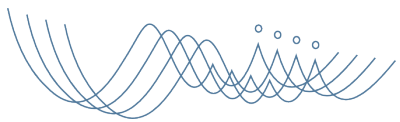
Laboratori de Gràfics, part 2.

À. Vinacua, C. Andújar i professors de Gràfics

5 d'abril de 2017

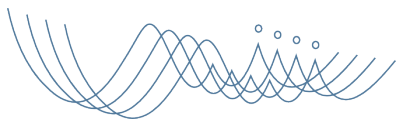
Segona part del laboratori



Segona part del laboratori

Objectius

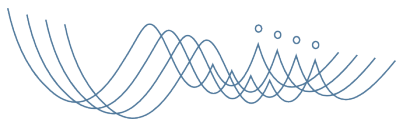
- Completarem un visualitzador d'escenes 3D (semblant al de VIG/IDI), però més eficient i realista
- Eficiència: Vertex Arrays, Vertex Buffer Objects (ho heu vist una mica a IDI...)
- Més realisme: Shaders, Textures, Ombres, Reflexions, Translúcids, ...



Segona part del laboratori

Objectius

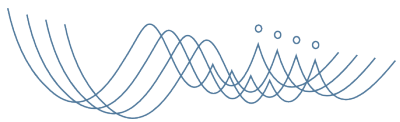
- Completarem un visualitzador d'escenes 3D (semblant al de VIG/IDI), però més eficient i realista
- Eficiència: Vertex Arrays, Vertex Buffer Objects (ho heu vist una mica a IDI...)
- Més realisme: Shaders, Textures, Ombres, Reflexions, Translúcids, ...



Segona part del laboratori

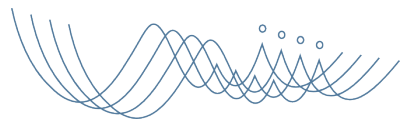
Objectius

- Completarem un visualitzador d'escenes 3D (semblant al de VIG/IDI), però més eficient i realista
- Eficiència: Vertex Arrays, Vertex Buffer Objects (ho heu vist una mica a IDI...)
- Més realisme: Shaders, Textures, Ombres, Reflexions, Translúcids, ...



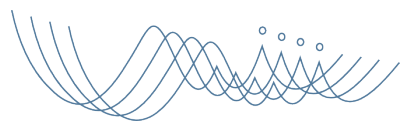
Eines

- C++
- Qt5 (però no caldran gaires coneixements específics)
- OpenGL (Core) + GLSL



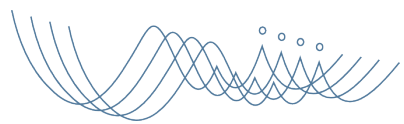
Visualitzador i plugins

- Us proporcionem un visualitzador senzill que haureu de completar via *plugins*.
- Cada exercici de la llista consisteix a implementar un o més *plugins*.



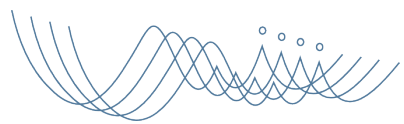
Avaluació

- 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

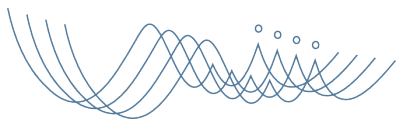


Avaluació

- 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

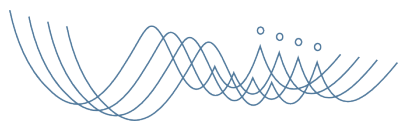


Estructura de directoris



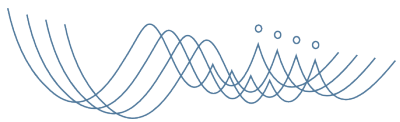
Codi de partida del Visualitzador

```
Viewer/ ←—Directori arrel
        |          de l'aplicació
        |
        |— all.pro
        |
        |— plugins/
        |
        |
        |— viewer/
```



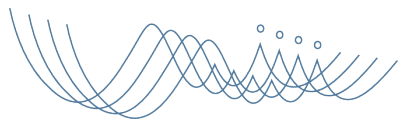
Codi de partida del Visualitzador

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



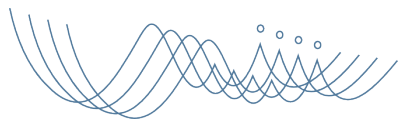
Codi de partida del Visualitzador

```
Viewer/ ←—Directori arrel
          de l'aplicació
  ├— all.pro ←—arxiu pel qmake
               recursiu
  ├— plugins/ ←—fonts dels
                plugins
  └— viewer/
```



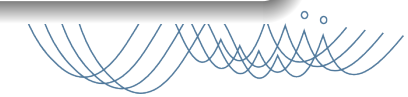
Codi de partida del Visualitzador

```
Viewer/ ←—Directori arrel
          de l'aplicació
  ├— all.pro ←—arxiu pel qmake
               recursiu
  ├— plugins/ ←—fonts dels
                plugins
  └— viewer/ ←—fonts del nucli
               del Viewer
```



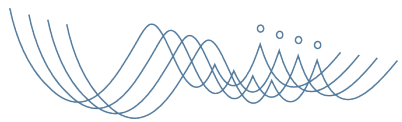
Codi de partida del Visualitzador

```
viewer/
├── bin/
├── app/
│   ├── app.pro
│   └── main.cpp
├── core/
│   ├── core.pro
│   ├── include/
│   └── src/
├── glwidget/
│   ├── glwidget.pro
│   ├── include/
│   └── src/
└── interfaces/
    └── basicplugin.h
```



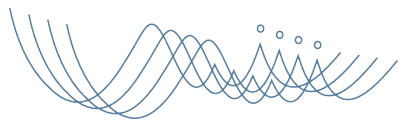
Codi de partida del Visualitzador

```
plugins/
├── bin/
├── common.pro
├── plugins.pro ←—Cal editar-lo per afegir
│                 nous plugins
├── draw-immediate/
│   │
│   ├── draw-immediate.pro
│   │
│   ├── drawimmediate.h
│   └── drawimmediate.cpp
├── navigate-default/
│   └── ...
└── ...
```



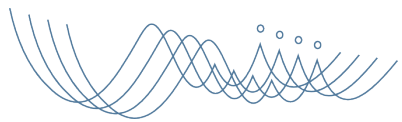
Codi de partida del Visualitzador

```
plugins/
├── bin/
├── common.pro
├── plugins.pro ←—Cal editar-lo per afegir
│                 nous plugins
├── draw-immediate/ ←—Un directori per cada
│   │                 plugin
│   ├── draw-immediate.pro
│   │
│   ├── drawimmediate.h
│   └── drawimmediate.cpp
├── navigate-default/
│   └── ...
└── ...
```

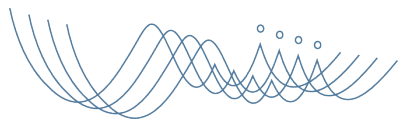


Codi de partida del Visualitzador

```
plugins/
├── bin/
├── common.pro
├── plugins.pro ←—Cal editar-lo per afegir
│                 nous plugins
├── draw-immediate/ ←—Un directori per cada
│   │                 plugin
│   ├── draw-immediate.pro ←—S'ha de dir igual que
│   │                        el directori
│   ├── drawimmediate.h
│   └── drawimmediate.cpp
├── navigate-default/
│   └── ...
└── ...
```



Compilació i Execució



Procediment per a obtenir els binaris (viewer + plugins)

- Desplegar els fonts a un directori en què puguem escriure
- Canviar al directori arrel del Viewer (aquí li direm "Viewer")
- Fer `qmake-qt5` (a la vostra màquina pot dir-se `qmake`)
- Fer `make`
- Els binaris del nucli seran a `Viewer/viewer/bin/` i els dels plugins a `Viewer/plugins/bin/`
- Fixeu-vos que a més de l'executable `viewer`, a `Viewer/viewer/bin/` hi ha dues llibreries dinàmiques. Cal que les pugui trobar en temps d'execució, i per tant cal fer (p.ex., en `tcsh`):

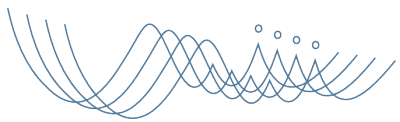
```
1    cd Viewer
2    setenv LD_LIBRARY_PATH $PWD/viewer/bin
```

- ...i ja podeu executar `viewer/bin/viewer`

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/...`
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
- `VTESTS` el directori on hi ha els arxius de test pels shaders
- `VPLUGINS` els plugins a carregar en engegar.

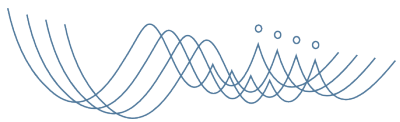


Execució

Si teniu una versió recent del `Viewer`, trobareu al directori arrel de la distribució dos *scripts*:

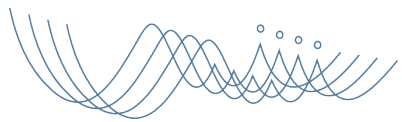
- `Gviewer` per a engegar el `Viewer`
- `GviewerSL` per a engegar el `Viewer` i carregar el `shaderloader`

Aquests scripts assignen la variable d'entorn `LD_LIBRARY_PATH` automàticament, i per tant sols cal afegir aquest directori al `PATH` o indicar el camí complet.



Tipus de plugins

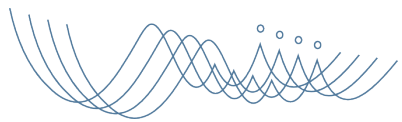
(es tracta d'una distinció semàntica: tant sols hi ha una interfície, comuna a tots els "tipus")



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
  - Recorren els objectes per pintar les primitives de l'escena.
  - Exemples: dibuixar amb `glBegin|glEnd`, 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
  - Dibuixar un frame amb un o més passos de rendering.
  - Exemples: múltiples passos de rendering, shadow mapping...

Sessió 1: Effect plugins



Effect plugins

Mètodes

- `virtual void preFrame();`
- `virtual void postFrame();`
- `virtual void onPluginLoad();`
- `virtual void onObjectAdd();`
- `GLWidget* pglwidget;`
- `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()`
  `//QOpenGLShaderProgram*`

Effect plugins

Mètodes

- `virtual void preFrame();`
- `virtual void postFrame();`
- `virtual void onPluginLoad();`
- `virtual void onObjectAdd();`
- `GLWidget* pglwidget;`
- `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()`
  `//QOpenGLShaderProgram*`

Effect plugins

Mètodes

- `virtual void preFrame();`
- `virtual void postFrame();`
- `virtual void onPluginLoad();`
- `virtual void onObjectAdd();`
- `GLWidget* pglwidget;`
- `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()`
  `//QOpenGLShaderProgram*`

Effect plugins

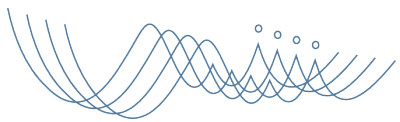
Mètodes

- `virtual void preFrame();`
- `virtual void postFrame();`
- `virtual void onPluginLoad();`
- `virtual void onObjectAdd();`
- `GLWidget* pglwidget;`
- `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()`
  `//QOpenGLShaderProgram*`

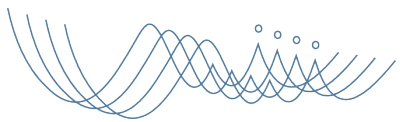
Exemples d'effect plugins: 1/3



alphablending

alphablending.pro

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



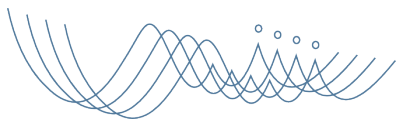
alphablending.h

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

alphablending.cpp

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

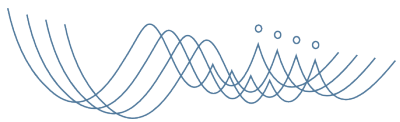
Exemples d'effect plugins: 2/3



effect-crt

effect-crt.pro

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



effectcrt.h

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

effectcrt.cpp

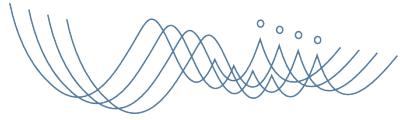
```
1  #include "effectcrt.h"
2
3  void EffectCRT::onPluginLoad()
4  {
5     glwidget()->makeCurrent(); //  !!!
6     QString vs_src =
7       "#version␣330␣core\n"
8       "uniform␣mat4␣modelViewProjectionMatrix;"
9       "in␣vec3␣vertex;"
10      "in␣vec3␣color;"
11      "out␣vec4␣col;"
12      "void␣main()␣{"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

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

effect-crt.cpp...

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

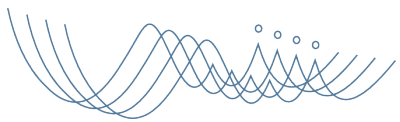
Exemples d'effect plugins: 3/3



show-help

show-help.pro

```
1 TARGET    = $$qtLibraryTarget(show-help)
2 include(../common.pro)
```



show-help.h

```
1  #ifndef _SHOWHELP_H
2  #define _SHOWHELP_H
3  #include "basicplugin.h"
4  class ShowHelp : public QObject, BasicPlugin
5   {
6       Q_OBJECT
7       Q_PLUGIN_METADATA(IID "BasicPlugin")
8       Q_INTERFACES(BasicPlugin)
9   public:
10     void postFrame() Q_DECL_OVERRIDE;
11     void onPluginLoad() Q_DECL_OVERRIDE;
12  private:
13     GLuint textureID;
14     QOpenGLShaderProgram* program;
15     QOpenGLShader* vs;
16     QOpenGLShader* fs;
17  };
18  #endif
```

part of show-help.cpp (1/4)

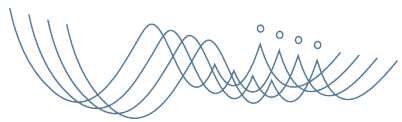
```
1  #include "show-help.h"
2  #include "glwidget.h"
3  #include <QPainter>
4  void ShowHelp::onPluginLoad()
5  {
6    glwidget()->makeCurrent();
7    // Carregar shader, compile & link
8    vs = new QOpenGLShader(QOpenGLShader::Vertex, this);
9    vs->compileSourceFile("plugins/show-help/show.vert");
10   fs = new QOpenGLShader(QOpenGLShader::Fragment, this);
11   fs->compileSourceFile("plugins/show-help/show.frag");
12   program = new QOpenGLShaderProgram(this);
13   program->addShader(vs);
14   program->addShader(fs);
15   program->link();
16 }
```

part of show-help.cpp (2/4)

```
1  void ShowHelp::postFrame()
2  {
3    GLWidget &g=*glwidget(); //  !!!
4    g.makeCurrent();
5    const int SIZE = 1024;
6    // 1. Create image with text
7    QImage image(SIZE,SIZE,QImage::Format_RGB32);
8    image.fill(Qt::white);
9    QPainter painter;
10   painter.begin(&image);
11   QFont font;
12   font.setPixelSize(32);
13   painter.setFont(font);
14   painter.setPen(QColor(50,50,50));
15   int x = 15;
16   int y = 50;
17   painter.drawText(x, y, QString("L␣-␣Load␣object␣␣␣␣␣A␣-␣Add␣pl
18   painter.end();
19 ...
```

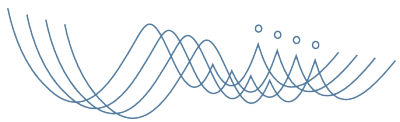
part of show-help.cpp (3/4)

```
1   // 2. Create texture
2   const int textureUnit = 5;
3   g.glActiveTexture(GL_TEXTURE0+textureUnit);
4   QImage im0 = image.mirrored(false, true).convertToFormat(QImag
5   g.glGenTextures( 1, &textureID);
6   g.glBindTexture(GL_TEXTURE_2D, textureID);
7   g.glTexImage2D( GL_TEXTURE_2D, 0, GL_RGBA, im0.width(), im0.he
8   g.glGenerateMipmap(GL_TEXTURE_2D);
9   g.glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LI
10  g.glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LI
11  g.glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT
12  g.glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT
13  ...
```

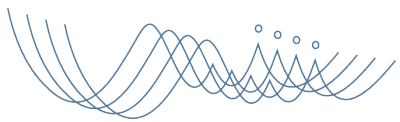


part of show-help.cpp (4/4)

```
1    // Pass 3. Draw quad using texture
2    program->bind();
3    program->setUniformValue("colorMap", textureUnit);
4    program->setUniformValue("WIDTH", float(glwidget()->width()));
5    program->setUniformValue("HEIGHT", float(glwidget()->height())
6    // quad covering viewport
7    drawRect(g);
8    program->release();
9    g.glBindTexture(GL_TEXTURE_2D, 0);
10
11   g.glDeleteTextures(1, &textureID);
12 }
```



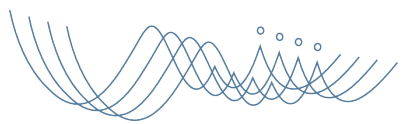
Com afegir un Plugin



Crear nous plugins

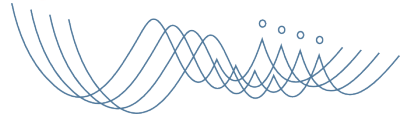
Procediment per afegir un plugin 'MyEffect'

- Crear el directori `plugins/my-effect` (eviteu usar espais)
- Dins d'aquest directori:
  - Editar el fitxer `my-effect.pro`
  - Editar el fitxer `include/my-effect.h`
  - Editar el fitxer `src/my-effect.cpp`
- Afegiu una linia a `plugins/plugins.pro`
  - `SUBDIRS += my-effect`
- qmake + make (des del directori viewer)
- Executar el viewer
- Per carregar un nou plugin al viewer, premeu 'a'



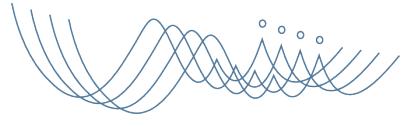
Fluxe de control

- 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



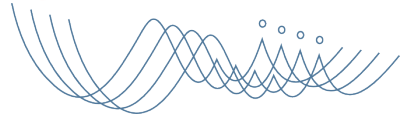
Fluxe de control

- 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



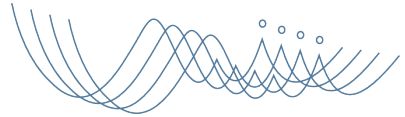
Fluxe de control

- 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



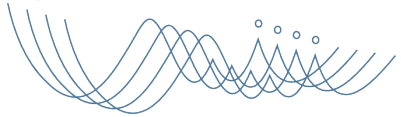
Fluxe de control

- 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



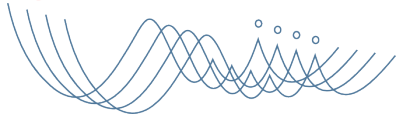
Fluxe de control

- 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



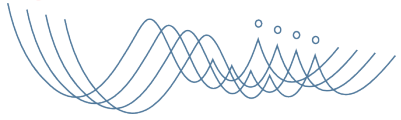
Fluxe de control

- 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



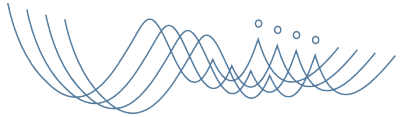
Fluxe de control

- 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



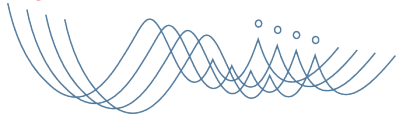
Fluxe de control

- 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



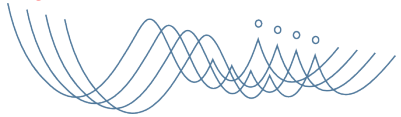
Fluxe de control

- 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

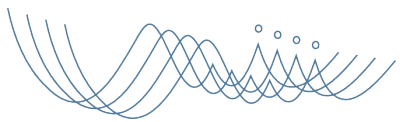


Fluxe de control

- 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



Classes de core/



Classes

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

box: Caixes englobants

camera: Un embolcall per a una càmera rudimentària

face: Cares d'un model

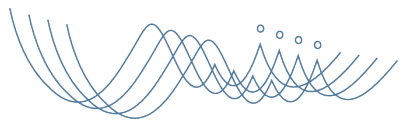
object: objecte (inclou codi per a carregar `.obj`)

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

scene: Model simple d'escena usat pel `GLWidget`.

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

vertex: Model de vèrtex usat a les demés classes.



Classes

Per a representar l'escena:

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

box: Caixes englobants

camera: Un embolcall per a una càmera rudimentària

face: Cares d'un model

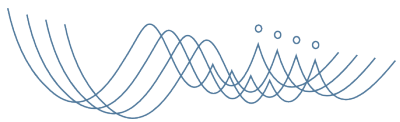
object: objecte (inclou codi per a carregar `.obj`)

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

scene: Model simple d'escena usat pel `GLWidget`.

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

vertex: Model de vèrtex usat a les demés classes.



Classes

Support a la geometria:

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

box: Caixes englobants

camera: Un embolcall per a una càmera rudimentària

face: Cares d'un model

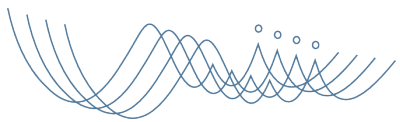
object: objecte (inclou codi per a carregar `.obj`)

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

scene: Model simple d'escena usat pel `GLWidget`.

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

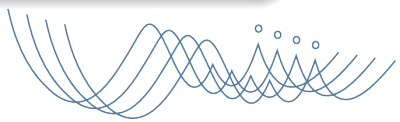
vertex: Model de vèrtex usat a les demés classes.



Vector, Punt

Vector

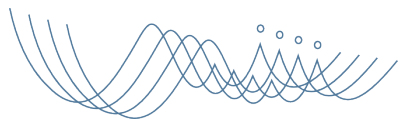
	Vector (qreal *xpos*, qreal *ypos*, qreal *zpos*)
qreal	**length** () const
void	**normalize** ()
Point	**normalized** () const
void	**setX** (qreal *x*)
void	**setY** (qreal *y*)
void	**setZ** (qreal *z*)
qreal	**x** () const
qreal	**y** () const
qreal	**z** () const
Vector	**crossProduct** (const QVector3D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



Vector, Point

Vector

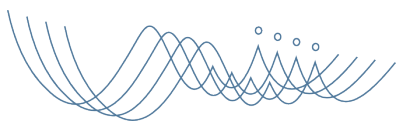
```
1        Vector v(1.0, 0.0, 0.0);
2        float l = v.length();
3        v.normalize();
4        Vector w = v.normalized();
5        v.setX(2.0);
6        v.setY(-3.0);
7        v.setZ(1.0);
8        cout << "[" << v << "]" << endl;
9        Vector u = QVector3D::crossProduct(v,w);
10       float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



Vector, Point

Point

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



Box

```
1  class Box
2  {
3  public:
4      Box(const Point& point=Point());
5      Box(const Point& minimum, const Point& maximum);
6
7      void expand(const Point& p); // incloure un punt
8      void expand(const Box& p); // incloure una 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

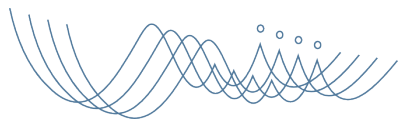
Face té una seqüència ordenada de 3 o més índexs a vèrtex

```
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; // indexs dels vertexs
13 };
```

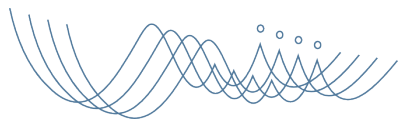
Vertex

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

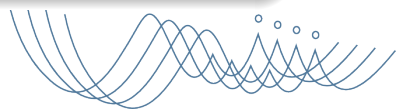


L'API d'OpenGL per a shaders

Passos necessaris

1. Crear *shader objects* amb `glCreateShader()`
2. Assignar-los codi segons convingui amb `glShaderSource()`
3. Compilar cadascun amb `glCompileShader()`
4. Crear un programa (buit) amb `glCreateProgram()`
5. Incloure-hi els *shaders* que calgui amb `glAttachShader()`
6. *Linkar* el programa amb `glLinkProgram()`
7. Activar l'ús del programa amb `glUseProgram()`

Les crides `glGetShader()` i `glGetShaderInfoLog()` permeten comprovar el resultat i obtenir-ne informació adicional. També podem desfer el que hem fet amb `glDetachShader()`, `glDeleteShader()` i `glDeleteProgram()`.



L'API d'OpenGL per a shaders

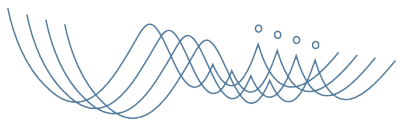
Fluxe d'informació

Atributs

Podem afegir atributs segons sigui necessari amb `glBindAttribLocation()`/`glGetAttribLocation()`, usant `glVertexAttrib*()` entre `glBegin()` i `glEnd()`, tal com ho faríem amb atributs estàndard d'OpenGL.

Uniforms

De forma semblant, disposem de `glGetUniformLocation()` per a obtenir el `GLuint` que identifica una variable d'aquest tipus, i podem ulteriorment donar-li valors amb `glUniform*()` i `glUniformMatrix*()`



Support per a shaders a Qt

Alternativament, 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();
```

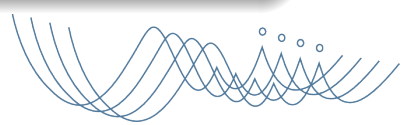
Alguns mètodes de QOpenGLShaderProgram

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;
```