

## 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*).

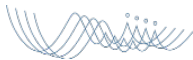
Exercicis de plugins pel visualitzador

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/  
    └── ...  
└── ...
```



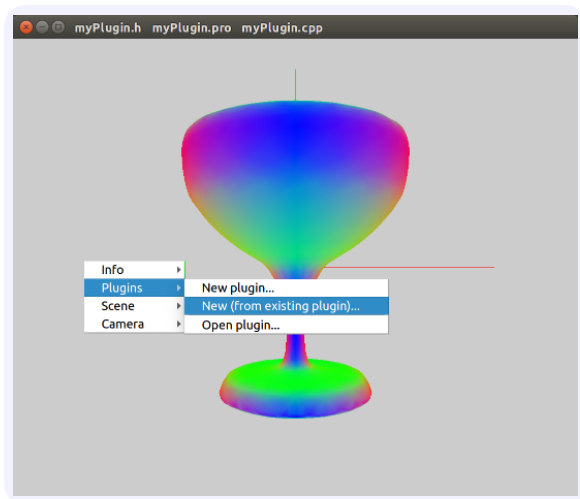
## Codi de partida del Visualitzador

```
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:

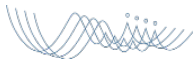
VIMAGE defineix l'executable a fer servir per mostrar imatges

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()
```

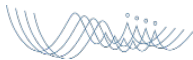


## Exemples d'accés als objectes de l'aplicació

```
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
17     vs = new QOpenGLShader(QOpenGLShader::Vertex, this);
18     vs->compileSourceCode(vs_src);
19     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

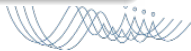
```
1  class Object {
2  public:
3      ...
4      Box boundingBox() const;
5      const vector<Face>& faces() const;
6      const vector<Vertex>& vertices() const;
7      void computeNormals();           // normals *per-cara*
8      void computeBoundingBox();
9      void applyGT(const QMatrix4x4& mat);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

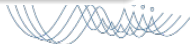




## Support per a shaders a Qt

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(GGLenum outputType);
```



# QOpenGLShader és semblant

Interfície semblant:

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

