

# Laboratori Gràfics

Shaders - Sessió 3

Sistemes de Coordenades, Transformacions  
geomètriques, animacions.

# Sistemes de coordenades i matrius

Object space



*Modeling transform*

World space



*Viewing transform*

Eye space



*Projection transform*

Clip space



*Perspective division*

Normalized Device space



*Viewport transform &  
Depth transform*

Window space

**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;

# Transformacions bàsiques

Object space

**Modeling transform**

World space

**Viewing transform**

Eye space

**Projection transform**

Clip space

**Perspective division**

Normalized Device space

**Viewport transform &  
Depth transform**

Window 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( $a, x, y, z$ )

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

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

# Transformacions bàsiques

Object space

*Modeling transform*

World space

*Viewing transform*

Eye space

*Projection transform*

Clip space

*Perspective division*

Normalized Device space

*Viewport transform &  
Depth transform*

Window space

**Modeling transforms**

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

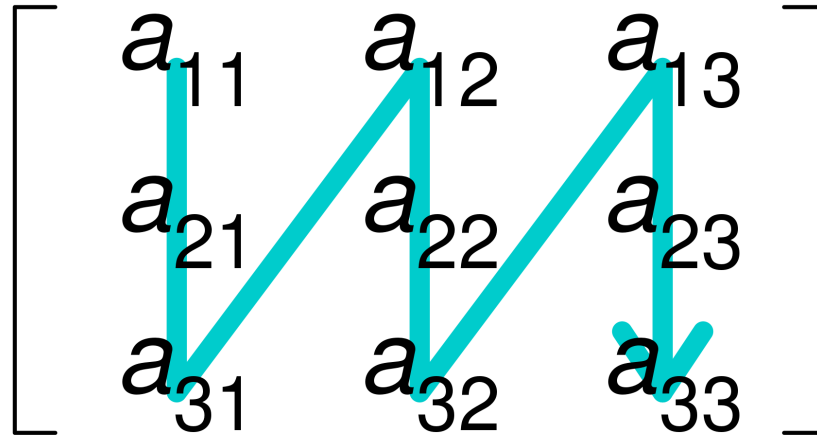
$$\text{glRotate}^*(\alpha, 0, 1, 0): \begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

# Matrius en GLSL

```
mat3 m = mat3(vec3(1,0,0), vec3(0,1,0), vec3(0,0,1));
```

// els tres vectors són les **columnes** de la matriu



A 3x3 matrix is shown within large square brackets. The elements are labeled as  $a_{ij}$  where  $i$  is the row index and  $j$  is the column index. The elements are arranged as follows:

$a_{11}$	$a_{12}$	$a_{13}$
$a_{21}$	$a_{22}$	$a_{23}$
$a_{31}$	$a_{32}$	$a_{33}$

Cyan arrows illustrate the column-major storage order: a vertical arrow points down from  $a_{11}$  to  $a_{31}$ , a diagonal arrow points from  $a_{31}$  to  $a_{12}$ , a vertical arrow points down from  $a_{12}$  to  $a_{32}$ , a diagonal arrow points from  $a_{32}$  to  $a_{13}$ , and a vertical arrow points down from  $a_{13}$  to  $a_{33}$ .

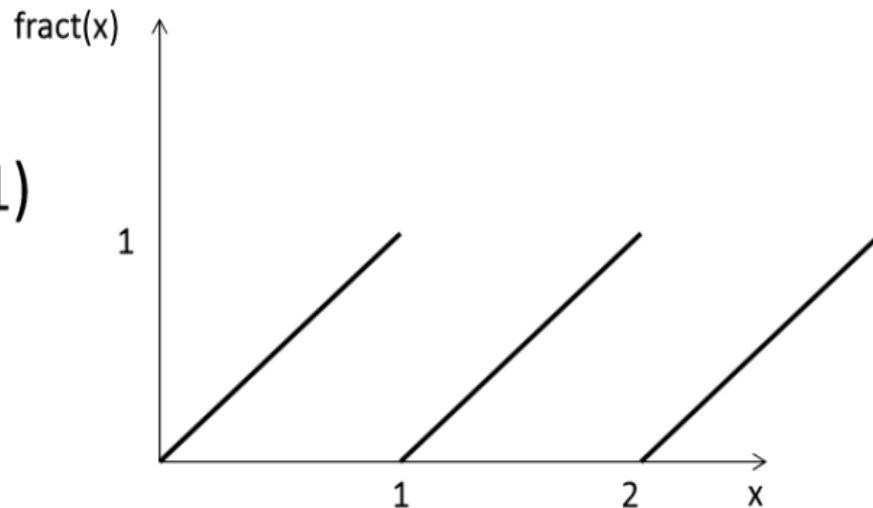
Funcions GLSL

# Funcions GLSL – **fract(x)**

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

$$x - \text{floor}(x)$$

- Domini:  $\mathbb{R}^n$
- Recorregut:  $[0, 1)$
- Període: 1

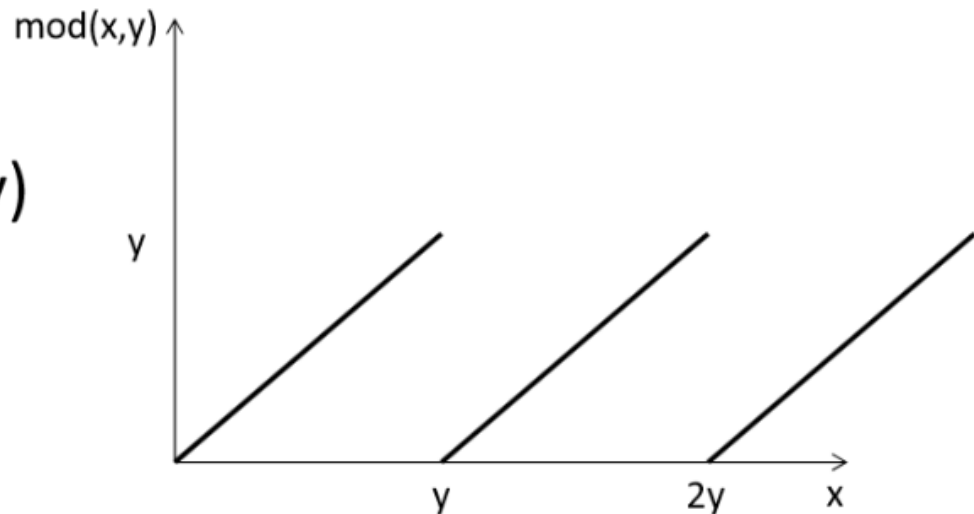


# Funcions GLSL – $\text{mod}(x,y)$

Retorna  **$x$  mòdul  $y$** , calculat com

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

- Domini:  $\mathbb{R}^n$
- Recorregut:  $[0, y)$
- Període:  $y$





# 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 sinusoidals de la forma:

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

$A$  = amplitud

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

$t$  = temps (en segons)

$\Theta$  = fase; si per exemple  $\Theta = \{0, \pi/2, 3\pi/2\}$ , llavors per  $t=0$  la sinusoidal serà  $\{0, A, -A\}$

# Animacions als shaders

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

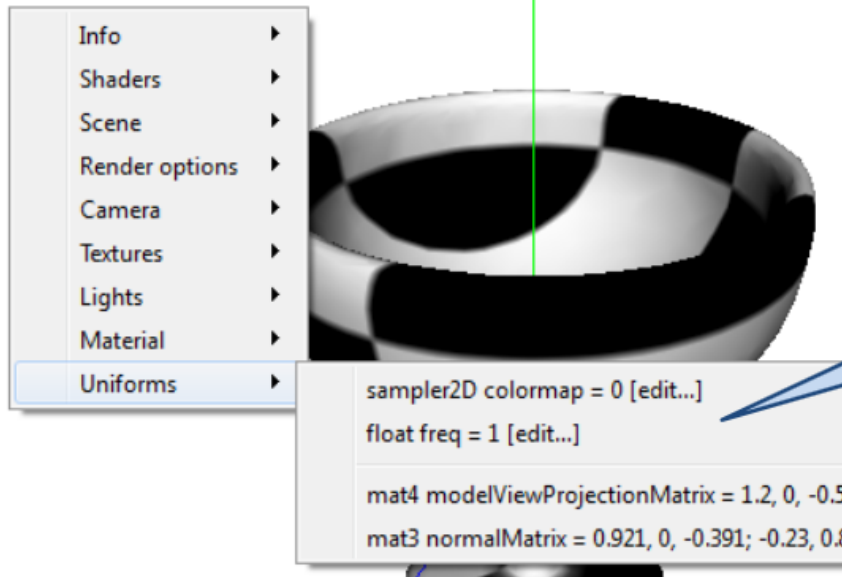
Temps (segons) des de la darrera compilació.

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

# 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)

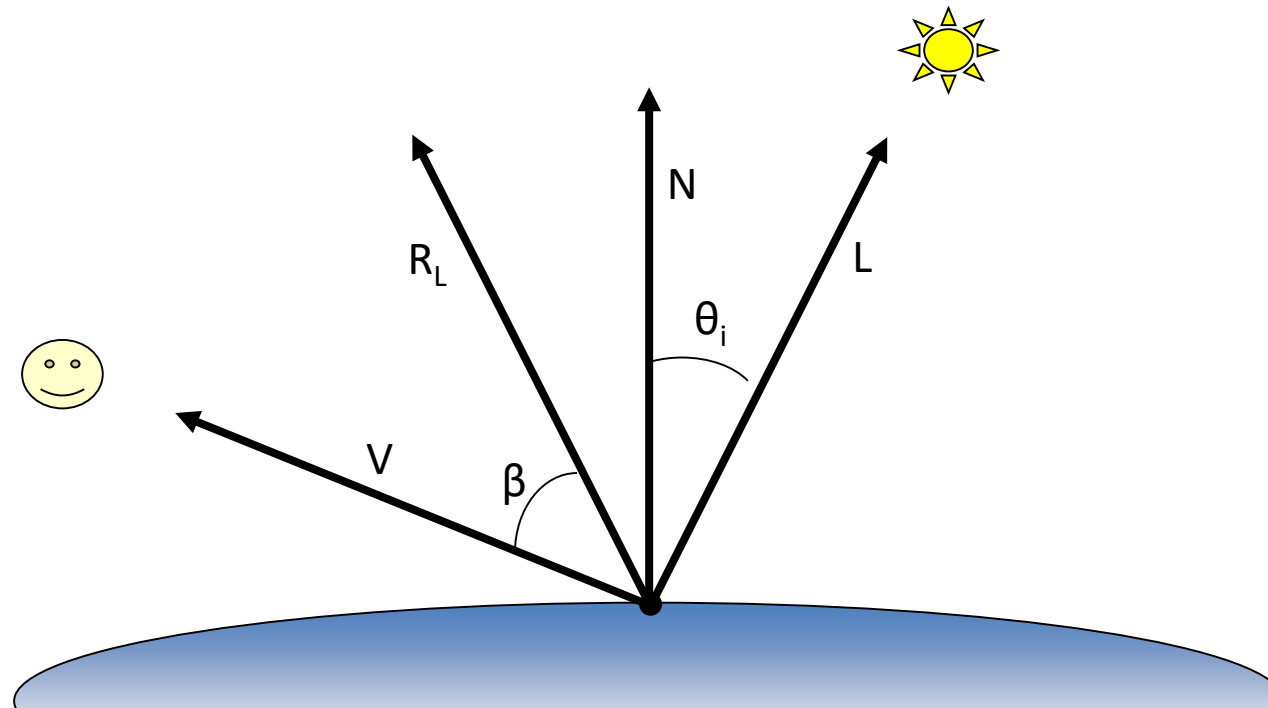
Uniforms definits pel viewer (el menu no en permet l'edició)

# Guia exercicis

Il·luminació

2020

# Notació



# Model de Phong

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

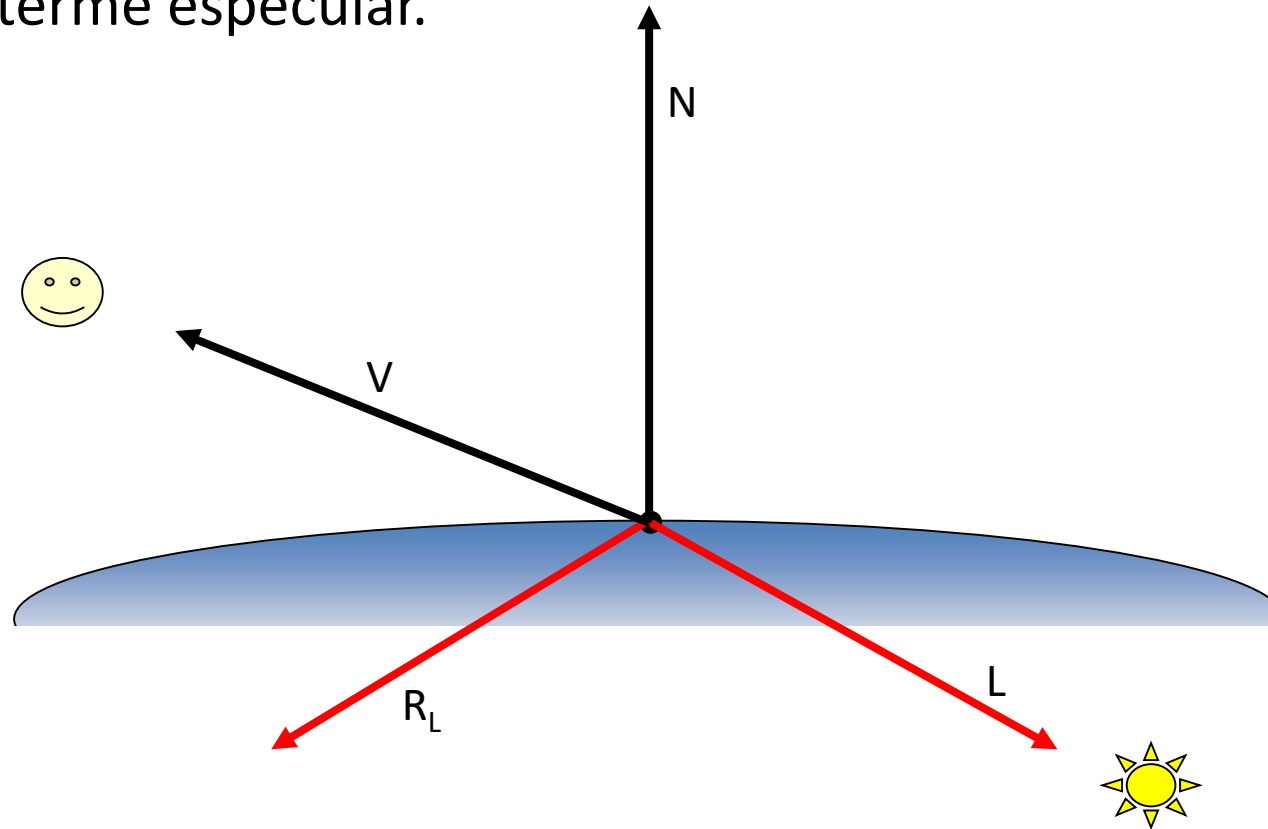
- $K_*$  = material

- $I_*$  = llum

# Notació

Si  $N \cdot L < 0$ :

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

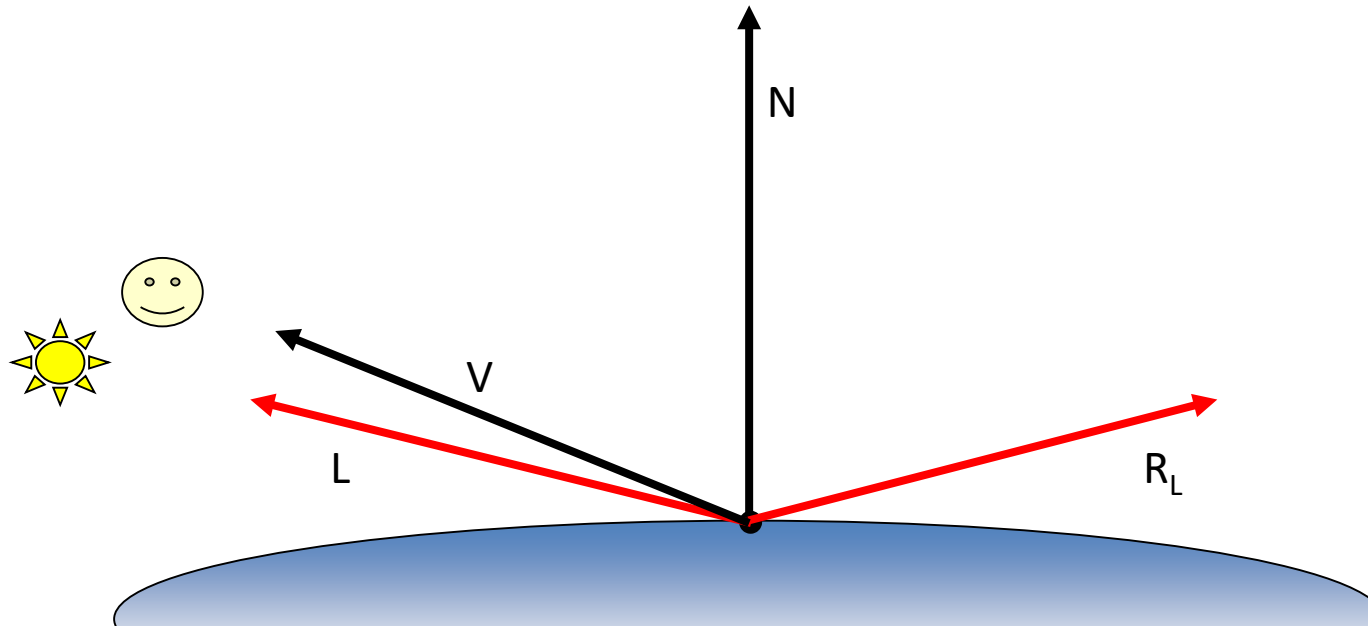




# Notació

Si  $R \cdot V < 0$ :

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



# Exemple senzill

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

- $K_*$  = material

- $I_*$  = llum

# Quan normalitzar

- Els vectors (N, L, R, V) que apareixen a les equacions 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 linealment (ex. `out vec3 N`)
- On normalitzar? Immediatament abans de fer els càlculs que assumeixen que el vector és unitari: al VS si il·lum per vèrtex, al FS si il·lum per fragment.

# Laboratori de Gràfics

Sessió 5

# Interpolació per fragment

- Tot el que s'interpola 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

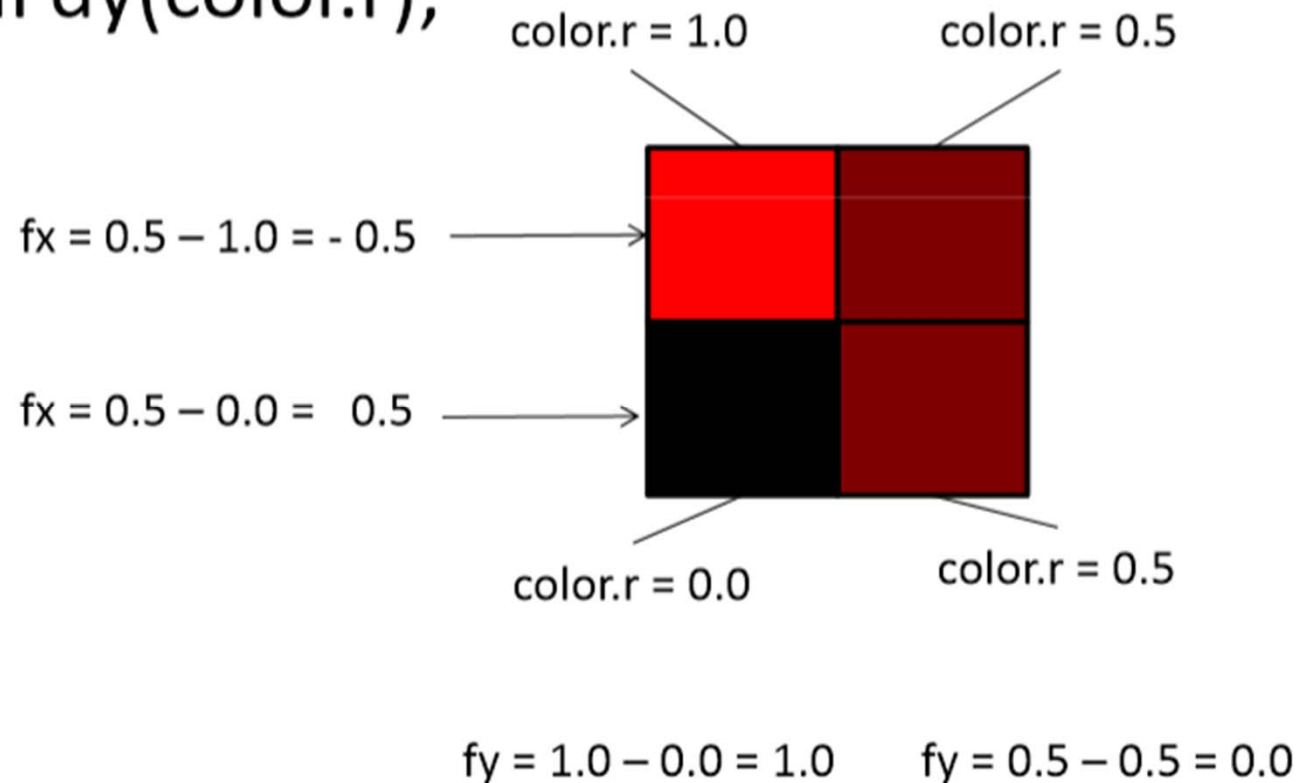
- En algunes versions de GLSL, és pot eliminar aquest offset redeclarant `gl_FragCoord`

**layout(pixel\_center\_integer) in vec4** gl\_FragCoord;

# dFdx, dFdy - exemple

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

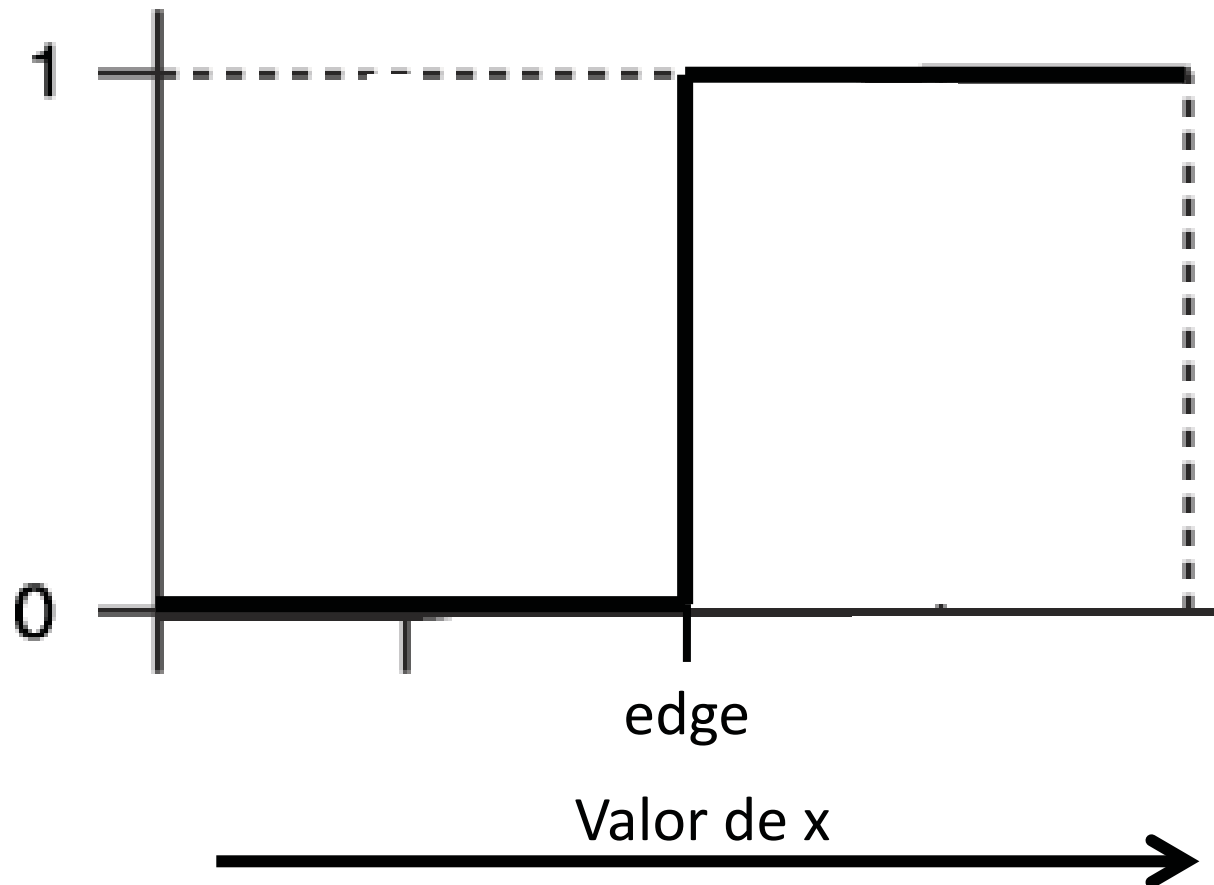
```
float fy = dFdy(color.r);
```



# Functions step, smoothstep

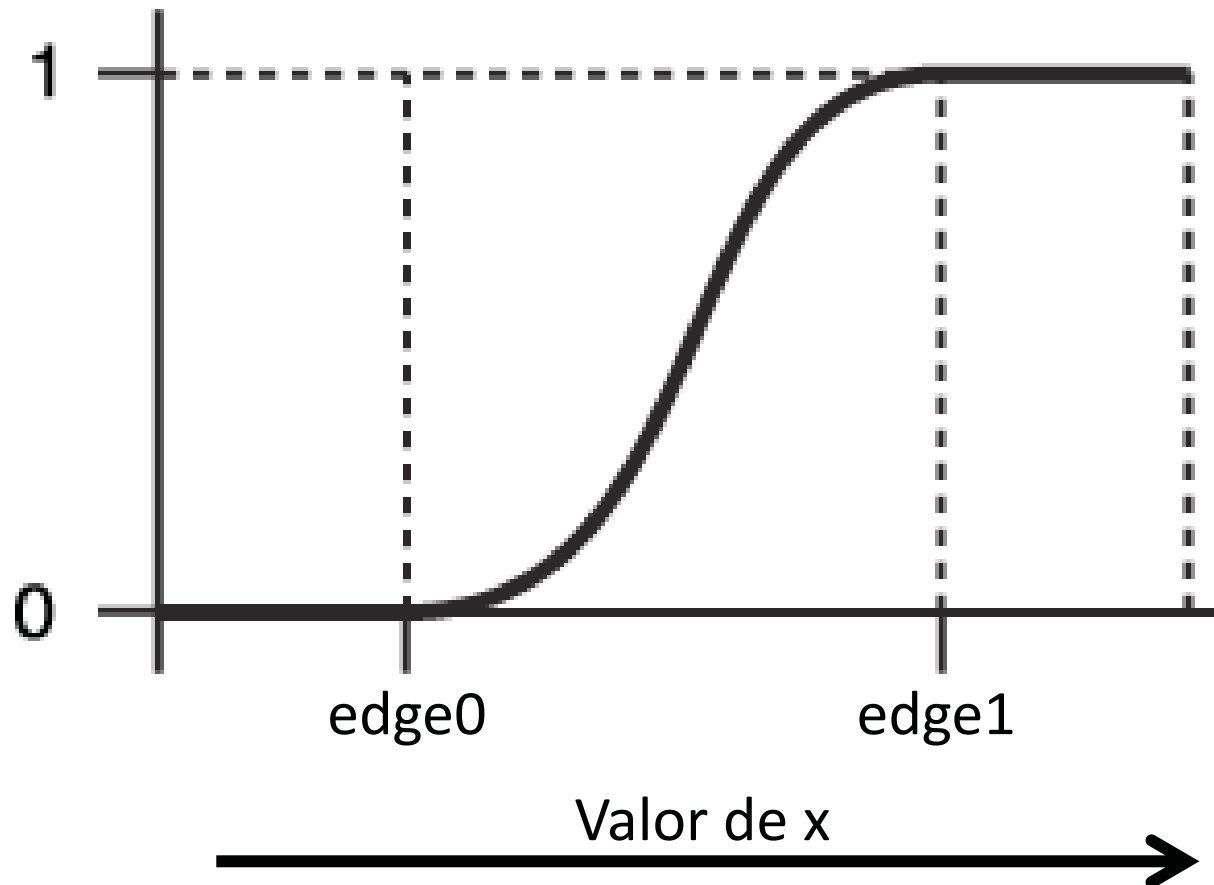
**float step(float edge, float x)**

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



# Functions step, smoothstep

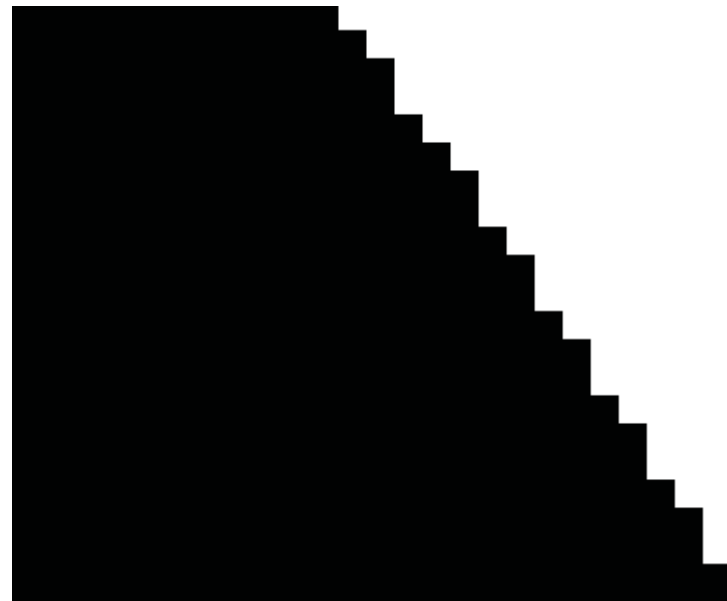
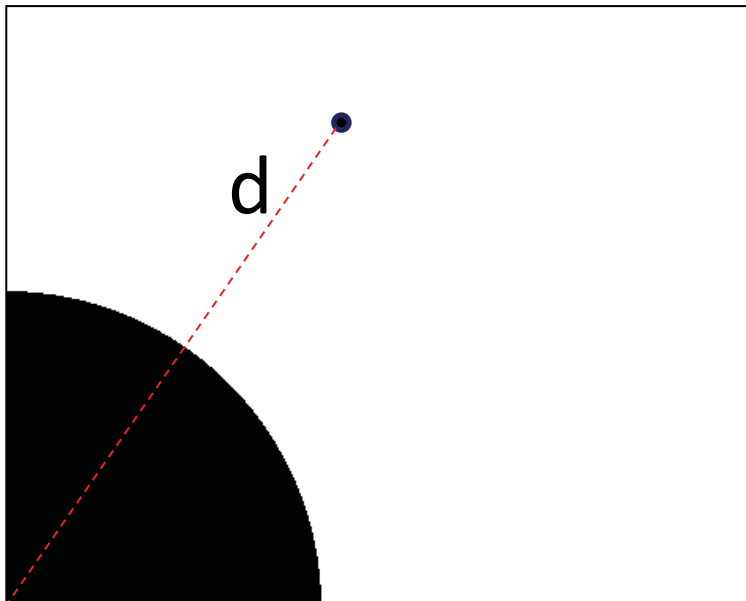
**float smoothstep(float edge0, float edge1, float x)**





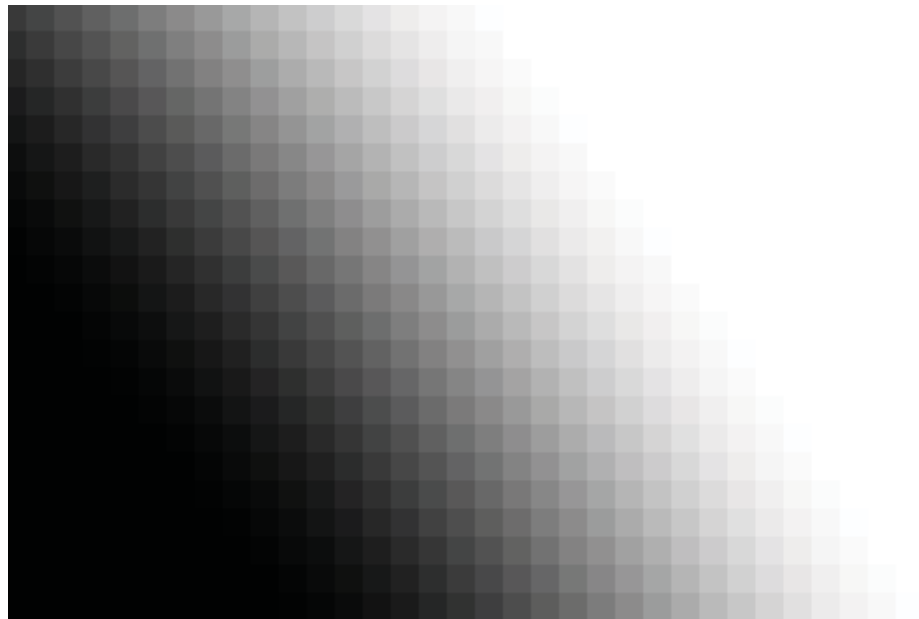
# Exemple - step

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



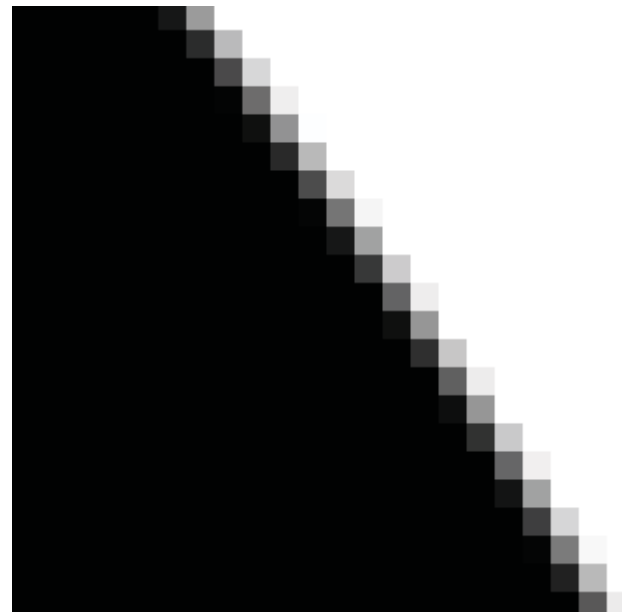
# Exemple - step

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



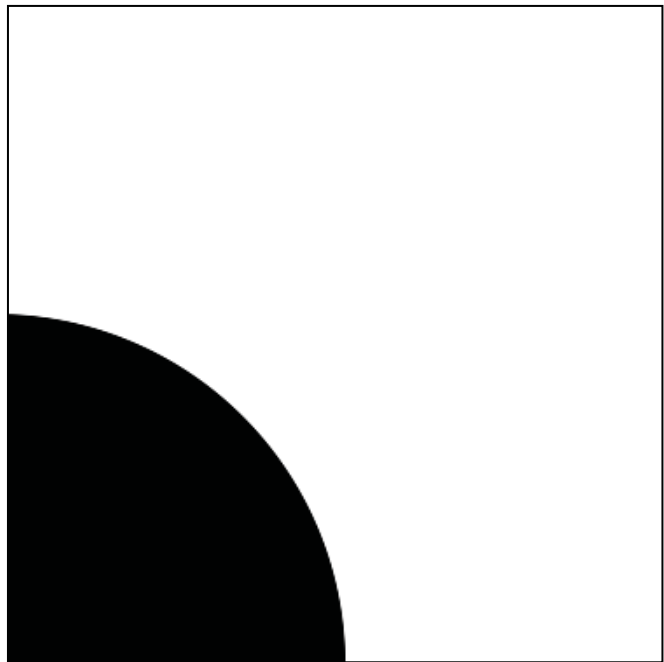
# Example - smoothstep

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



# Example - smoothstep

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



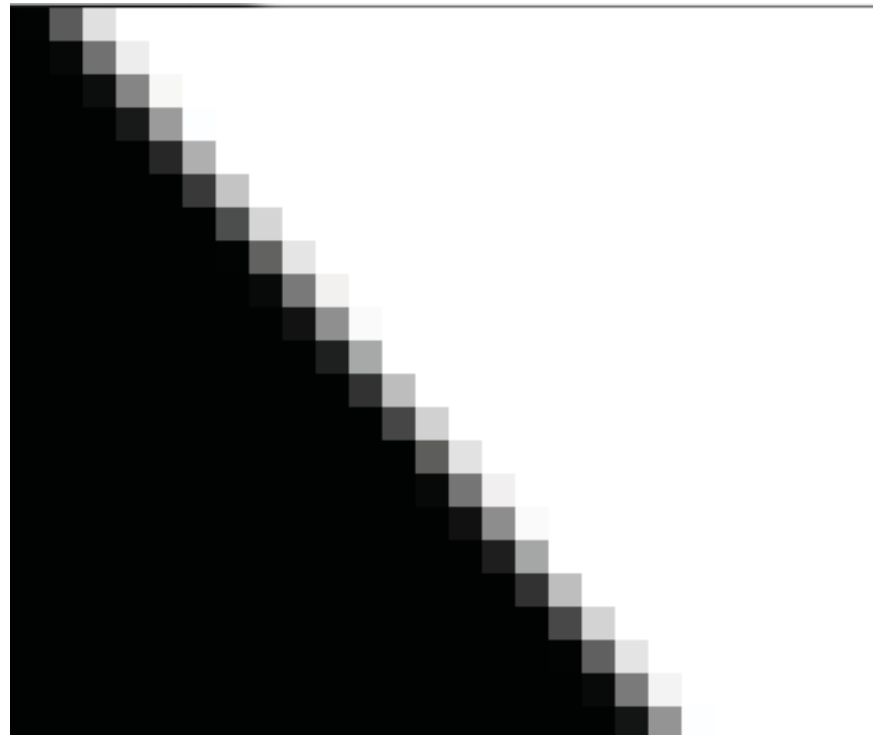
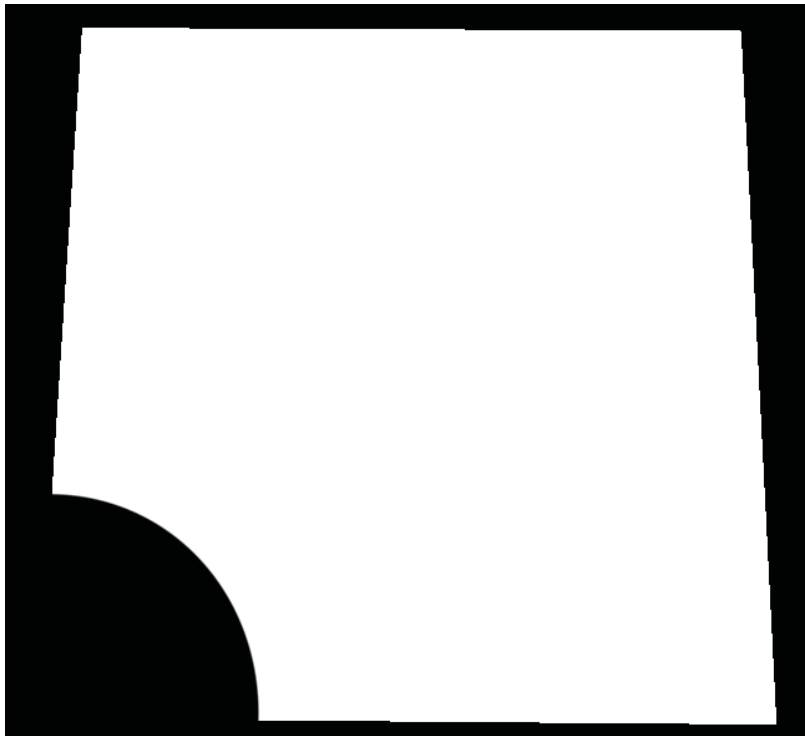
## Exemple 2 - smoothstep

```
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

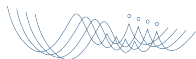
## Laboratori de Gràfics, part 2.

À. Vinacua, C. Andújar i professors de Gràfics

5 de novembre de 2019



# Segona part del laboratori

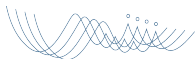


## Segona part del laboratori

### 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, . . .

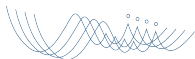


## Segona part del laboratori

### 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**, . . .

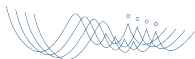


# Eines

C++

Qt5 (però no caldran gaires coneixements específics)

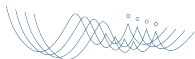
OpenGL (Core) + GLSL



## 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).



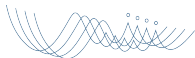
# Avaluació

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**



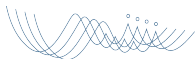
# Avaluació

El control final de laboratori inclourà:

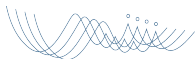
Exercicis de shaders pel visualitzador (fins ara heu fet servir un plugin específic: *shaderloader*).

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 directoris

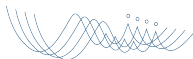




## Codi de partida del Visualitzador

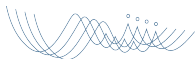
Viewer/ ← Directori arrel de l'aplicació

- all.pro
- GLarena
- GLarenaPL
- GLarenaSL
- plugins/
- viewer/



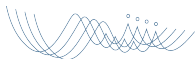
## Codi de partida del Visualitzador

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



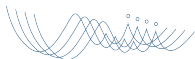
## Codi de partida del Visualitzador

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



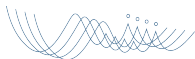
## Codi de partida del Visualitzador

```
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

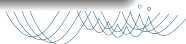
```
Viewer/ ← Directori arrel de l'aplicació
├── all.pro ← arxiu pel qmake recursiu
├── GLarena
├── GLarenaPL ← scripts per a engegar
               l'aplicació
├── GLarenaSL
├── plugins/ ← fonts dels plugins
└── viewer/ ← fonts del nucli del Viewer
```



## Codi de partida del Visualitzador

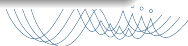
viewer/ ←D'aquí no heu de canviar res...

- ├ bin/
- ├ app/
  - ├ app.pro
  - └ main.cpp
- ├ core/
  - ├ core.pro
  - ├ include/
  - └ src/
- ├ glwidget/
  - ├ glwidget.pro
  - ├ include/
  - └ src/
- └ interfaces/
  - └ plugin.h



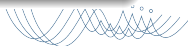
## Codi de partida del Visualitzador

```
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

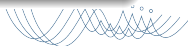
```
plugins/  
├── bin/  
├── common.pro  
├── plugins.pro ← Cal editar-lo per afegir nous  
                plugins 'permanentment'  
  
├── alphablending/ ← Un directori per cada  
                   plugin  
    ├── alphablending.pro  
    ├── alphablending.h  
    ├── alphablending.cpp  
├── navigate-default/  
    └── ...  
└── ...
```





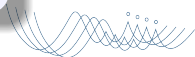
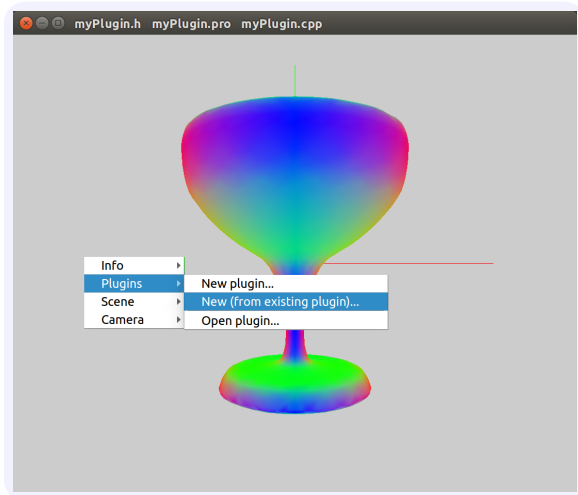
## Codi de partida del Visualitzador

```
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/
│   └── ...
└── ...
```



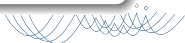
# pluginLoader

Un plugin similar a shaderLoader, per a programar plugins

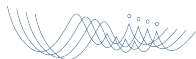


## Algunes restriccions del pluginLoader

- 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:

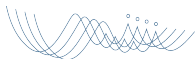
VIMAGE defineix l'executable a fer servir per mostrar imatges

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

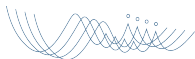
VMODELS el directori on trobar models

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`

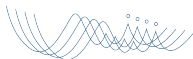
- Editar el fitxer `my-effect.h`

- Editar el fitxer `my-effect.cpp`

Afegiu una línia a `plugins/plugins.pro`

```
SUBDIRS += my-effect
```

`[qmake +] make` (des del directori `viewer`)



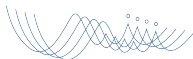


## Amb pluginLoader...

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

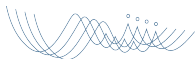
No cal preocupar-se de cap pas dels mencionats anteriorment, però convé ser conscient d'algunes particularitats:

- 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.



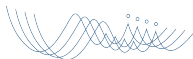
# Tipus de plugins

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



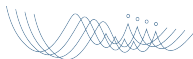
## (Alguns) Mètodes virtuals de la classe base dels plugins:

```
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 pinti 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

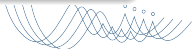
Executen accions arbitràries en resposta a events (mouse, teclat).

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...



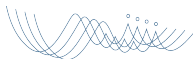
## Plugins per defecte

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

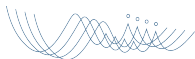
**render-default**: un *render plugin* bàsic; sols esborra els buffers, crida al `drawPlugin` si està carregat, i afegeix els eixos coordenats.

**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



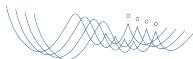
## Effect plugins

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();
```



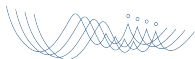


## Exemples d'accés als objectes de l'aplicació

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

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

```
glwidget()->defaultProgram()
```

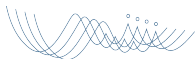


## Exemples d'accés als objectes de l'aplicació

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

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

```
glwidget()->defaultProgram()
```

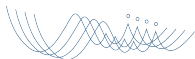


## Exemples d'accés als objectes de l'aplicació

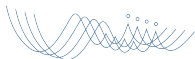
```
scene()->objects().size() // num objectes
```

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

```
glwidget()->defaultProgram()
```



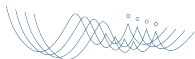
Exemples d'effect plugins: 1/3



# alphablending

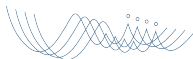
alphablending.pro

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



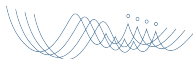
## alphablending.h

```
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
```

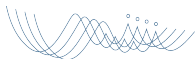


## alphablending.cpp

```
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

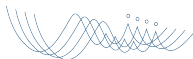




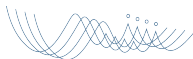
## effect-crt

effect-crt.pro

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

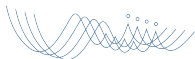


```
1  #ifndef _EFFECTCRT_H
2  #define _EFFECTCRT_H
3  #include "plugin.h"
4  #include <QOpenGLShader>
5  #include <QOpenGLShaderProgram>
6  class EffectCRT : public QObject, public Plugin
7  {
8      Q_OBJECT
9      Q_PLUGIN_METADATA(IID "Plugin")
10     Q_INTERFACES(Plugin)
11 public:
12     void onPluginLoad();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

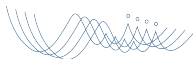


```
1  #include "effectcrt.h"
2
3  void EffectCRT::onPluginLoad() {
4      glwidget()->makeCurrent(); // !!!
5      QString vs_src =
6          "#version 330 core\n"
7          "uniform mat4 modelViewProjectionMatrix;"
8          "in vec3 vertex;"
9          "in vec3 color;"
10         "out vec4 col;"
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;
```

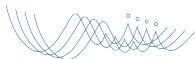
```
19 QString fs_src =
20     "#version 330 core\n"
21     "out vec4 fragColor;"
22     "in vec4 col;"
23     "uniform int n;"
24     "void main() {"
25     "    if (mod((gl_FragCoord.y-0.5), float(n)) > 0.0) dis
26     "    fragColor=col;"
27     "}";
28 fs = new QOpenGLShader(QOpenGLShader::Fragment,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



```
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 }
```



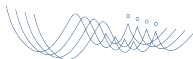
Exemples d'effect plugins: 3/3



# showHelpNg

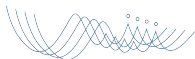
showHelpNg.pro

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



## showHelpNg.h

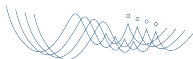
```
1  #ifndef _SHOWHELPNG_H
2  #define _SHOWHELPNG_H
3
4  #include "plugin.h"
5  #include <QPainter>
6
7  class ShowHelpNg : public QObject, Plugin
8  {
9      Q_OBJECT
10     Q_PLUGIN_METADATA(IID "Plugin")
11     Q_INTERFACES(Plugin)
12
13 public:
14     void postFrame() Q_DECL_OVERRIDE;
15 private: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



## 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::paint()` crida:

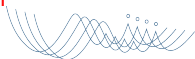
- `bind()` dels shaders per defecte

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

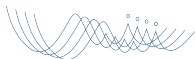
- `preFrame()` de tots els plugins

- `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

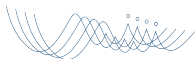
**object:** objecte (inclou codi per a carregar .obj)

**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

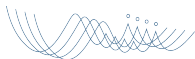
**object:** objecte (inclou codi per a carregar .obj)

**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

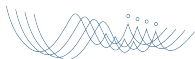
**object:** objecte (inclou codi per a carregar .obj)

**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.

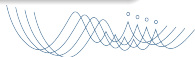
**vertex:** Model de vèrtex usat a les demés classes.



# Vector, Punt

## Vector

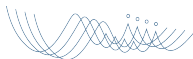
	<b>Vector</b> ( qreal xpos, qreal ypos, qreal zpos )
qreal	<b>length</b> () const
void	<b>normalize</b> ()
Point	<b>normalized</b> () const
void	<b>setX</b> ( qreal x )
void	<b>setY</b> ( qreal y )
void	<b>setZ</b> ( qreal z )
qreal	<b>x</b> () const
qreal	<b>y</b> () const
qreal	<b>z</b> () const
Vector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

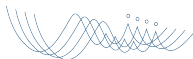




# Vector, Point

## 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 subtraction (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 capsa
9
10        void render();               // dibuixa en filferros
11        Point center() const; // centre de la capsa
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

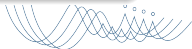
```
1  class Object {
2  public:
3      ...
4      Box boundingBox() const;
5      const vector<Face>& faces() const;
6      const vector<Vertex>& vertices() const;
7      void computeNormals();           // normals *per-cara*
8      void computeBoundingBox();
9      void applyGT(const QMatrix4x4& mat);
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

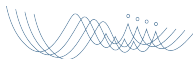
```
1 class Face
2 {
3 public:
4     ...
5     int numVertices() const;
6     int vertexIndex(int i) const;
7     Vector normal() const;
8     void addVertexIndex(int i);
9     void computeNormal(const vector<Vertex> &);
10 private:
11     Vector pnormal;
12     vector<int> pvertices; // índexs dels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



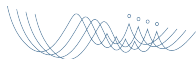
# Vertex

Simplement les coordenades d'un punt

```
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 };
```



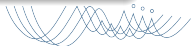
# APIs per treballar amb shaders



## 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();
```





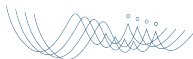
# Alguns mètodes de QOpenGLShaderProgram

## 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

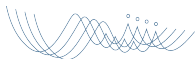
```
1 bool isLinked() const;  
2 QString log() const;  
3 void setGeometryOutputType(GLenum outputType);
```



# 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 immediat

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

# Mode immediat

Client side

Vertices

x	y	z	N <sub>x</sub>	N <sub>y</sub>	N <sub>z</sub>
-1.0	-2.0	5.0	1.0	0.0	0.0
...	...	...	...	...	...
...	...	...	...	...	...

Faces

Normal	ind	ind	ind
1, 0, 0	0	4	2
...	...	...	...
...	...	...	...

T crides a glBegin()  
3\*T crides glVertex()  
3\*T crides glNormal()

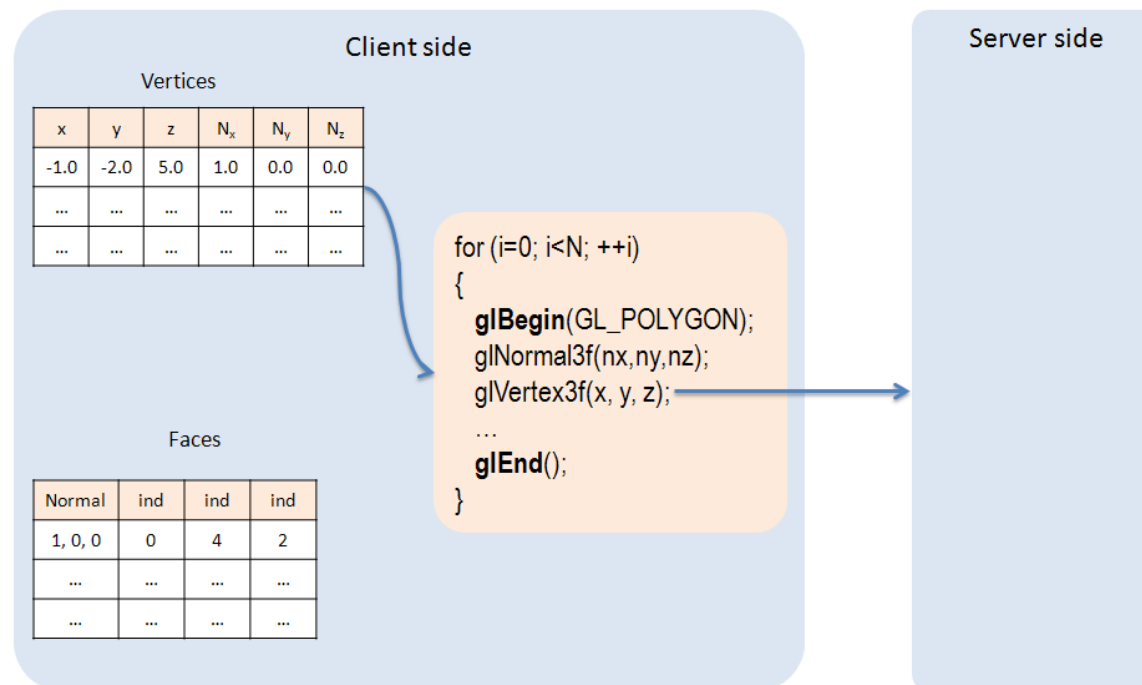
```
for (i=0; i<N; ++i)
{
    glBegin(GL_TRIANGLES);
    glNormal3f(nx,ny,nz);
    glVertex3f(x, y, z);
    ...
    glEnd();
}
```

**3\*2\*3\*T floats**  
(3 vèrtexs/triangle,  
2 atributs/vèrtex,  
3 float/attrib)

Server side

# Mode immediat

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



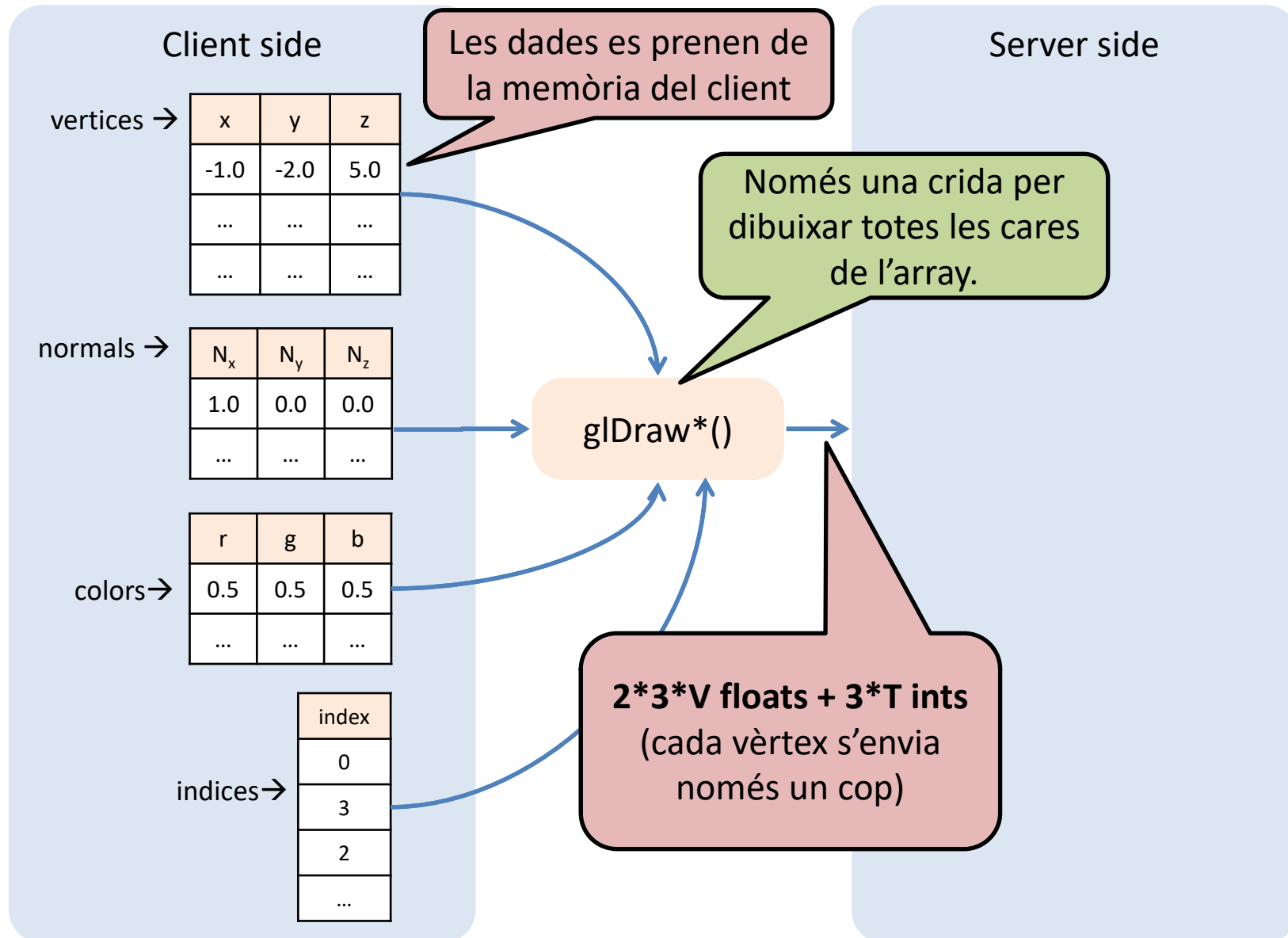
# Vertex Arrays

Objectius:

- Reduir crides a OpenGL
- Enviar un cop cada vèrtex



# Vertex Arrays



# Vertex Arrays

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

❶

❷

❸

❹

- ❶ És la primitiva: `GL_TRIANGLES`, `GL_QUADS` ...
- ❷ És el número d'índexos a l'array (ex. 12 triangles  $\rightarrow 12 \cdot 3 = 36$ )
- ❸ És el tipus dels índexs (normalment `GL_UNSIGNED_INT`)
- ❹ És l'apuntador a l'array amb els índexs (que haurem definit previament)

Quins atributs (normal, color, coords textura...) s'usaran?  
Com s'especifiquen els apuntadors a aquests atributs?

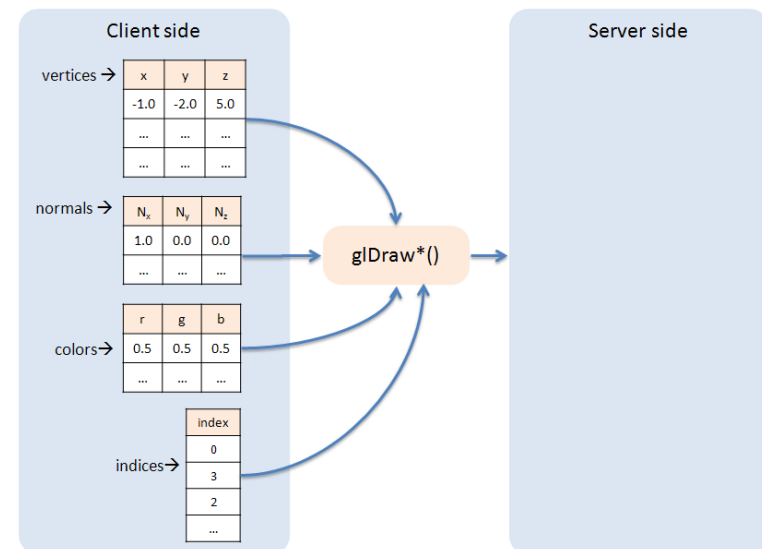
# Vertex Arrays

```
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 convertira 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

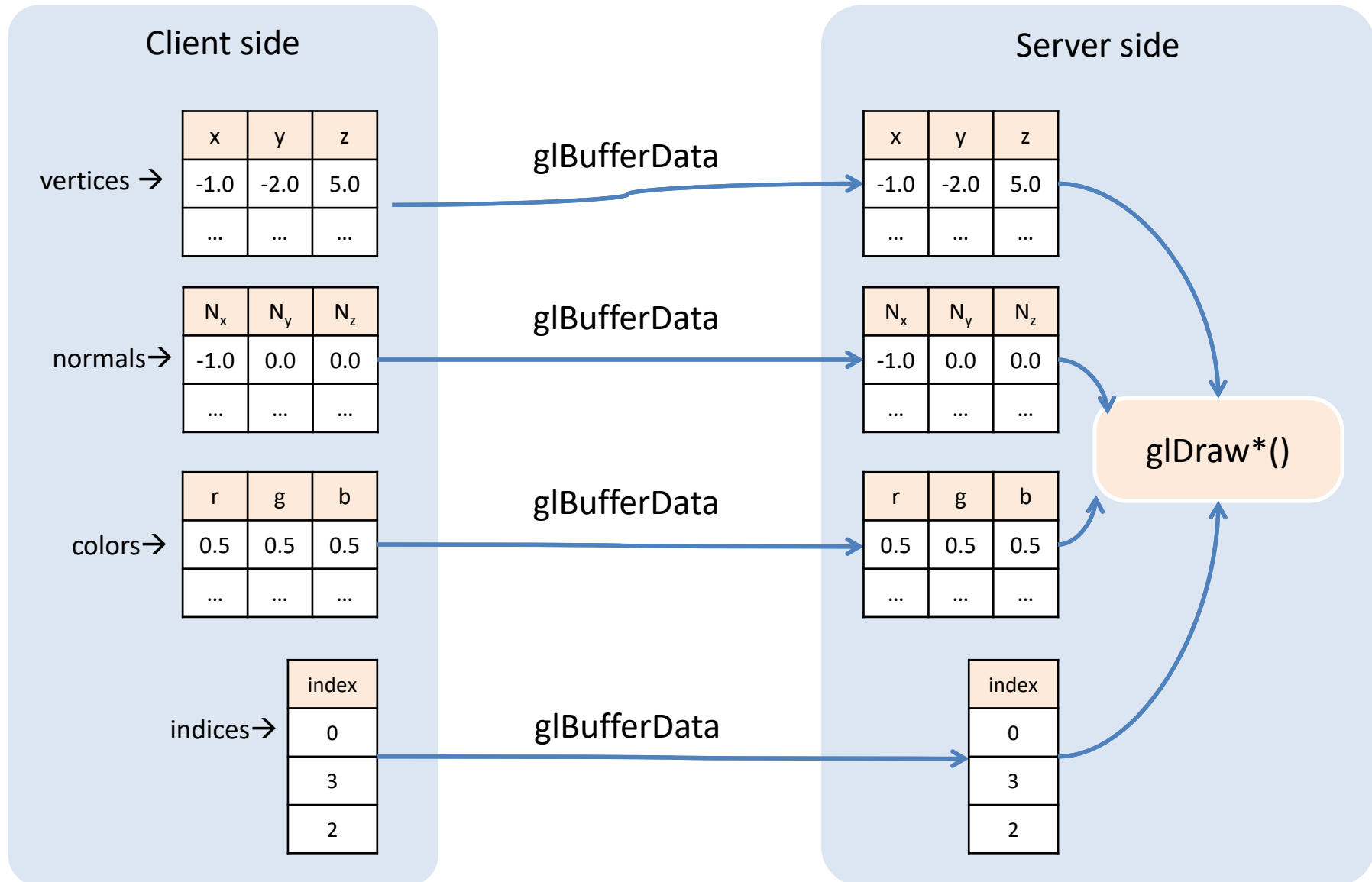
Objectiu:

- Evitar transferir les dades cada frame

Idea:

- Emmagatzemar les dades del VA al servidor!

# Vertex buffer object



## **EXAMPLE 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 índices)

```
// 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);**

## **EXAMPLE 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); // vèrtexs duplicats!
    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 indexes)

```
// 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);**

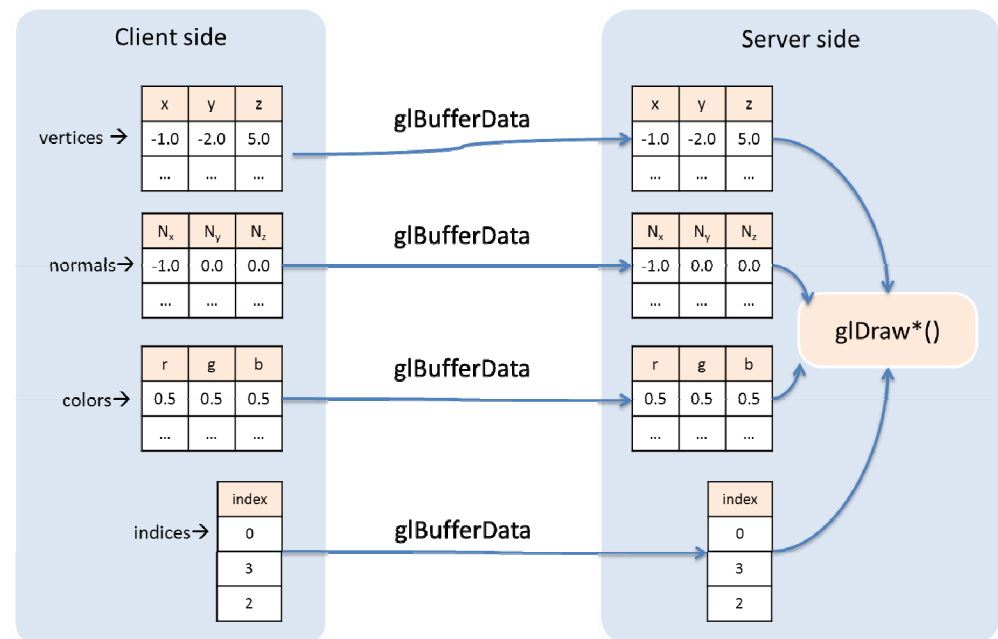
...

~~**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



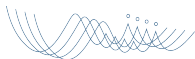
# Render plugins

À. Vinacua, C. Andújar i professors de Gràfics

novembre de 2019

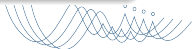
# Tipus de plugins

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



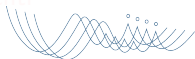
## Tipus de plugins

- Effect Plugins
  - Canvien l'estat d'OpenGL abans i/o després de que es pinti 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. . .



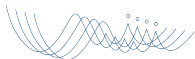
## Fluxe de control

- 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

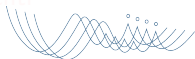
- Quan es carrega un nou plugin, es crida el seu `onPluginLoad()`
- Quan s'afegeix un nou model a l'escena es crida a `onObjectAdd()` de tots els plugins carregats
- Quan s'esborra l'escena, es crida a `onSceneClear()` de tots els plugins carregats
- Els events de ratolí i teclat (`keyPressEvent()`... `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





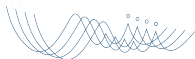
## Fluxe de control

- 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



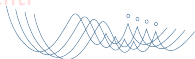
## Fluxe de control

- 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



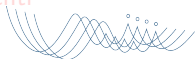
## Fluxe de control

- 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



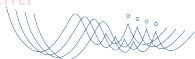
## Fluxe de control

- 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



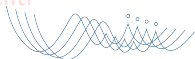
## Fluxe de control

- 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



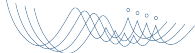
## Fluxe de control

- 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



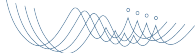
## Fluxe de control

- 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





# 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*: glTexEnv
    - *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**

```
glTexEnvf(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 passen a *pixels* (GL\_BYTE, GL\_FLOAT...)
    - *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 **glTexParameterf**(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 filtres pot no veure's res!!!

# Dibuixat escena

- Comportament més enllà de [0.0, 1.0]:
  - void **glTexParameter**i(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)

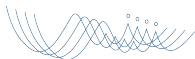


# **Laboratori de Gràfics, Action plugins.**

**À. Vinacua, C. Andújar i professors de Gràfics**

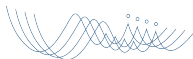
23 de novembre de 2015

# Tipus de plugins



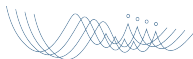
## Tipus de plugins

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



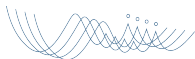
## Tipus de plugins

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



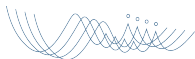
## Tipus de plugins

- Effect Plugins
  - Canvien l'estat d'OpenGL abans i/o després de que es pinti l'escena.
  - Exemples: activar shaders, configurar textures, alpha blending...
- Draw Plugins
  - Recórren 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
  - Dibuir un frame amb un o més passos de rendering.
  - Exemples: múltiples passos de rendering, shadow mapping...

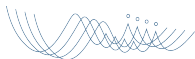


## Tipus de plugins

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

# Fluxe de control

## Per cada refresc:

- 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 ...
```

