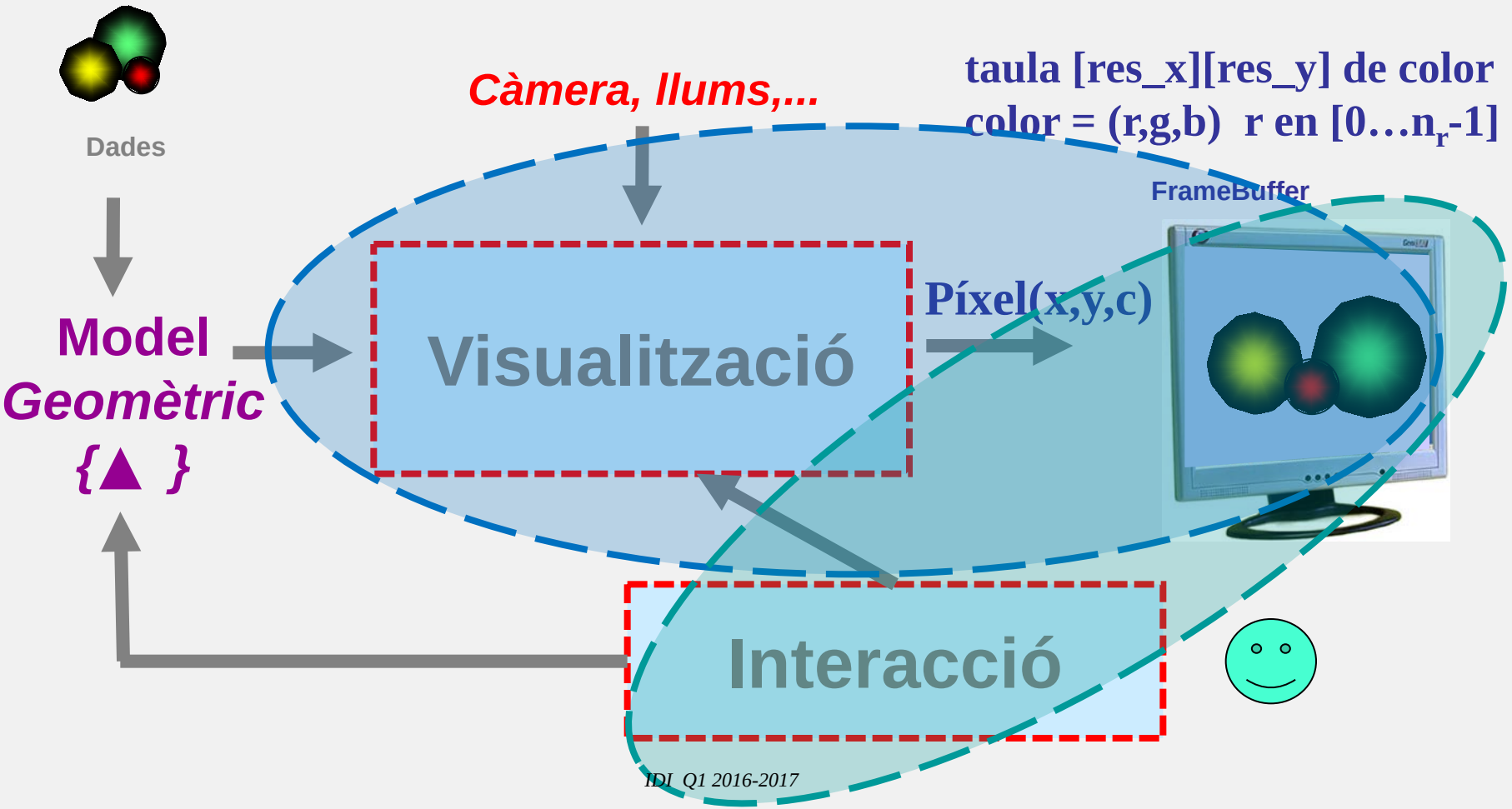
Laboratori OpenGL – Sessió 1

- Introducció
- Llibreria Qt amb OpenGL
- Introducció a OpenGL
  - Què és?
  - Crides per a donar informació del model
  - Pintar
- Exemple complet
  - Fitxer .pro
  - Aplicació Qt en main.cpp
  - Classe MyGLWidget
    - Declaracions: MyGLWidget.h
    - Implementació: MyGLWidget.cpp

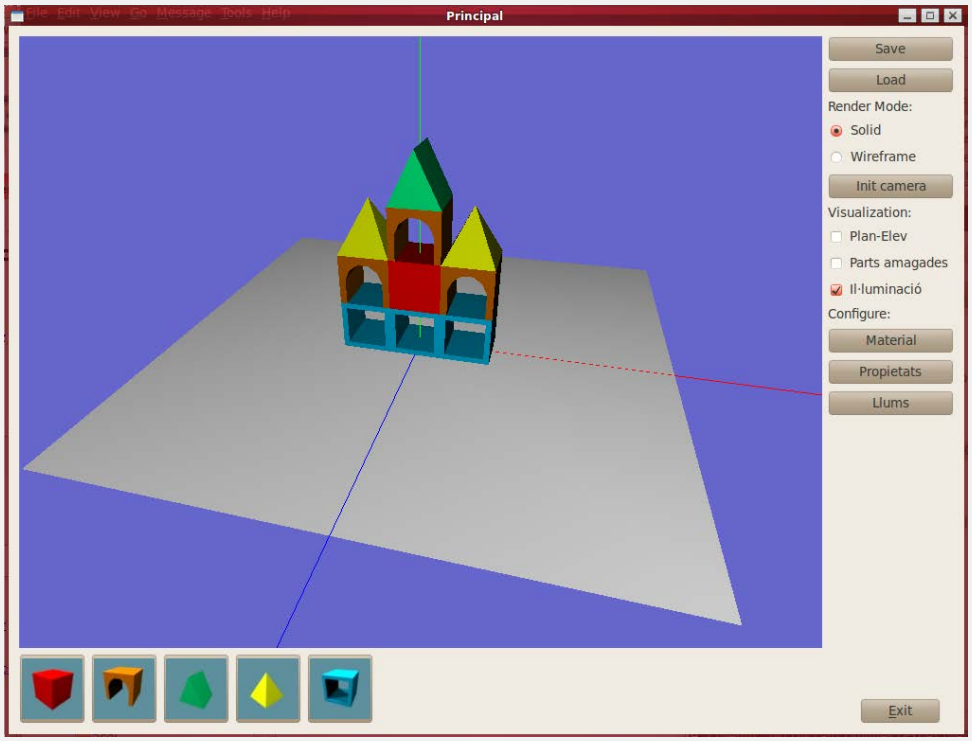
Introducció a OpenGL i Qt

- OpenGL: API per visualització de gràfics 3D.
- Qt: API per a disseny d'interfícies i interacció.



Llibreria Qt amb OpenGL

- Qt pot ser usat per aplicacions OpenGL mitjançant la classe virtual QOpenGLWidget.
- Cal afegir al fitxer .pro la sentènccia: QT += opengl



OpenGL amb Qt

Per usar OpenGL amb Qt cal derivar una classe de QOpenGLWidget.

Mètodes virtuals a implementar:

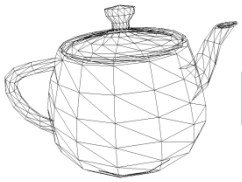
- *initializeGL* ()
  - Codi d'inicialització d'OpenGL.
  - Qt la cridarà abans de la 1ª crida a *resizeGL.*
- *paintGL* ()
  - Codi per redibuixar l'escena.
  - Qt la cridarà cada cop que calgui el repintat. El swapBuffers() és automàtic per defecte.
- *resizeGL* ()
  - Codi per redefinir l'àrea de dibuix (viewport).
  - Qt la cridarà quan es creï la finestra, i cada cop que es modifiqui la mida de la finestra.

Introducció a OpenGL

- API per visualització de gràfics 3D
  - Només visualització 3D
  - Cap funció de gestió d'entrada/events
  - Cap funció de gestió de finestres

- Aspectes bàsics
  - A cada frame es redibuixa tota l'escena.
  - Animació via doble-buffering
  - Màquina d'estats

Paradigma projectiu simplificat/bàsic



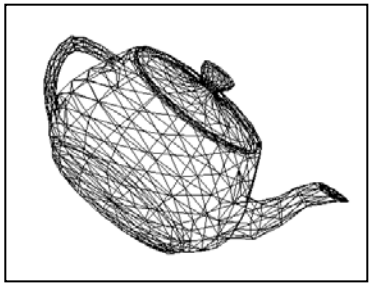
∀ polígon

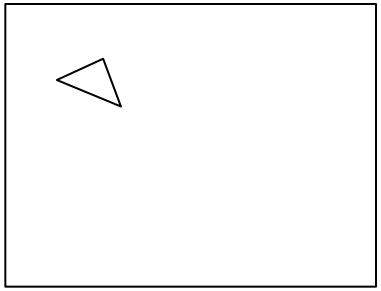
Processament de vèrtexs

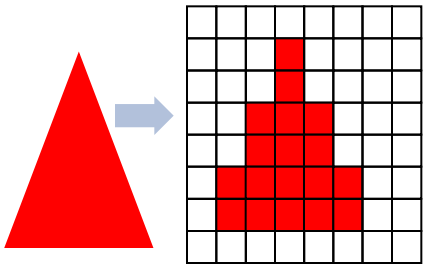
Rasterització

Processament de fragments

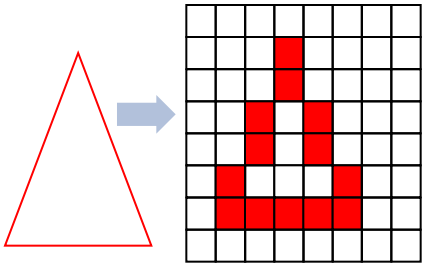
Imatge



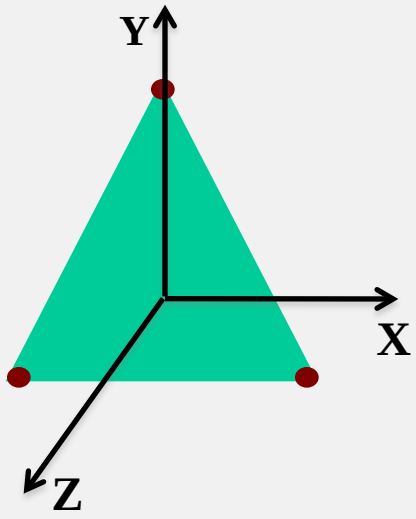




{(xf ,yf, zf, c)}



Informació del model



Possible informació associada a un vèrtex:

- Posició (coordenades)
- Color (rgb/rgba)
- Vector normal (coordenades)
- …

Per a cada model cal generar un Vertex Array Object (VAO).

Les dades dels vèrtexs s'han de passar a la tarja gràfica guardats en Vertex Buffer Object (VBO).

Pintarem els VAOs.

Informació del model

Per a generar un VAO:

void glGenVertexArrays (GLsizei *n*, GLuint **arrays*);

Genera *n* identificadors per a VAOs i els retorna a *arrays*

n : nombre de VAOs a generar

arrays : vector de GLuint on els noms dels VAO generats es retornen

void glBindVertexArray (GLuint *array*);

Activa el VAO identificat per *array*

array : nom del VAO a activar

Informació del model

Per a generar i omplir de dades un VBO:

void glGenBuffers (GLsizei *n*, GLuint **buffers*);

Genera *n* identificadors per a VBOs i els retorna a *buffers*

n : nombre de VBOs a generar

buffers : vector de Gluint on els noms dels VBO generats es retornen

void glBindBuffer (GLenum *target*, GLuint *buffer*);

Activa el VBO identificat per *buffer*

target : tipus de buffer de la GPU que s'usarà (GL_ARRAY_BUFFER, …)

buffer : nom del VBO a activar

void glBufferData (GLenum *target*, GLsizeiptr *size*, const GLvoid **data*, GLenum *usage*);

Envia les dades que es troben en *data* per a què siguin emmagatzemades a la GPU

target : tipus de buffer de la GPU que s'usarà (GL_ARRAY_BUFFER, …)

size : mida en bytes de les dades

data : apuntador a les dades

usage : patró d'ús esperat per a aquestes dades (GL_STATIC_DRAW, GL_DYNAMIC_DRAW, …)

Informació del model

Per a indicar a la GPU l'atribut dels vèrtexs a tenir en compte:

void glVertexAttribPointer (GLuint *index*, GLint *size,* GLenum *type,*
GLboolean *normalized,* GLsizei *stride,* const GLvoid **pointer*);

Indica les característiques de l'atribut del vèrtex identificat per *index*

index : nom de l'atribut

size : nombre de components que composen l'atribut

type : tipus de cada component (GL_FLOAT, GL_INT, …)

normalized : indica si els valors de cada component s'han de normalitzar

stride : offset en bytes entre dos atributs consecutius (normalment 0)

pointer : offset del primer component del primer atribut respecte al buffer (normalment 0)

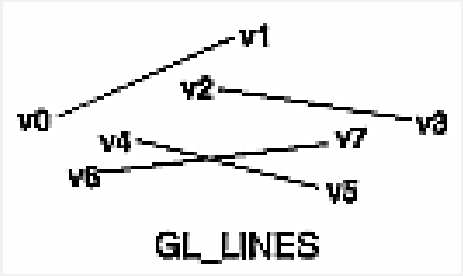
void glEnableVertexAttribArray (GLuint *index*);

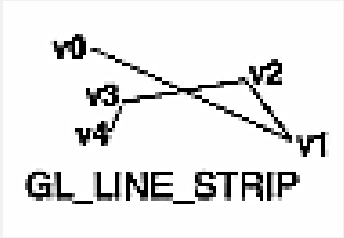
Activa l'atribut del vèrtex identificat per *index*

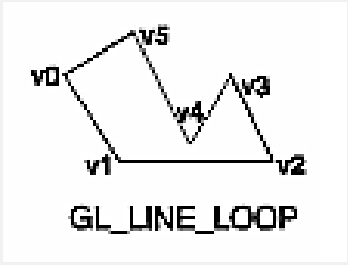
index : nom de l'atribut a activar

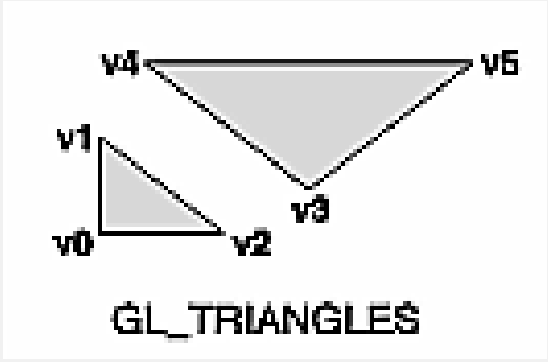
Primitives en OpenGL

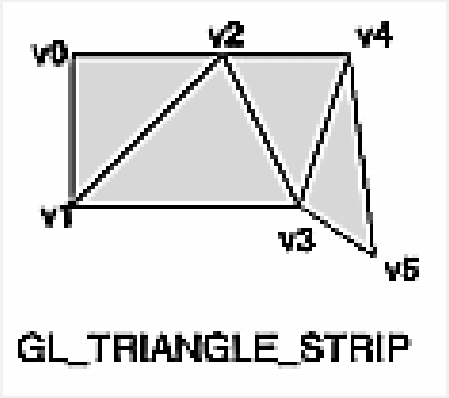
- Totes les primitives s'especifiquen mitjançant vèrtexs:

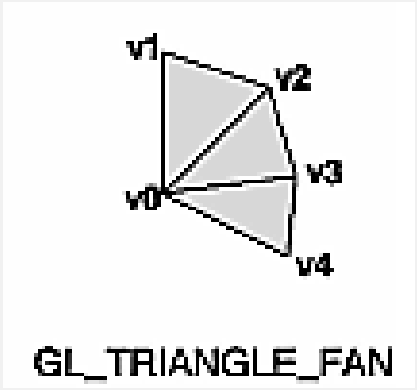












Pintar un VAO

Per a pintar un VAO:

1) Activar el VAO amb glBindVertexArray (GLuint *array*);
2) Pintar el VAO:

   void glDrawArrays (GLenum *mode*, GLint first, GLsizei *count*);

   mode : tipus de primitiva a pintar (GL_TRIANGLES, …)

   first : índex del primer element de l'array

   count : nombre d'elements a tenir en compte de l'array

Exemple complet

- Exemple que teniu a /assig/idi/blocs/bloc-1

Defineix els components de l'aplicació

Bloc1_exemple.pro

Disseny de la interfície

MyForm.ui

Programa principal

main.cpp

Classe que hereta de QOpenGLWidget
Implementa tot el procés de pintat

MyGLWidget.h

MyGLWidget.cpp

Classe que engloba la interfície

MyForm.h

MyForm.cpp

Exemple complet:
Bloc1_exemple.pro

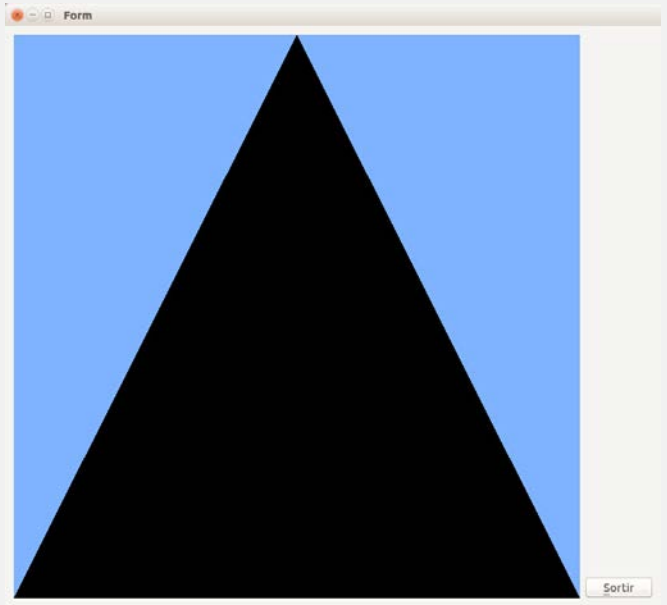
```
TEMPLATE = app
QT += opengl

INCLUDEPATH += /usr/include/glm

FORMS += MyForm.ui

HEADERS += MyForm.h MyGLWidget.h

SOURCES += main.cpp \
           MyForm.cpp MyGLWidget.cpp
```



Exemple complet:

main.cpp

```
#include <QApplication>
#include "MyForm.h“

int main (int argc, char **argv)
{
  QApplication a(argc, argv);

  QSurfaceFormat f;
  f.setVersion (3, 3);
  f.setProfile (QSurfaceFormat::CoreProfile);
  QSurfaceFormat::setDefaultFormat (f);

  MyForm myf;
  myf.show ();

  return a.exec ();
}
```

Exemple complet:
MyGLWidget.h

```
#include <QOpenGLFunctions_3_3_Core>
#include <QOpenGLWidget>
    …….    //  ho explicarem el proper dia
#include "glm/glm.hpp"

class MyGLWidget : public QOpenGLWidget, protected QOpenGLFunctions_3_3_Core
{
  Q_OBJECT
  public:
    MyGLWidget (QWidget *parent=0);
    ~MyGLWidget ();
  protected:
    virtual void initializeGL ();  // Inicialitzacions del contexte gràfic
    virtual void paintGL ();  // Mètode de pintat
    virtual void resizeGL (int width, int height);  // Es crida quan canvia dimensió finestra
  private:
    void createBuffers ();
        ……. //  ho explicarem el proper dia
    GLuint VAO, VBO, vertexLoc;
};
```

Exemple complet:
MyGLWidget.h

```
#include <QOpenGLFunctions_3_3_Core>
#include <QOpenGLWidget>
    …….    //  ho explicarem el proper dia
#include "glm/glm.hpp"

class MyGLWidget : public QOpenGLWidget, protected QOpenGLFunctions_3_3_Core
{
  Q_OBJECT
  public:
    MyGLWidget (QWidget *parent=0);
    ~MyGLWidget ();
  protected:
    virtual void initializeGL ();  // Inicialitzacions del contexte gràfic
    virtual void paintGL ();  // Mètode de pintat
    virtual void resizeGL (int width, int height);  // Es crida quan canvia dimensió finestra
  private:
    void createBuffers ();
        ……. //  ho explicarem el proper dia
    GLuint VAO, VBO, vertexLoc;
};
```

Exemple complet:
MyGLWidget.h

```
#include <QOpenGLFunctions_3_3_Core>
#include <QOpenGLWidget>
    …….    //  ho explicarem el proper dia
#include "glm/glm.hpp"

class MyGLWidget : public QOpenGLWidget, protected QOpenGLFunctions_3_3_Core
{
  Q_OBJECT
  public:
    MyGLWidget (QWidget *parent=0);
    ~MyGLWidget ();
  protected:
    virtual void initializeGL ();  // Inicialitzacions del contexte gràfic
    virtual void paintGL ();  // Mètode de pintat
    virtual void resizeGL (int width, int height);  // Es crida quan canvia dimensió finestra
  private:
    void createBuffers ();
        ……. //  ho explicarem el proper dia
    GLuint VAO, VBO, vertexLoc;
};
```

Exemple complet:
MyGLWidget.cpp (1)

```
#include "MyGLWidget.h"

MyGLWidget::MyGLWidget (QWidget* parent) : QOpenGLWidget (parent)
{
  setFocusPolicy(Qt::ClickFocus);  // per rebre events de teclat
}
MyGLWidget::~MyGLWidget ()
{
   if (program != NULL)  delete program;
}

void MyGLWidget::initializeGL ()
{
  // cal inicialitzar l'ús de les funcions d'OpenGL
  initializeOpenGLFunctions ();

  glClearColor (0.5, 0.7, 1.0, 1.0); // defineix color de fons (d'esborrat)
   …….    //  ho explicarem el proper dia
  createBuffers();
}
```

Exemple complet:
MyGLWidget.cpp (2)

```
void MyGLWidget::createBuffers ()
{
  glm::vec3 Vertices[3];  // Tres vèrtexs amb X, Y i Z
    Vertices[0] = glm::vec3(-1.0, -1.0, 0.0);
    Vertices[1] = glm::vec3(1.0, -1.0, 0.0);
    Vertices[2] = glm::vec3(0.0, 1.0, 0.0);
  // Creació del Vertex Array Object (VAO) que usarem per pintar
  glGenVertexArrays(1, &VAO);
  glBindVertexArray(VAO);
  // Creació del buffer amb les dades dels vèrtexs
  glGenBuffers(1, &VBO);
  glBindBuffer(GL_ARRAY_BUFFER, VBO);
  glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);
  // Activem l'atribut que farem servir per vèrtex
  glVertexAttribPointer(vertexLoc, 3, GL_FLOAT, GL_FALSE, 0, 0);
  glEnableVertexAttribArray(vertexLoc);
  // Desactivem el VAO
  glBindVertexArray(0);
}
```

Exemple complet:
MyGLWidget.cpp (3)

```
void MyGLWidget::paintGL ()
{
  glClear (GL_COLOR_BUFFER_BIT);  // Esborrem el frame-buffer

  // Activem l'Array a pintar
  glBindVertexArray(VAO);
  // Pintem l'escena
  glDrawArrays(GL_TRIANGLES, 0, 3);
  // Desactivem el VAO
  glBindVertexArray(0);
}

void MyGLWidget::resizeGL (int w, int h)
{
  glViewport (0, 0, w, h);  // Definim el viewport per a que ocupi tota la finestra
}
```

Exercicis sessió 1

El que cal que feu en aquesta sessió és:

1) Copieu-vos l'exemple, compileu-lo i proveu-lo.
2) Feu els exercicis que teniu al guió per a aquesta sessió:
    1) Jugueu amb les coordenades dels vèrtexs, tingueu en compte que el món que estem veient és aquell en què x, y, i z pertanyen a [-1, 1].
    2) Fes que pinti un quadrat (usant triangles i triangle-strip).
    3) Fes que pinti una caseta (3 triangles). Es pot fer també amb triangle-strip?
    4) Pinta dos objectes. Cal crear un nou VAO per al segon objecte, així com el VBO corresponent i l'atribut també. A l'hora de pintar cal pintar tots dos objectes.