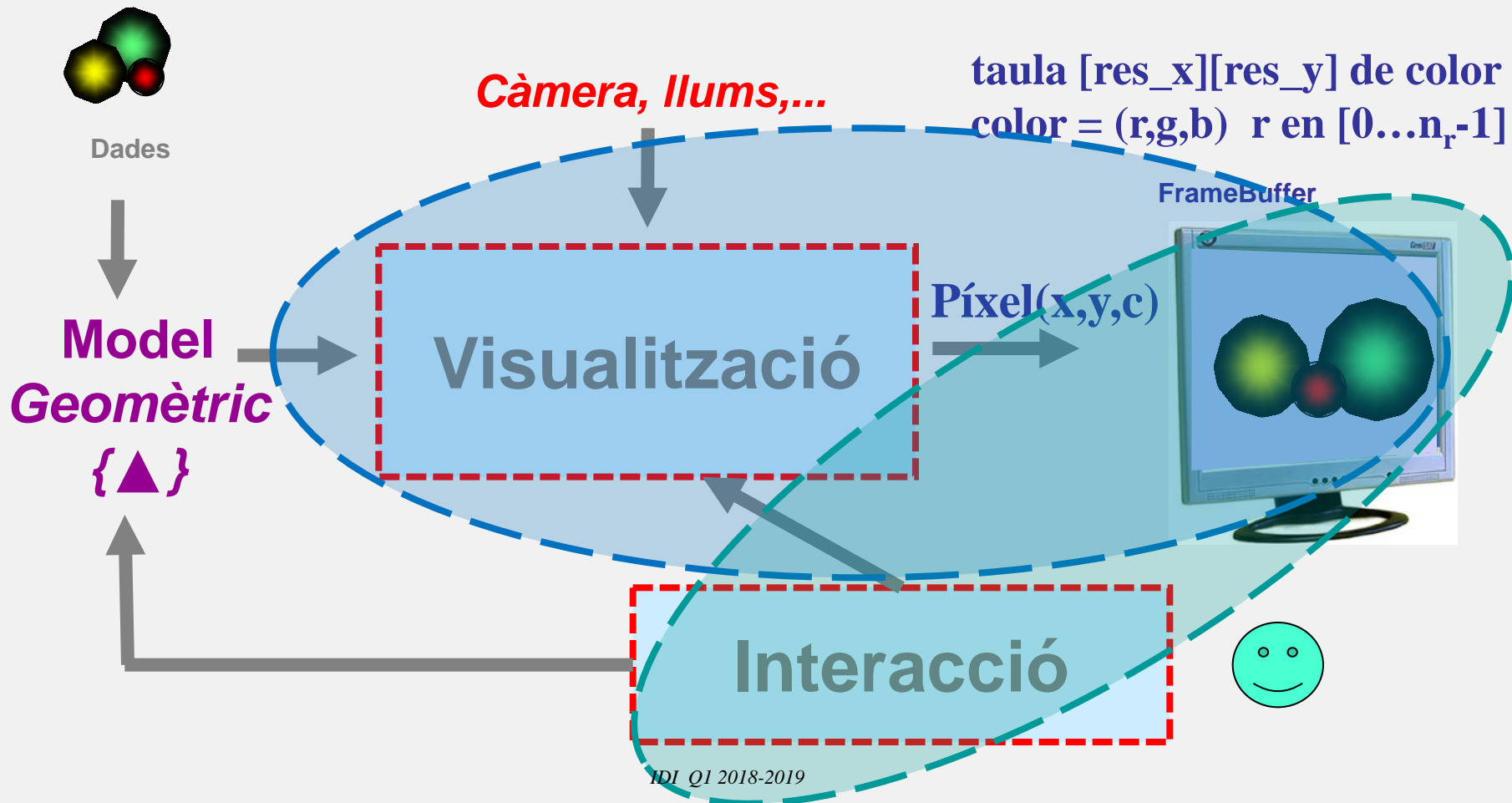


Laboratori OpenGL – Sessió 1.1

- Introducció
- Llibreria Qt
 - Introducció a Qt
 - Qt amb OpenGL
- Introducció a OpenGL
 - Què és?
 - Crides per a donar informació del model
 - Pintar
- Exemple esquelet 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

- Va ser creada per Trolltech i actualment és una companyia pròpia (Qt Group Plc.) que té 38 partners.

<https://www.qt.io>

- Per a plataformes Windows, Linux, Mac i Android

Llibreria Qt

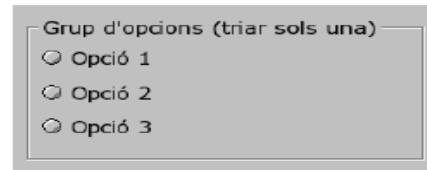
- Una llibreria en C++ per a dissenyar interfícies gràfiques d'usuari (GUI) en diferents plataformes.
- Proporciona diversos components atòmics (widgets) configurables.



(a)
Un
botó



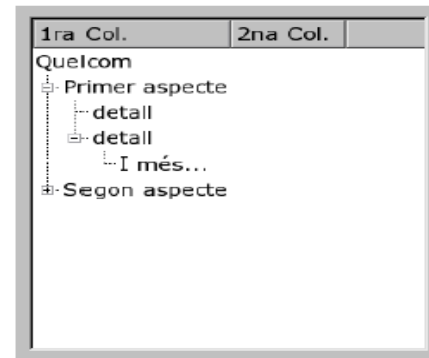
(b) Un "slider"



(c) "Radio buttons"



(d) Barra de "scroll"



(e) Llista jeràrquica

Projecte Qt

main.cpp

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

```
int main (int argc, char **argv)
{
```

```
→   QApplication a(argc, argv);
```

```
    MyForm myf;
    myf.show ();
```

```
→   return a.exec ();
}
```

Projecte Qt

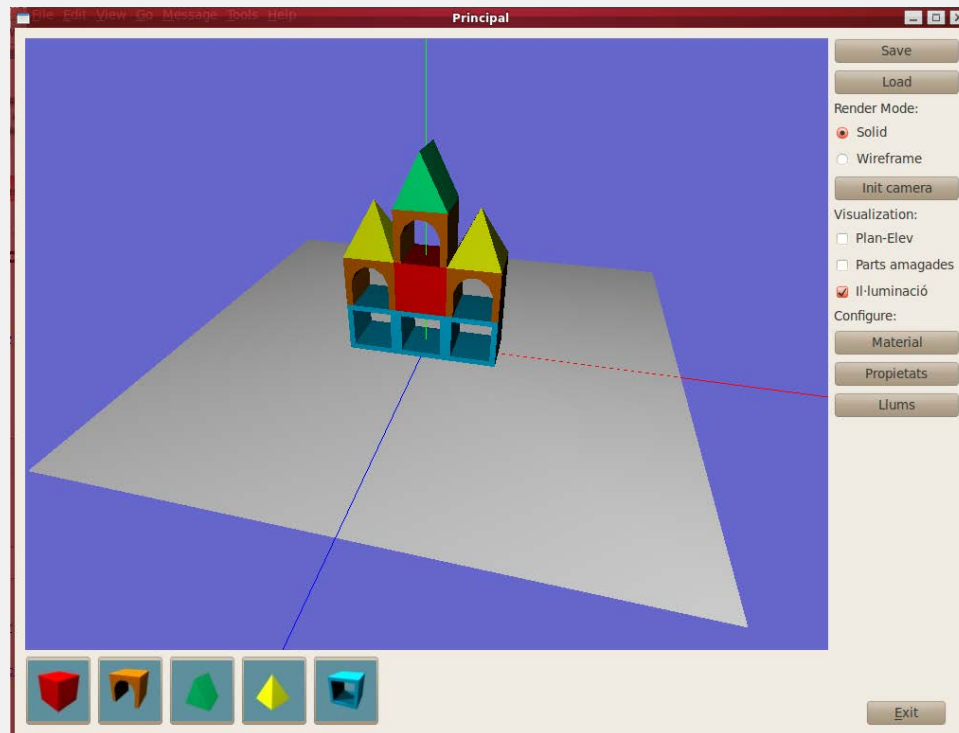
- Crear un fitxer `.pro` que conté la descripció del projecte que estem programant
- Utilitzar les comandes `qmake` i `make`.
 - `qmake` genera el `Makefile` a partir del `.pro`
 - `make` compila i enllaça.

Compilar i enllaçar

- Crear un fitxer “*helloQt.pro*”
 - TEMPLATE = app
 - QT += widgets
 - DEPENDPATH += .
 - INCLUDEPATH += .
 - #Input
 - SOURCES += main.cpp
- Compilem i enllacem
 - qmake (al laboratori cal fer **qmake-qt5**)
 - make
- Executable anomenat *helloQt* en el directori on estiguem.
- Executar-lo amb:
 - ./helloQt

Projecte Qt usant OpenGL

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



OpenGL amb Qt

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

Mètodes virtuals que cal implementar:

- *initializeGL ()*

- Codi d'inicialització d'OpenGL.
- Qt la cridarà abans de la 1^a crida a *resizeGL*.

- *paintGL ()*

- Codi per redibuixar l'escena.
- Qt la cridarà cada cop que calgui el repintat. El `swapBuffers()` és automàtic per defecte.

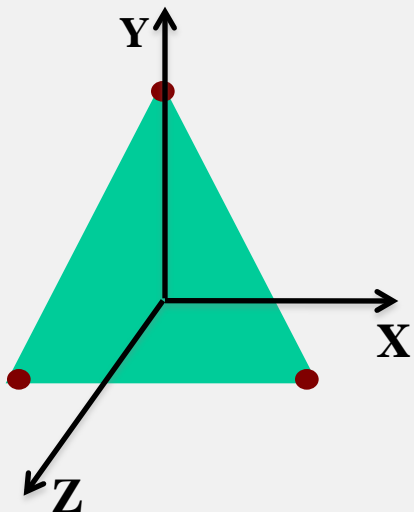
- *resizeGL ()*

- Codi que cal fer quan es redimensiona la finestra.
- 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

Informació del model



Possible informació associada a un vèrtex:

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

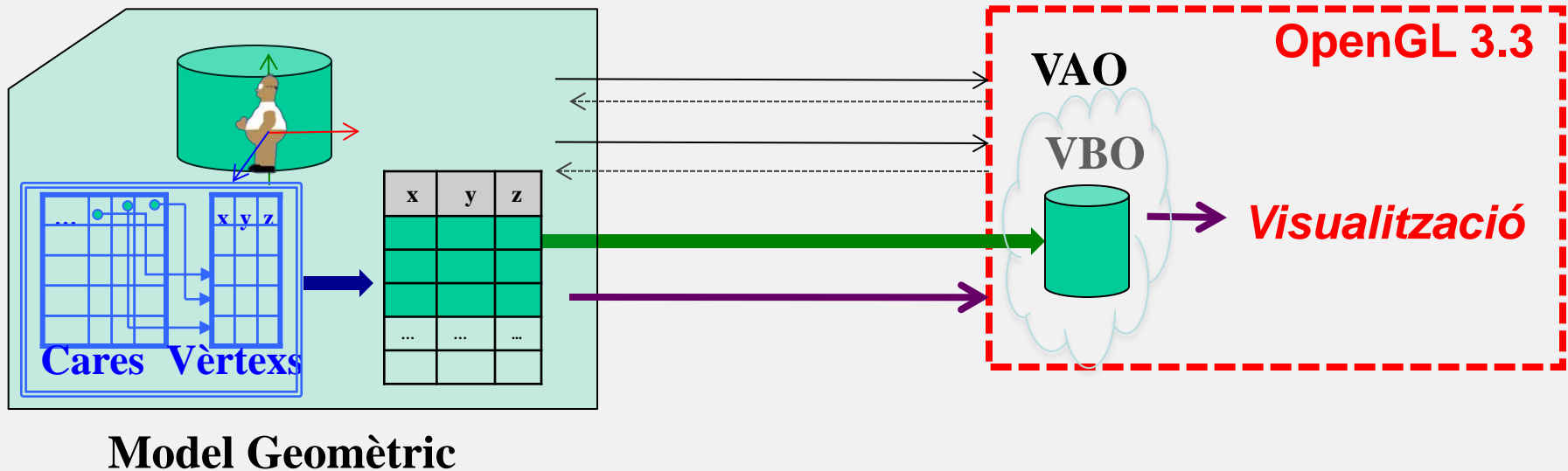
Per a cada model cal generar un Vertex Array Object (VAO).

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

Pintarem els VAOs.

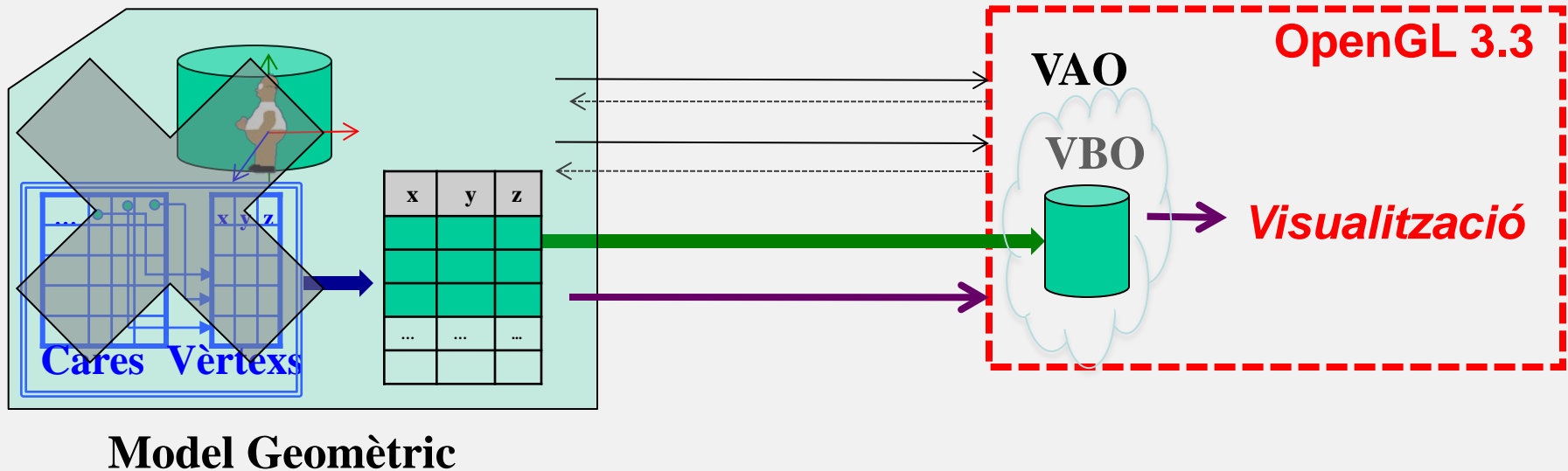
Pintar en OpenGL 3.3: “core” mode

1. Crear en GPU/OpenGL un *VAO* que encapsularà dades del model. Crear *VBO* que guardarà les coordenades dels vèrtexs (potser cal altres per normal, color,...)
2. Guardar llista de vèrtexs (amb repetició) en el *VBO* (i si cal, color i normal en els seus *VBO*)
3. Cada cop que es requereix pintar, indicar el *VAO* a pintar i dir que es pinti: *glDrawArrays(...)*. Acció *pinta_model()* a teoria.



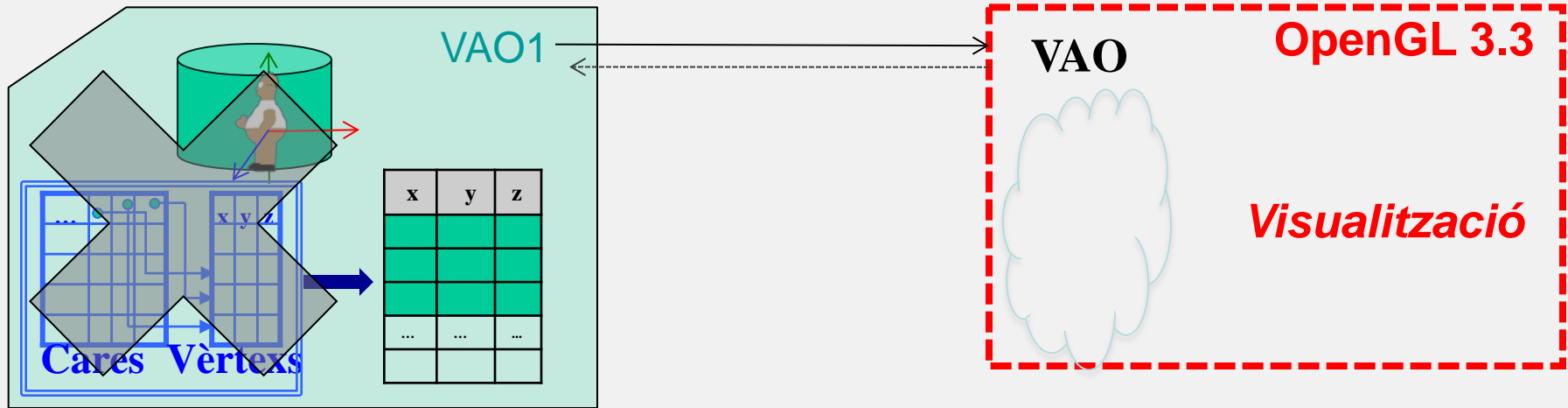
Pintar en OpenGL 3.3: “core” mode

1. Crear en GPU/OpenGL un *VAO* que encapsularà dades del model. Crear *VBO* que guardarà les coordenades dels vèrtexs (potser cal altres per normal, color,...)
2. Guardar llista de vèrtexs (amb repetició) en el *VBO* (i si cal, color i normal en els seus *VBO*)
3. Cada cop que es requereix pintar, indicar el *VAO* a pintar i dir que es pinti: *glDrawArrays(...)*. Acció *pinta_model()* a teoria.



Pintar en OpenGL 3.3: “core” mode

1. Crear en GPU/OpenGL un VAO



Model Geomètric

```
GLuint VAO1; // variable on guardarem l'identificador del VAO
```

```
glGenVertexArrays (1, &VAO1); // generació de l'identificador
```

```
glBindVertexArray (VAO1); // activació del VAO
```

Informació del model

Per a generar un VAO, descripció de les crides:

`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

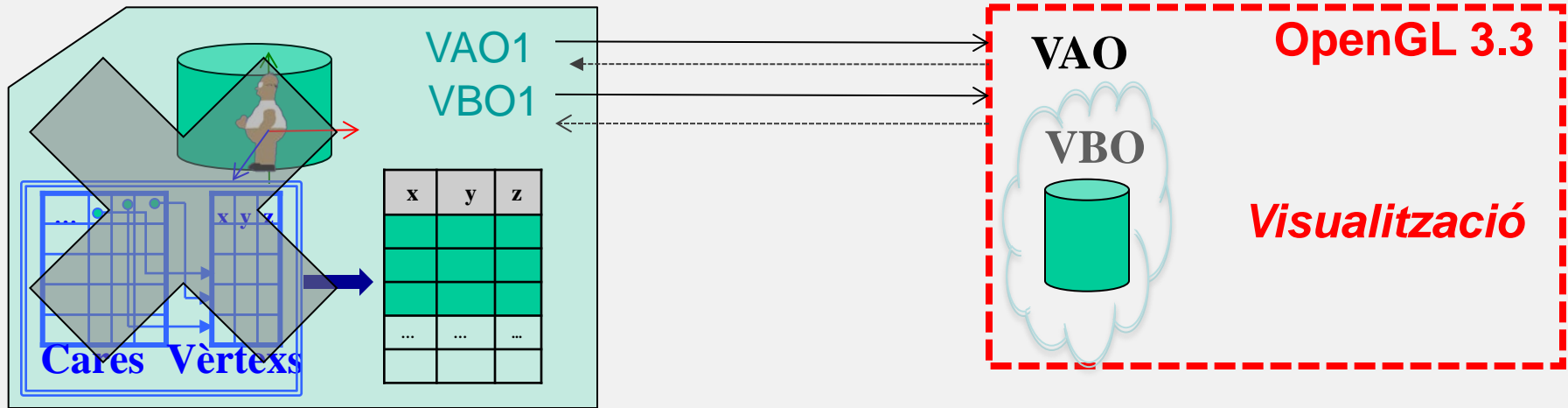
`void glBindVertexArray (GLuint array);`

Activa el VAO identificat per *array*

array : nom del VAO a activar

Pintar en OpenGL 3.3: “core” mode

1. Crear en GPU/OpenGL un *VAO*. Crear *VBO*.



Model Geomètric

`GLuint VBO1; // variable on guardarem l'identificador del VBO`

`glGenBuffers (1, &VBO1); // generació de l'identificador`

`glBindBuffer (GL_ARRAY_BUFFER, VBO1); // activació del VBO`

Informació del model

Per a generar un VBO, descripció de les crides:

`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);`

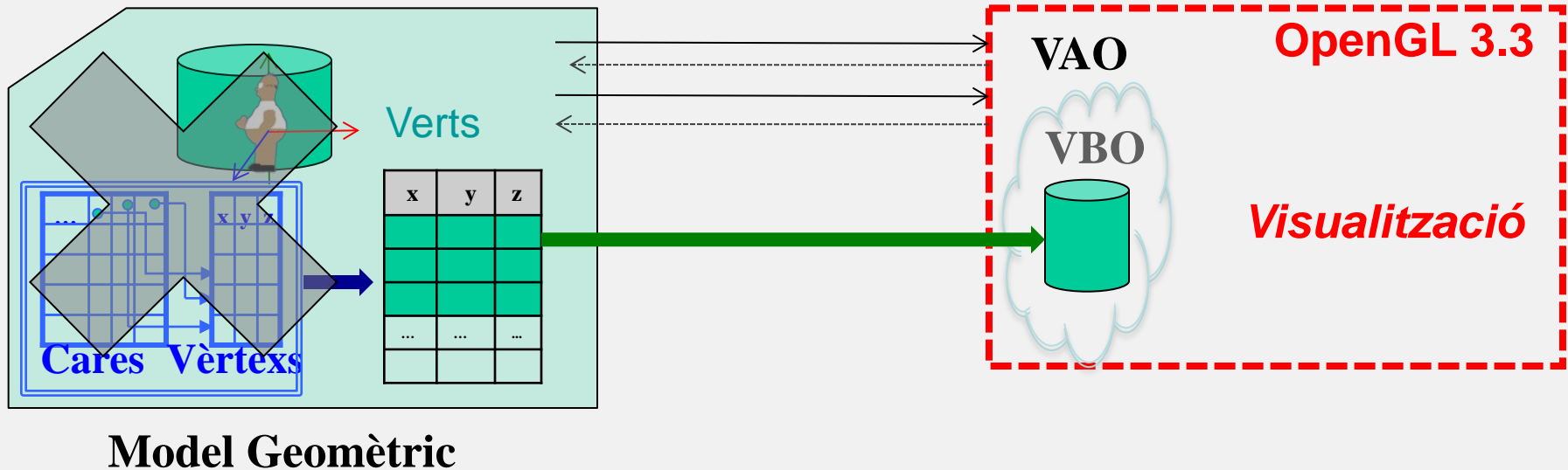
Activa el VBO identificat per *buffer*

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

buffer : nom del VBO a activar

Pintar en OpenGL 3.3: “core” mode

1. Crear en GPU/OpenGL un *VAO*. Crear *VBO*.
2. Guardar llista de vèrtexs (amb repetició) en el *VBO*



```
glBufferData (GL_ARRAY_BUFFER, sizeof (Verts), Verts, GL_STATIC_DRAW);
```

Informació del model

Per a omplir les dades d'un VBO:

```
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, ...)

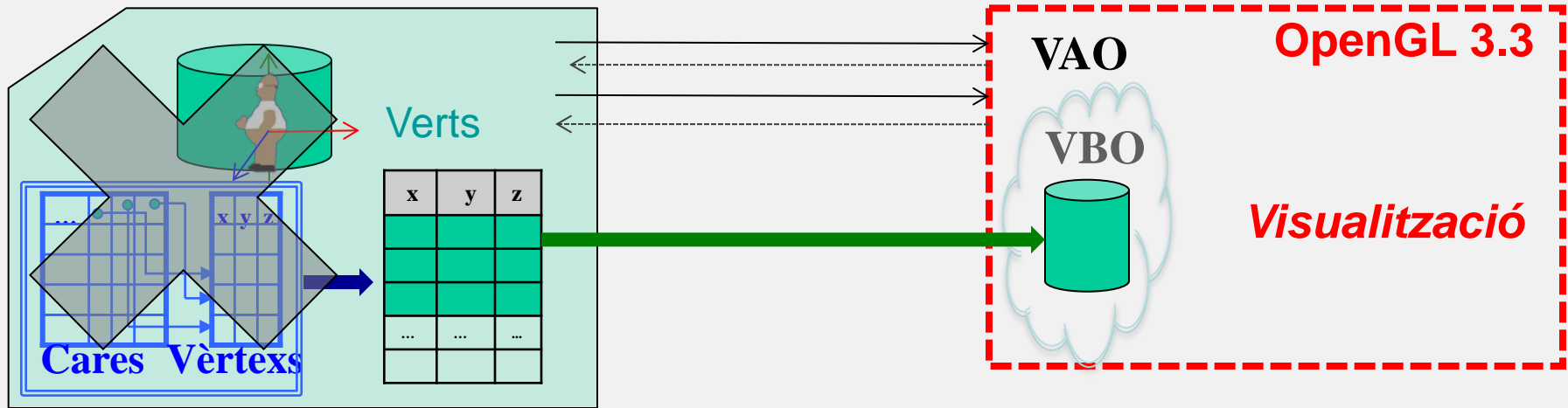
size : mida en bytes de les dades

data : apuntador a les dades

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

Pintar en OpenGL 3.3: “core” mode

1. Crear en GPU/OpenGL un *VAO*. Crear *VBO*.
2. Guardar llista de vèrtexs (amb repetició) en el *VBO*



Model Geomètric

// Cal indicar a la GPU com ha d'interpretar les dades que li hem passat (Verts)

```
glVertexAttribPointer (vertexLoc, 3, GL_FLOAT, GL_FALSE, 0, 0);  
glEnableVertexAttribArray (vertexLoc);
```

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 componen 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)

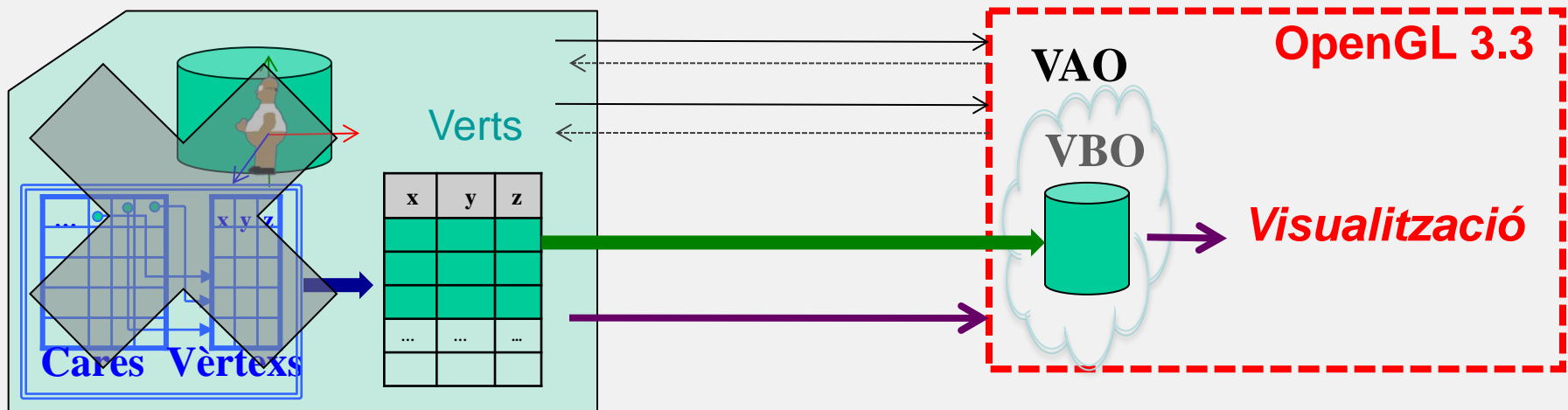
```
void glEnableVertexAttribArray (GLuint index);
```

Activa l'atribut del vèrtex identificat per *index*

index : nom de l'atribut a activar

Pintar en OpenGL 3.3: “core” mode

1. Crear en GPU/OpenGL un *VAO*. Crear *VBO*.
2. Guardar llista de vèrtexs (amb repetició) en el *VBO*
3. Per pintar: Indicar el *VAO* a pintar i dir que es pinti



Model Geomètric

```
glBindVertexArray (VAO1);  
glDrawArrays (GL_TRIANGLES, 0, 3);
```

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);`

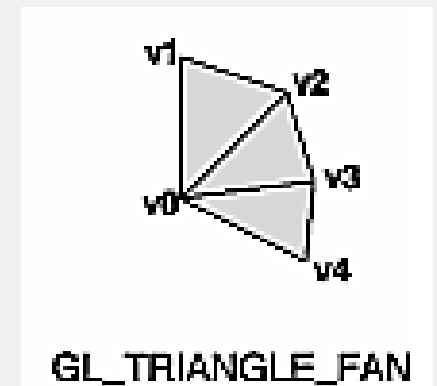
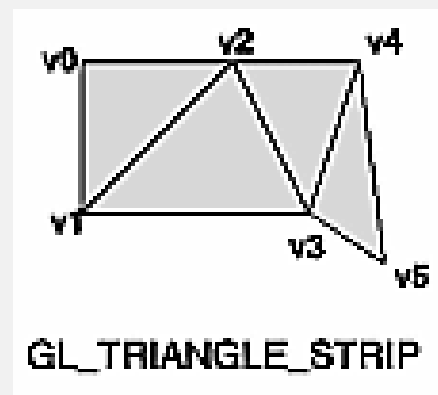
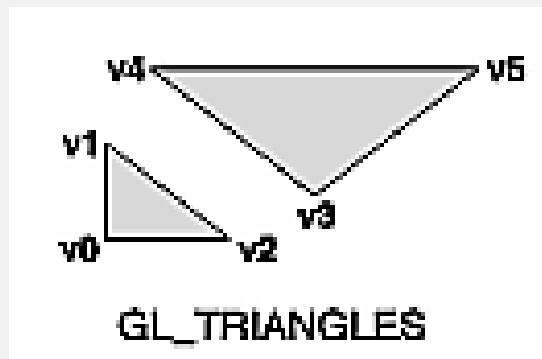
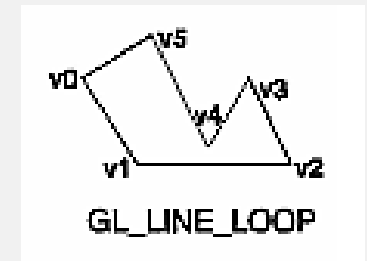
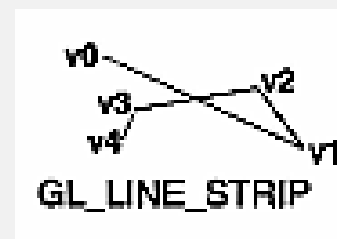
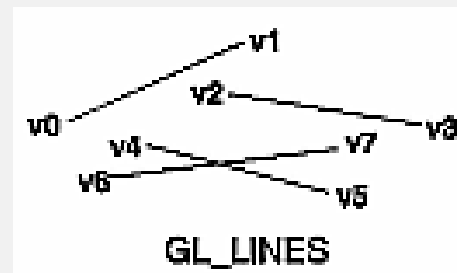
mode : tipus de primitiva a pintar (GL_TRIANGLES, ...)

first : índex del primer element de l'array

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

Primitives en OpenGL

- Totes les primitives s'especificuen mitjançant vèrtexs:



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

Classe que engloba la interfície

MyForm.h

MyForm.cpp

MyGLWidget.h

MyGLWidget.cpp

Exemple complet:

Bloc1_exemple.pro

TEMPLATE = app

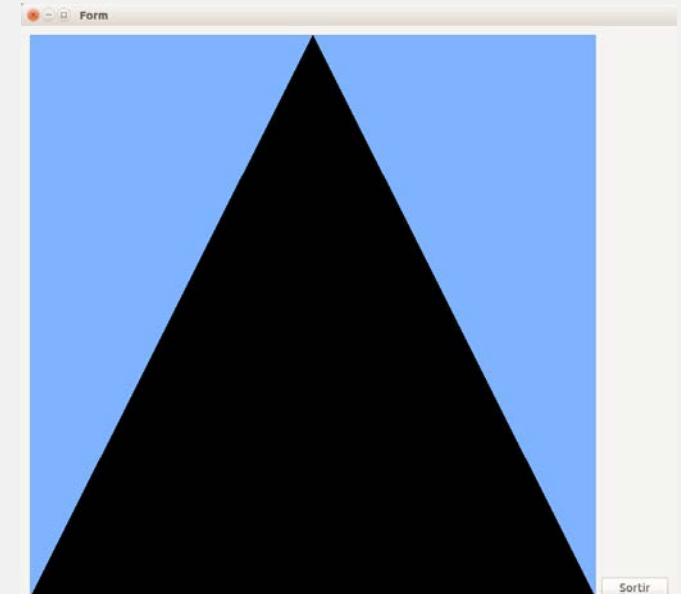
QT += opengl

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);
```

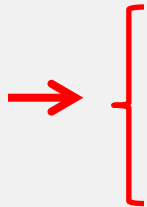
```
    QSurfaceFormat::setDefaultFormat (f);
```

```
    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 VAO1, 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 VAO1, 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 VAO1, vertexLoc;
};
```

Exemple complet:

MyGLWidget.cpp (1)

```
#include "MyGLWidget.h"
```

```
MyGLWidget::MyGLWidget (QWidget* parent) : QOpenGLWidget (parent), program(NULL)
{
    setFocusPolicy(Qt::StrongFocus); // 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, &VAO1);
    glBindVertexArray(VAO1);
    // Creació del buffer amb les dades dels vèrtexs
    GLuint VBO1;
    glGenBuffers(1, &VBO1);
    glBindBuffer(GL_ARRAY_BUFFER, VBO1);
    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

    glViewport (0, 0, width(), height()); // Aquesta crida no cal, Qt la fa de forma automàtica

    // Activem l'Array a pintar
    glBindVertexArray(VAO1);
    // Pintem l'escena
    glDrawArrays(GL_TRIANGLES, 0, 3);
    // Desactivem el VAO
    glBindVertexArray(0);
}

void MyGLWidget::resizeGL (int w, int h)
{
    // Aquí anirà el codi que cal fer quan es redimensiona la finestra
}
```

Exercicis sessió 1.1

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).
 - 3) Fes que pinti una caseta (3 triangles).
 - 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.
 - 5) Juga amb la crida a `glViewport(...)`: Primer viewport més petit i després dos viewports.

Exercicis sessió 1.1

