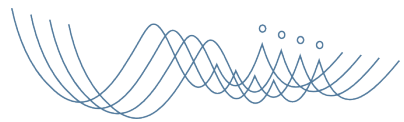
Laboratori de Gràfics, part 2.

À. Vinacua, C. Andújar i professors de Gràfics

12 d'abril de 2015

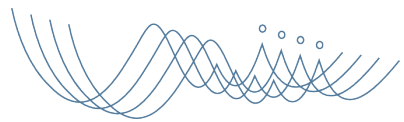
Segona part del laboratori



Segona part del laboratori

Objectius

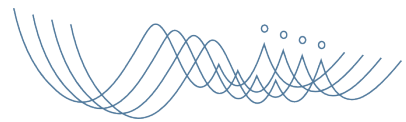
- Completarem un visualitzador d'escenes 3D (no massa diferent del d'IDI), però més eficient i realista
- Eficiència: Vertex Arrays, Vertex Buffer Objects
- Més realisme: Shaders, Textures, Ombres, Reflexions, Translúcids



Segona part del laboratori

Objectius

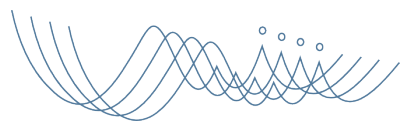
- Completarem un visualitzador d'escenes 3D (no massa diferent del d'IDI), però més eficient i realista
- Eficiència: Vertex Arrays, Vertex Buffer Objects
- Més realisme: Shaders, Textures, Ombres, Reflexions, Translúcids



Segona part del laboratori

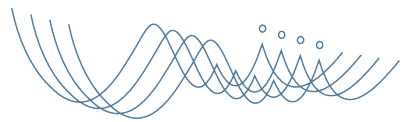
Objectius

- Completarem un visualitzador d'escenes 3D (no massa diferent del d'IDI), però més eficient i realista
- Eficiència: Vertex Arrays, Vertex Buffer Objects
- Més realisme: Shaders, Textures, Ombres, Reflexions, Translúcids



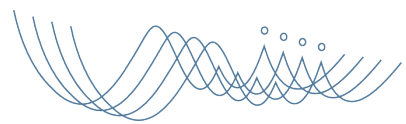
Eines

- C++
- Qt (però molt superficialment)
- OpenGL + GLSL
- GLEW



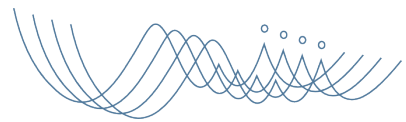
Visualitzador i plugins

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



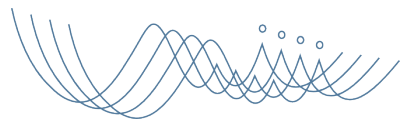
Visualitzador i plugins

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



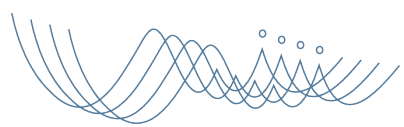
Avaluació

- El control final de laboratori inclourà:
  - Exercicis de shaders per ShaderMaker
  - Exercicis de plugins pel visualitzador
- Els vostres plugins hauran de funcionar sobre el visualitzador original. No feu canvis al codi del nucli que us passem



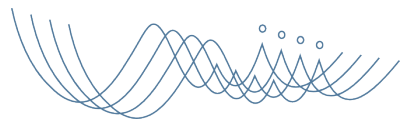
Avaluació

- El control final de laboratori inclourà:
  - Exercicis de shaders per ShaderMaker
  - Exercicis de plugins pel visualitzador
- Els vostres plugins hauran de funcionar sobre el visualitzador original. No feu canvis al codi del nucli que us passem

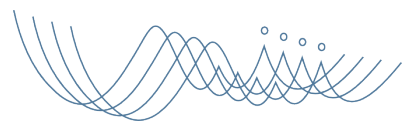


Avaluació

- El control final de laboratori inclourà:
  - Exercicis de shaders per ShaderMaker
  - Exercicis de plugins pel visualitzador
- Els vostres plugins hauran de funcionar sobre el visualitzador original. No feu canvis al codi del nucli que us passem

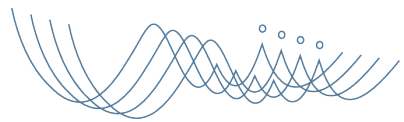


Estructura de directoris



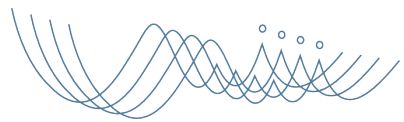
Codi de partida del Visualitzador

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



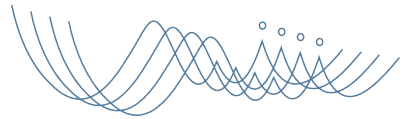
Codi de partida del Visualitzador

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



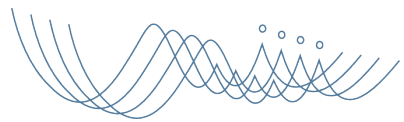
Codi de partida del Visualitzador

```
Viewer/ ←—Directori arrel
          de l'aplicació
  all.pro ←—arxiu pel qmake
             recursiu
  plugins/ ←—fonts dels
              plugins; hi
              trobareu exemples
              i és on heu de
              crear els vostres
              plugins
  viewer/
```

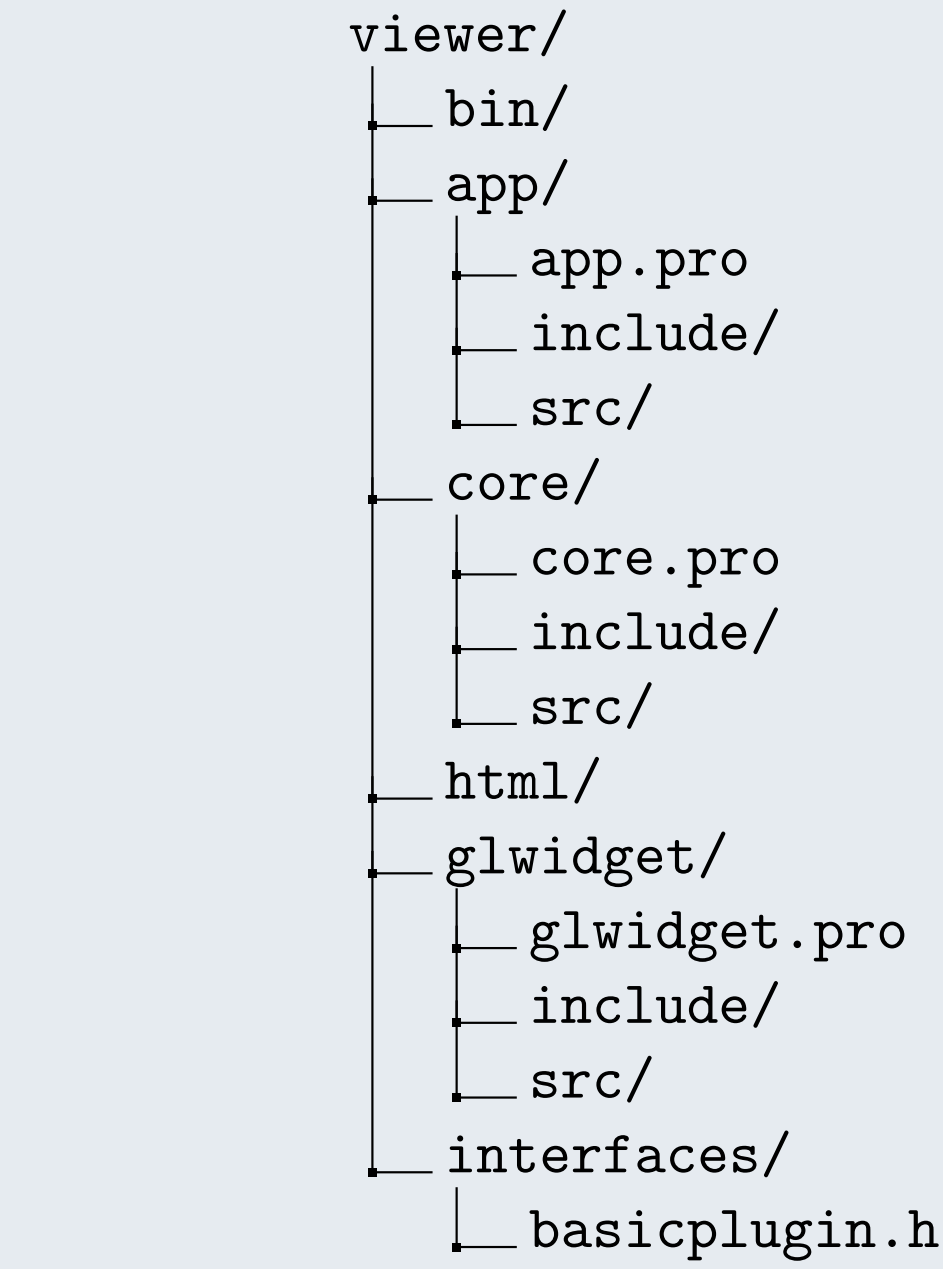


Codi de partida del Visualitzador

```
Viewer/ ←—Directori arrel
          de l'aplicació
  all.pro ←—arxiu pel qmake
             recursiu
  plugins/ ←—fonts dels
              plugins; hi
              trobareu exemples
              i és on heu de
              crear els vostres
              plugins
  viewer/ ←—fonts del nucli
             del Viewer; no
             modifiqueu cap
             arxiu
```

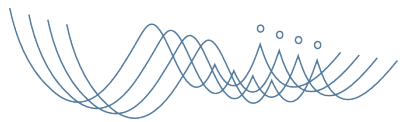


Codi de partida del Visualitzador



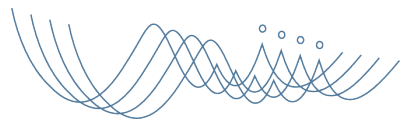
Codi de partida del Visualitzador

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



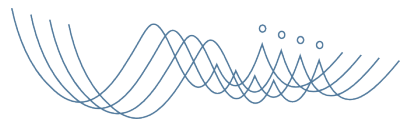
Codi de partida del Visualitzador

```
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/
    └── ...
```

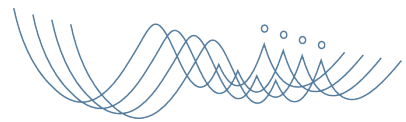


Codi de partida del Visualitzador

```
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

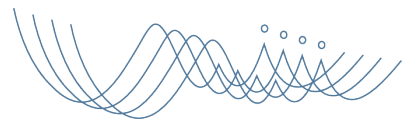


Procediment per a obtenir els binaris (viewer + plugins)

- Desplegar els fonts a un directori en què puguem escriure
- Canviar al directori arrel del Viewer
- Fer `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`):

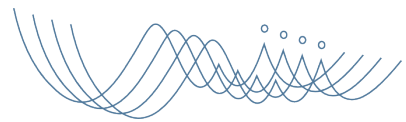
```
1    cd Viewer
2    setenv LD_LIBRARY_PATH $PWD/viewer/bin
```

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



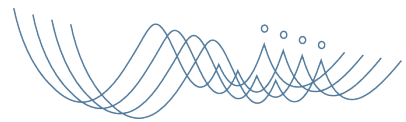
Tipus de plugins

(es tracta d'una distinció semàntica, només hi ha una única interfície)



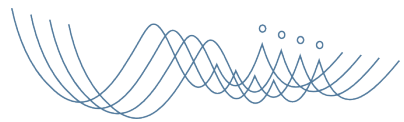
Tipus de plugins

- 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 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...



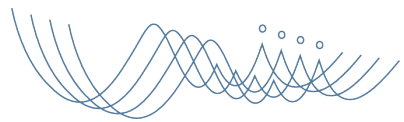
Tipus de plugins

- 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 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...



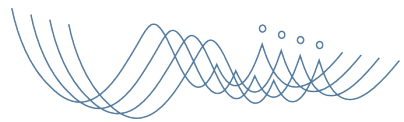
Tipus de plugins

- 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 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...

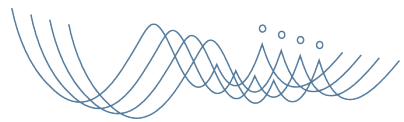


Tipus de plugins

- 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 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 específics

- virtual void preFrame();
- virtual void postFrame();

Altres mètodes

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget();
- Scene* scene();
- Camera* camera();

Exemples d'accés als objectes de l'aplicació

- scene()->objects().size() // num objectes a l'escena
- camera()->getObs() // pos de l'observador

Effect plugins

Mètodes específics

- virtual void preFrame();
- virtual void postFrame();

Altres mètodes

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget();
- Scene* scene();
- Camera* camera();

Exemples d'accés als objectes de l'aplicació

- scene()->objects().size() // num objectes a l'escena
- camera()->getObs() // pos de l'observador

Effect plugins

Mètodes específics

- virtual void preFrame();
- virtual void postFrame();

Altres mètodes

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget();
- Scene* scene();
- Camera* camera();

Exemples d'accés als objectes de l'aplicació

- scene()->objects().size() // num objectes a l'escena
- camera()->getObs() // pos de l'observador

Effect plugins

Mètodes específics

- virtual void preFrame();
- virtual void postFrame();

Altres mètodes

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget();
- Scene* scene();
- Camera* camera();

Exemples d'accés als objectes de l'aplicació

- scene()->objects().size() // num objectes a l'escena
- camera()->getObs() // pos de l'observador

Effect plugins

Mètodes específics

- virtual void preFrame();
- virtual void postFrame();

Altres mètodes

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget();
- Scene* scene();
- Camera* camera();

Exemples d'accés als objectes de l'aplicació

- scene()->objects().size() // num objectes a l'escena
- camera()->getObs() // pos de l'observador

Effect plugins

Mètodes específics

- virtual void preFrame();
- virtual void postFrame();

Altres mètodes

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget();
- Scene* scene();
- Camera* camera();

Exemples d'accés als objectes de l'aplicació

- scene()->objects().size() // num objectes a l'escena
- camera()->getObs() // pos de l'observador

Effect plugins

Mètodes específics

- virtual void preFrame();
- virtual void postFrame();

Altres mètodes

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget();
- Scene* scene();
- Camera* camera();

Exemples d'accés als objectes de l'aplicació

- scene()->objects().size() // num objectes a l'escena
- camera()->getObs() // pos de l'observador

Effect plugins

Mètodes específics

- virtual void preFrame();
- virtual void postFrame();

Altres mètodes

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget();
- Scene* scene();
- Camera* camera();

Exemples d'accés als objectes de l'aplicació

- scene()->objects().size() // num objectes a l'escena
- camera()->getObs() // pos de l'observador

Effect plugins

Mètodes específics

- virtual void preFrame();
- virtual void postFrame();

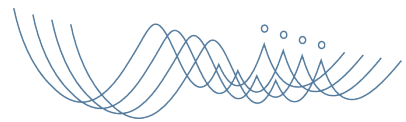
Altres mètodes

- virtual void onPluginLoad()
- virtual void onObjectAdd()
- GLWidget* glwidget();
- Scene* scene();
- Camera* camera();

Exemples d'accés als objectes de l'aplicació

- scene()->objects().size() // num objectes a l'escena
- camera()->getObs() // pos de l'observador

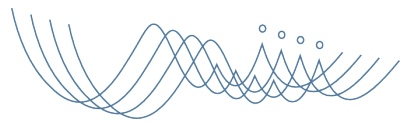
Exemples d'effect plugins: 1/3



alpha-blending

alpha-blending.pro

```
1 TARGET    = $$qtLibraryTarget(alpha-blending)
2 include(../common.pro)
```



alpha-blending

alpha-blending.h

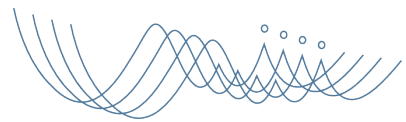
```
1  #ifndef _ALPHABLENDING_H
2  #define _ALPHABLENDING_H
3
4  #include "basicplugin.h"
5
6  class AlphaBlending : public QObject, public BasicPlugin
7   {
8       Q_OBJECT
9       Q_INTERFACES(BasicPlugin)
10
11   public:
12      void preFrame();
13      void postFrame();
14   };
15
16   #endif
```

alpha-blending

alpha-blending.cpp

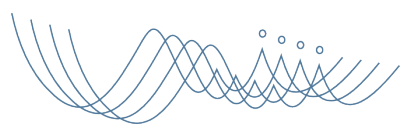
```
1  #include "alpha-blending.h"
2  #include "glwidget.h"
3
4  void AlphaBlending::preFrame() {
5      glDisable(GL_DEPTH_TEST);
6      glBlendFunc(GL_SRC_ALPHA, GL_ONE);
7      glEnable(GL_BLEND);
8  }
9  void AlphaBlending::postFrame() {
10     glEnable(GL_DEPTH_TEST);
11     glDisable(GL_BLEND);
12 }
13 Q_EXPORT_PLUGIN2(alpha-blending, AlphaBlending)
   // plugin name, plugin class
```

Exemples d'effect plugins: 2/3



effect-crt.pro

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

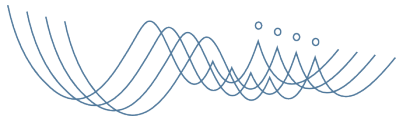


effectcrt.h

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

effectcrt.cpp

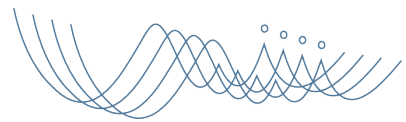
```
1  #include "effectcrt.h"
2
3  void EffectCRT::onPluginLoad()
4  {
5      // Carregar shader, compile & link
6      QString fs_src = "uniform␣int␣n;␣void␣main()␣{if␣(mod((gl
7      fs = new QGLShader(QGLShader::Fragment, this);
8      fs->compileSourceCode(fs_src);
9      program = new QGLShaderProgram(this);
10     program->addShader(fs);
11     program->link();
12 }
```



effectcrt.cpp

```
1  void EffectCRT::preFrame()
2  {
3      // bind shader and define uniforms
4      program->bind();
5      program->setUniformValue("n", 6);
6  }
7
8  void EffectCRT::postFrame()
9  {
10     // unbind shader
11     program->release();
12 }
13
14 Q_EXPORT_PLUGIN2(effectcrt, EffectCRT)
   // plugin name, plugin class
```

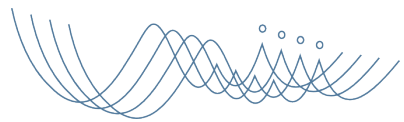
Exemples d'effect plugins: 3/3



show-help

show-help.pro

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

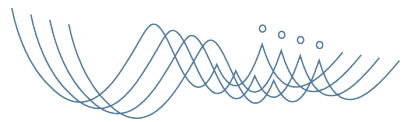


show-help.h

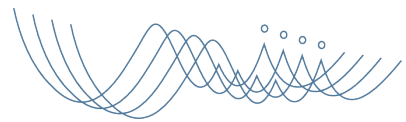
```
1  #ifndef _SHOWHELP_H
2  #define _SHOWHELP_H
3
4  #include "basicplugin.h"
5
6  class ShowHelp : public QObject, public BasicPlugin
7   {
8       Q_OBJECT
9       Q_INTERFACES(BasicPlugin)
10
11   public:
12      void postFrame();
13   };
14   #endif
```

show-help.cpp

```
1  #include "show-help.h"
2  #include "glwidget.h"
3
4  void ShowHelp::postFrame()
5  {
6          glColor3f(0.0, 0.0, 0.0);
7          int x = 5;
8          int y = 15;
9          glwidget()->renderText(x,y, QString("L␣-␣Load␣object␣␣
10 }
11 Q_EXPORT_PLUGIN2(show-help, ShowHelp)
   // plugin name, plugin class
```



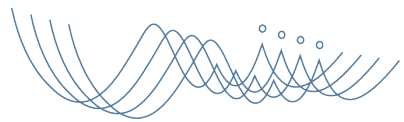
Com afegir un nou Plugin



Crear nous plugins

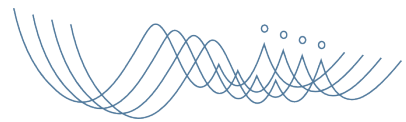
Procediment per afegir un plugin 'MyEffect'

- Crear el directori `plugins/my-effect` (eviteu usar espais)
- Dins d'aquest directori:
  - Editar el fitxer `my-effect.pro`
  - Editar el fitxer `my-effect.h`
  - Editar el fitxer `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'



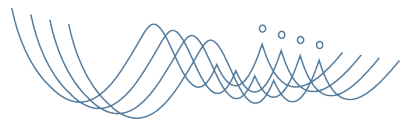
Fluxe de control

- Cada cop que es carrega un nou plugin, es crida el seu onPluginLoad()
- Cada cop que s'afegeix un nou model a l'escena, es crida a onObjectAdd() de tots els plugins.
- GLWidget::paintGL() fa tres coses: 1) crida preFrame() de tots els plugins, 2) crida paintGL() del plugin que ho implementa, i 3) crida postFrame() de tots els plugins.
- Els events de mouse i teclat (keyPressEvent...) es propaguen a tots els plugins.



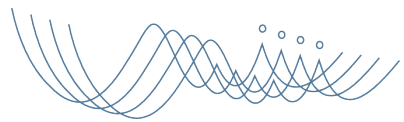
Fluxe de control

- Cada cop que es carrega un nou plugin, es crida el seu onPluginLoad()
- Cada cop que s'afegeix un nou model a l'escena, es crida a onObjectAdd() de tots els plugins.
- GLWidget::paintGL() fa tres coses: 1) crida preFrame() de tots els plugins, 2) crida paintGL() del plugin que ho implementa, i 3) crida postFrame() de tots els plugins.
- Els events de mouse i teclat (keyPressEvent...) es propaguen a tots els plugins.



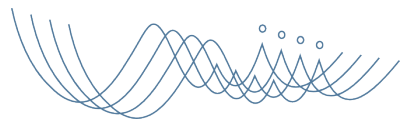
Fluxe de control

- Cada cop que es carrega un nou plugin, es crida el seu onPluginLoad()
- Cada cop que s'afegeix un nou model a l'escena, es crida a onObjectAdd() de tots els plugins.
- GLWidget::paintGL() fa tres coses: 1) crida preFrame() de tots els plugins, 2) crida paintGL() del plugin que ho implementa, i 3) crida postFrame() de tots els plugins.
- Els events de mouse i teclat (keyPressEvent...) es propaguen a tots els plugins.

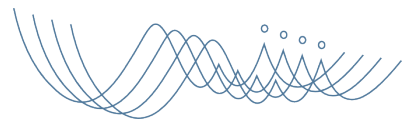


Fluxe de control

- Cada cop que es carrega un nou plugin, es crida el seu onPluginLoad()
- Cada cop que s'afegeix un nou model a l'escena, es crida a onObjectAdd() de tots els plugins.
- GLWidget::paintGL() fa tres coses: 1) crida preFrame() de tots els plugins, 2) crida paintGL() del plugin que ho implementa, i 3) crida postFrame() de tots els plugins.
- Els events de mouse i teclat (keyPressEvent...) es propaguen a 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

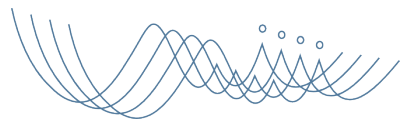
object: objecte (inclou codi per a carregar `.obj`)

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

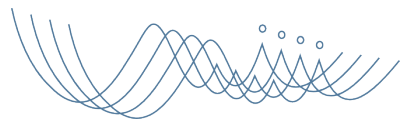
object: objecte (inclou codi per a carregar `.obj`)

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.



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

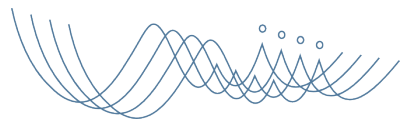
object: objecte (inclou codi per a carregar `.obj`)

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.

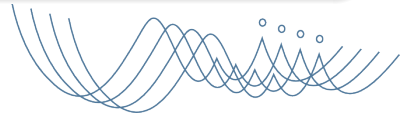
vertex: Model de vèrtex usat a les demés classes.



Vector, Punt

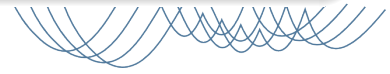
Vector

	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

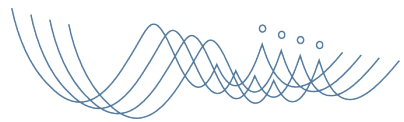
```
1        Vector v(1.0, 0.0, 0.0);
2        double l = v.length();
3        v.normalize();
4        Vector w = v.normalized();
5        v.setX(0.0);
6        v.setY(0.0);
7        v.setZ(1.0);
8        cout << v.x() << endl;
9        cout << v.y() << endl;
10       cout << v.z() << endl;
11       Vector u = QVector3D::crossProduct(v,w);
12       double dot = QVector3D::dotProduct(v,w);
13       Vector u = v + w;
```



Vector, Point

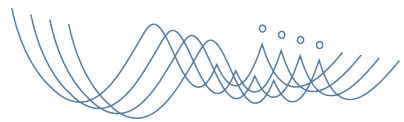
Point

	Point (qreal *xpos*, qreal *ypos*, qreal *zpos*)
void	**setX** (qreal *x*)
void	**setY** (qreal *y*)
void	**setZ** (qreal *z*)
qreal	**x** () const
qreal	**y** () const
qreal	**z** () const
const Point	**operator+** (const QVector3D & *v1*, const QVector3D & *v2*)



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.x() << endl;
6        cout << p.y() << endl;
7        cout << p.z() << endl;
```



Box

```
1  class Box
2  {
3  public:
4      Box(const Point& point=Point());
5
6      void expand(const Point& p); // incloure un punt
7      void expand(const Box& p); // incloure una capsa
8
9      void render();   // dibuixa en filferros
10     Point center() const; // centre de la capsa
11     float radius() const; // meitat de la diagonal
12
13 private:
14     Point pmin, pmax;
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    void addObject(Object &);
8
9    void computeBoundingBox();
10   Box boundingBox() const;
11
12 private:
13   vector<Object> pobjects;
14   Box pboundingBox;
15 };
```

Object

Object té un vector de cares i un vector de vèrtexs

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

Face

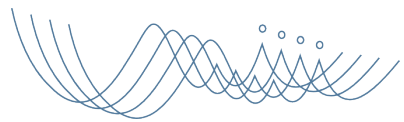
Face té una seqüència ordenada de 3 o 4 índexs a vèrtex

```
1  class Face
2  {
3  public:
4     Face();
5     int numVertices() const;
6     int vertexIndex(int i) const; // index vertex i-essim
7     Vector normal() const;
8
9     void addVertexIndex(int i);
10    void computeNormal(const vector<Vertex> &);
11
12 private:
13    Vector pnormal;
14    vector<int> pvertices; // indexs dels vertexs
15 };
```

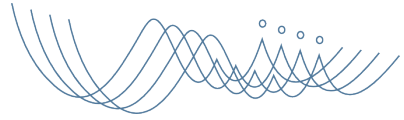
Vertex

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

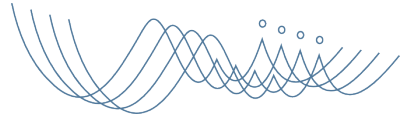


L'API d'OpenGL per a shaders

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()`.

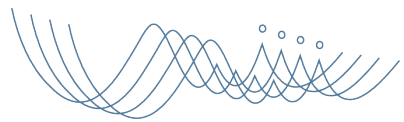


Atributs

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

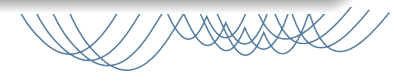
De forma semblant, disposem de `glGetUniformLocation()` per a obtenir el `GLuint` que identifica una variable d'aquest tipus, i podem ulteriorment donar-li valors amb `glUniform*()` i `glUniformMatrix*()`



Support per a shaders a Qt

Alternativament, podeu fer servir QGLShader i QGLShaderProgram

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



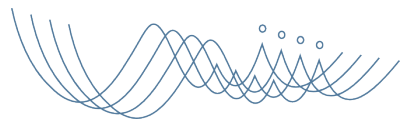
Alguns mètodes de QGLShaderProgram

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);
```



QGLShader és semblant

Interfície semblant:

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