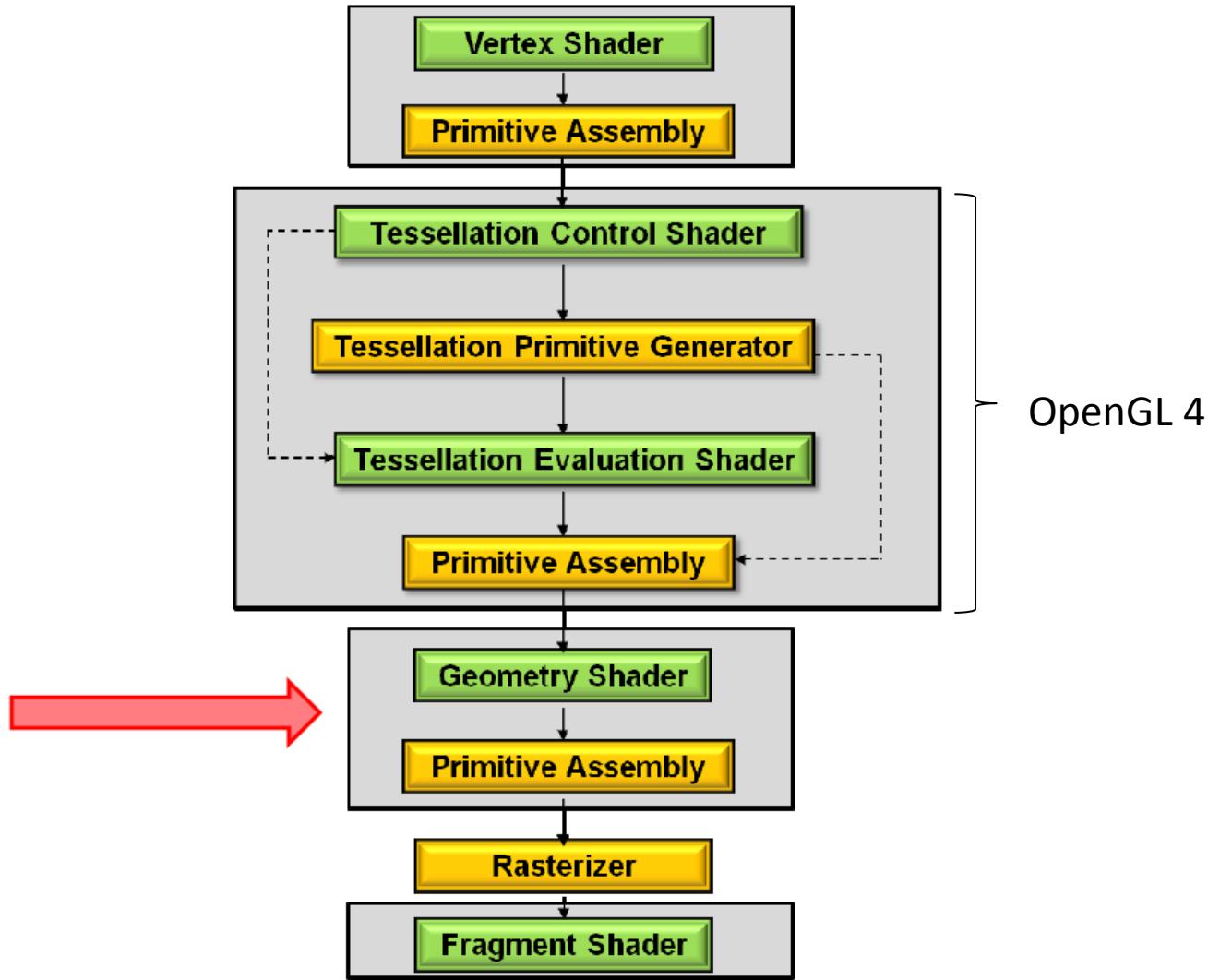
Nov 2015

(*) Basades en el material de Mike Bailey

Introducció

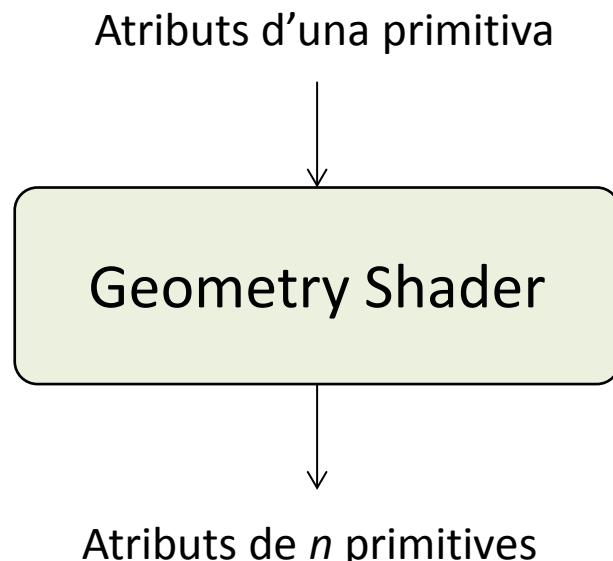
- Els GS processen **primitives** (punts, línies, triangles)
- Ofereixen la possibilitat de **crear noves primitives** i de canviar-ne la **topologia** (exemple: punt → triangle)
- Disponibles a partir d'OpenGL 2.1, GLSL 1.20.

Situació al pipeline

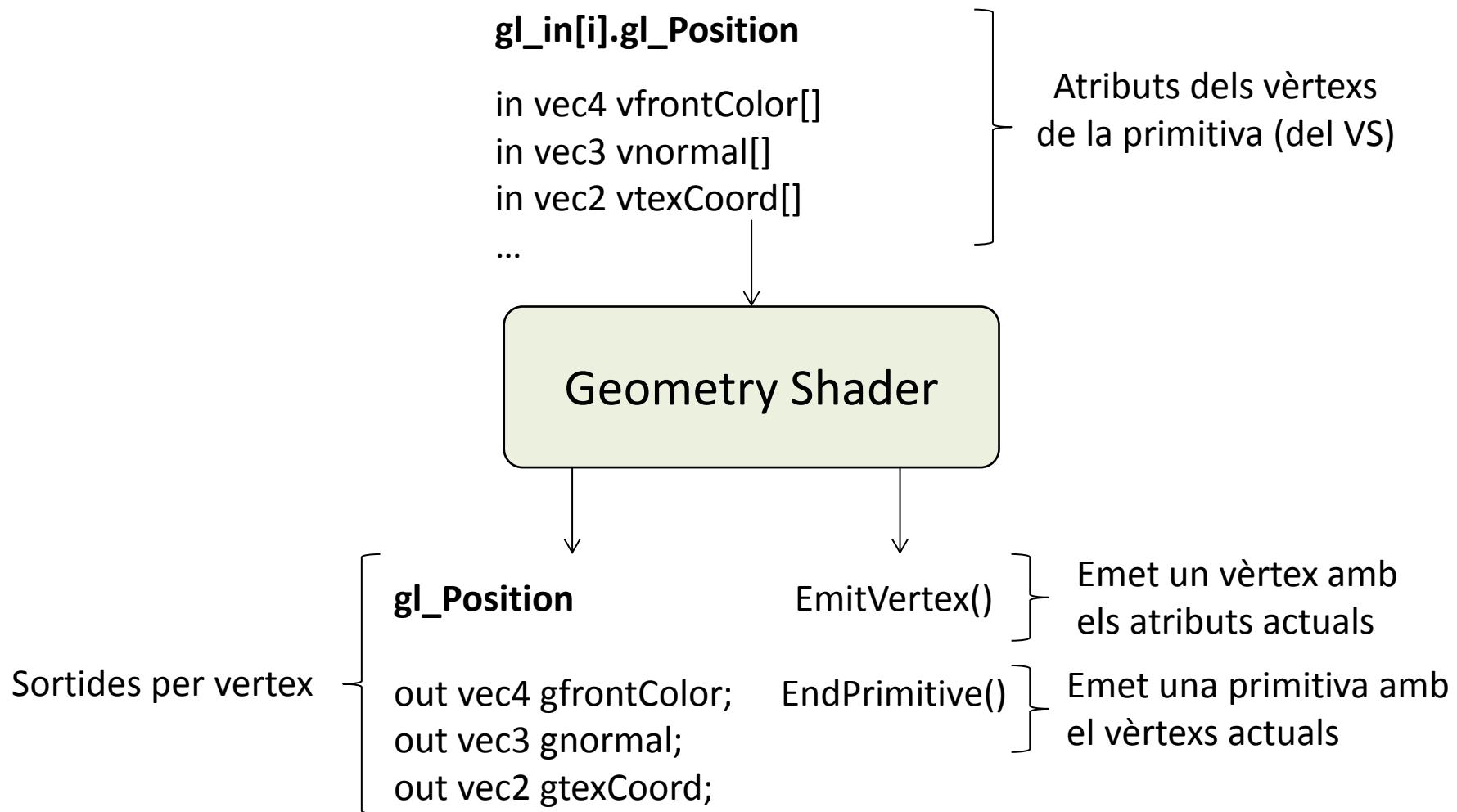


ENTORN D'EXECUCIÓ DEL GS

Entrades i sortides



Entrades i sortides



Exemple minimalista GS

```
#version 330 core
layout(triangles) in;
layout(triangle_strip, max_vertices = 36) out;

void main(void){
    for( int i = 0 ; i < 3 ; i++ )
    {
        gl_Position = gl_in[i].gl_Position;
        EmitVertex();
    }
    EndPrimitive();
}
```

Exemple



Shaders per defecte: VS

```
// default.vert  
#version 330 core  
layout (location = 0) in vec3 vertex;  
layout (location = 1) in vec3 normal;  
layout (location = 2) in vec3 color;  
layout (location = 3) in vec2 texCoord;  
out vec4 frontColor;  
  
...  
  
void main(){  
    vec3 N = normalize(normalMatrix * normal);  
    frontColor = vec4(color,1.0) * N.z;  
    gl_Position = modelViewProjectionMatrix *  
    vec4(vertex.xyz, 1.0);  
}
```

```
// default.vert  
#version 330 core  
layout (location = 0) in vec3 vertex;  
layout (location = 1) in vec3 normal;  
layout (location = 2) in vec3 color;  
layout (location = 3) in vec2 texCoord;  
out vec4 vfrontColor;  
  
...  
  
void main(){  
    vec3 N = normalize(normalMatrix * normal);  
    vfrontColor = vec4(color,1.0) * N.z;  
    gl_Position = modelViewProjectionMatrix *  
    vec4(vertex.xyz, 1.0);  
}
```

Shaders per defecte: GS

```
// default.geom
```

```
// default.geom
#version 330 core
layout(triangles) in;
layout(triangle_strip, max_vertices = 36) out;
in vec4 vfrontColor[];
out vec4 gfrontColor;
void main( void ){
    for( int i = 0 ; i < 3 ; i++ )
    {
        gfrontColor = vfrontColor[i];
        gl_Position = gl_in[i].gl_Position;
        EmitVertex();
    }
    EndPrimitive();
}
```

Shaders per defecte: FS

```
// default.frag  
#version 330 core  
in vec4 frontColor;  
out vec4 fragColor;  
  
void main()  
{  
    fragColor = frontColor;  
}
```

```
// default.frag  
#version 330 core  
in vec4 gfrontColor;  
out vec4 fragColor;  
  
void main()  
{  
    fragColor = gfrontColor;  
}
```

Il·luminació per fragment amb GS



Observacions

- Si useu GS, els **out** del VS només arribaràn al FS si el GS els hi ha passat.
- No hi ha cap **BeginPrimitive()**; és implícit
- Es recomana cridar **EndPrimitive()** al final de cada primitiva (tot i que la darrera crida és implícita).

TIPUS DE PRIMITIVES

Primitives

Your application generates these

Points, Lines, Line Strip, Line Loop,,Lines with Adjacency, Line Strip with Adjacency, Triangles, Triangle Strip, Triangle Fan, Triangles with Adjacency, Triangle Strip with Adjacency

The driver translates those and feeds these one-at-a-time into the Geometry Shader

Point, Line, Line with Adjacency, Triangle, Triangle with Adjacency

Geometry Shader

Primitive Assembly

The Geometry Shader generates (almost) as many of these as it wants

Points, LineStrips, TriangleStrips

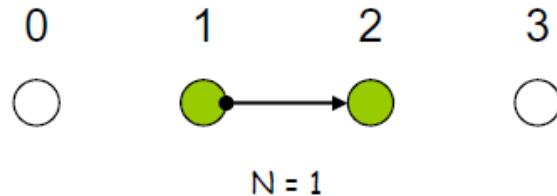
Primitives que envia l'aplicació

Primitives (`glBegin...`):

- **GL_POINT**
- **GL_TRIANGLES**
- ...
- **GL_LINES_ADJACENCY**
- **GL_LINE_STRIP_ADJACENCY**
- **GL_TRIANGLES_ADJACENCY**
- **GL_TRIANGLE_STRIP_ADJACENCY**

Adjacències - línies

Lines with Adjacency



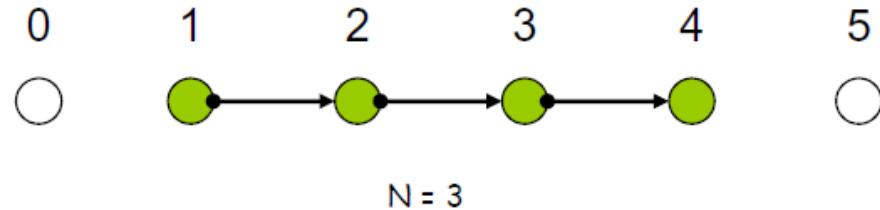
$4N$ vertices are given.

(where N is the number of line segments to draw).

A line segment is drawn between #1 and #2.

Vertices #0 and #3 are there to provide adjacency information.

Line Strip with Adjacency



$N+3$ vertices are given

(where N is the number of line segments to draw).

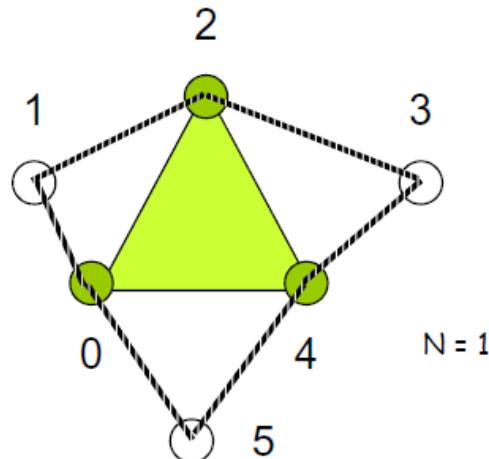
A line segment is drawn between #1 and #2, #2 and #3, #3 and #4, ..., # N and # $N+1$.

Vertices #0 and # $N+2$ are there to provide adjacency information.

Adjacències - triangles

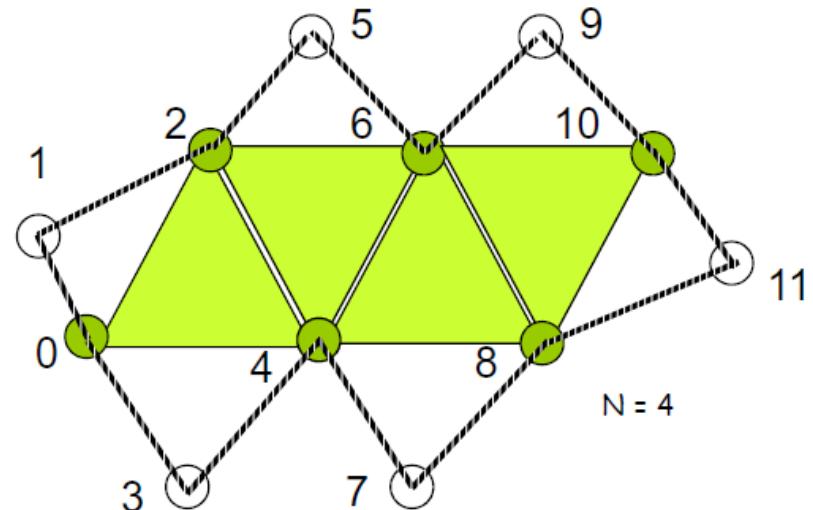
Triangles with Adjacency

6N vertices are given
(where N is the number of triangles to draw).
Points 0, 2, and 4 define the triangle.
Points 1, 3, and 5 tell where adjacent triangles are.



Triangle Strip with Adjacency

4+2N vertices are given
(where N is the number of triangles to draw).
Points 0, 2, 4, 6, 8, 10, ... define the triangles.
Points 1, 3, 5, 7, 9, 11, ... tell where adjacent triangles are.



Número de vèrtexs

Número de vèrtexs que rep el GS:

- **GL_POINTS** → 1
- **GL_LINES** → 2
- **GL_TRIANGLES** → 3
- **GL_LINES_ADJACENCY** → 4
- **GL_TRIANGLES_ADJACENCY** → 6

Geometry Language

```
in gl_PerVertex {  
    vec4 gl_Position;  
    float gl_PointSize;  
    float gl_ClipDistance[];  
} gl_in[];  
  
in int gl_PrimitiveIDIn;
```

```
out gl_PerVertex {  
    vec4 gl_Position;  
    float gl_PointSize;  
    float gl_ClipDistance[];  
};  
  
out int gl_PrimitiveID;  
out int gl_Layer;
```

Primitives que pot crear un GS

Un GS només pot generar:

- Punts (GL_POINTS)
- Segments (GL_LINE_STRIP)
- Triangles (GL_TRIANGLE_STRIP)