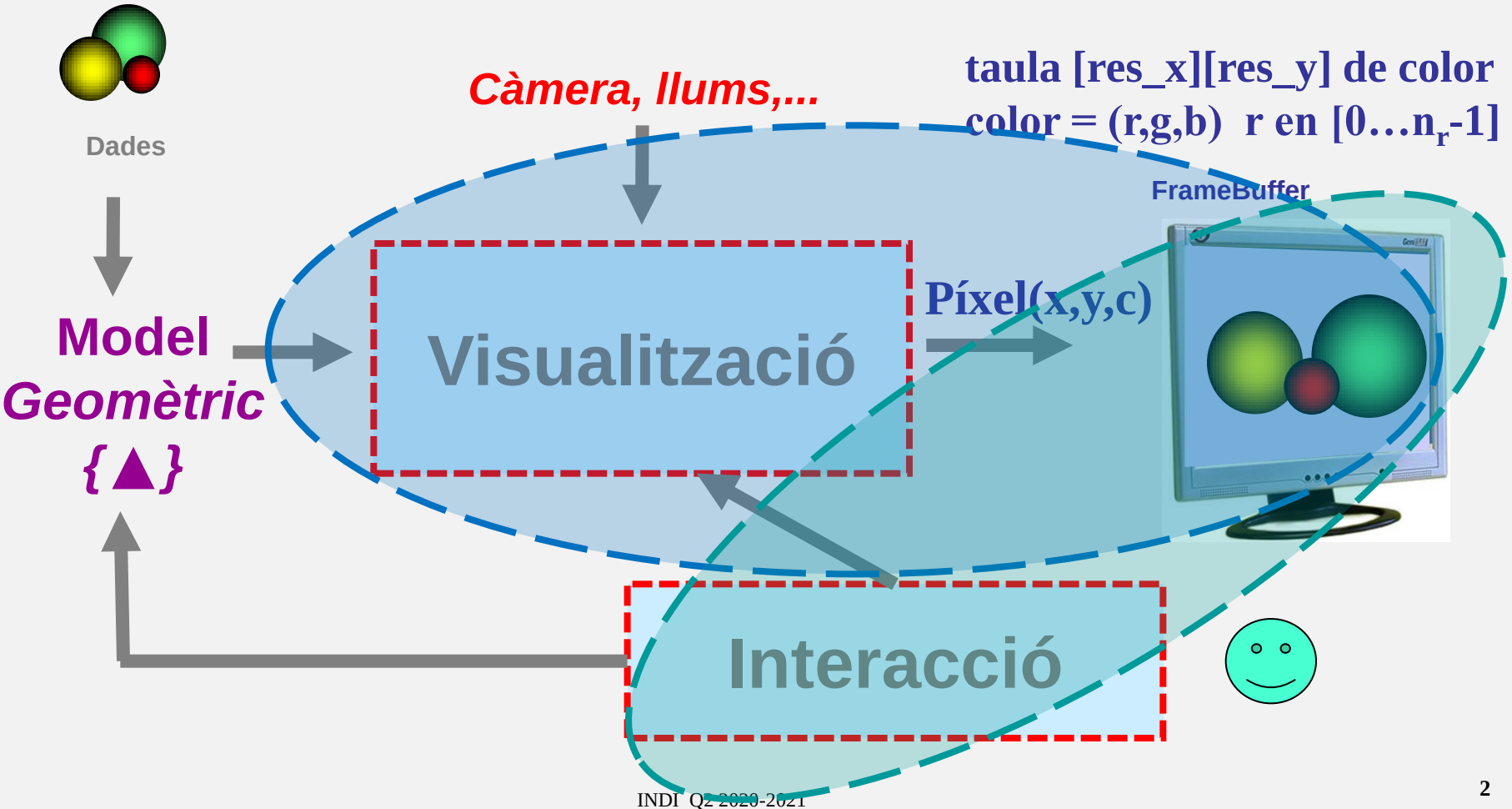
Laboratori OpenGL – Sessió 1.1

- 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

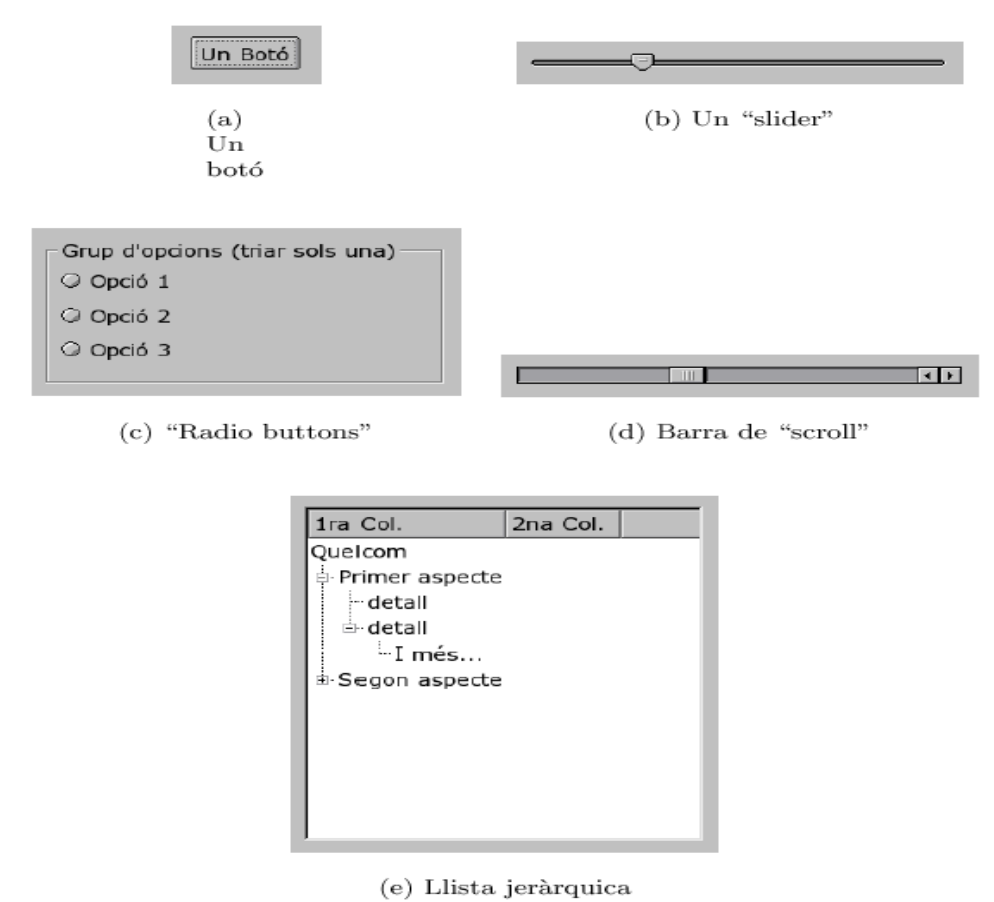
Introducció a OpenGL i Qt

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



Llibreria Qt

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



Projecte Qt
main.cpp

Defineix els components de l'aplicació

Bloc1_exemple.pro

Disseny de la interfície

MyForm.ui

Programa principal

main.cpp

Classe que engloba la interfície

MyForm.h

MyForm.cpp

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 += .
  - FORMS +- MyForm.ui
  - #Input
  - SOURCES += main.cpp MyForm.cpp
- Compilem i enllacem
  - qmake (al laboratori cal fer **qmake-qt5**)
  - make
- Executable anomenat *helloQt* en el directori on estiguem.
- Executar-lo amb:
  - ./helloQt

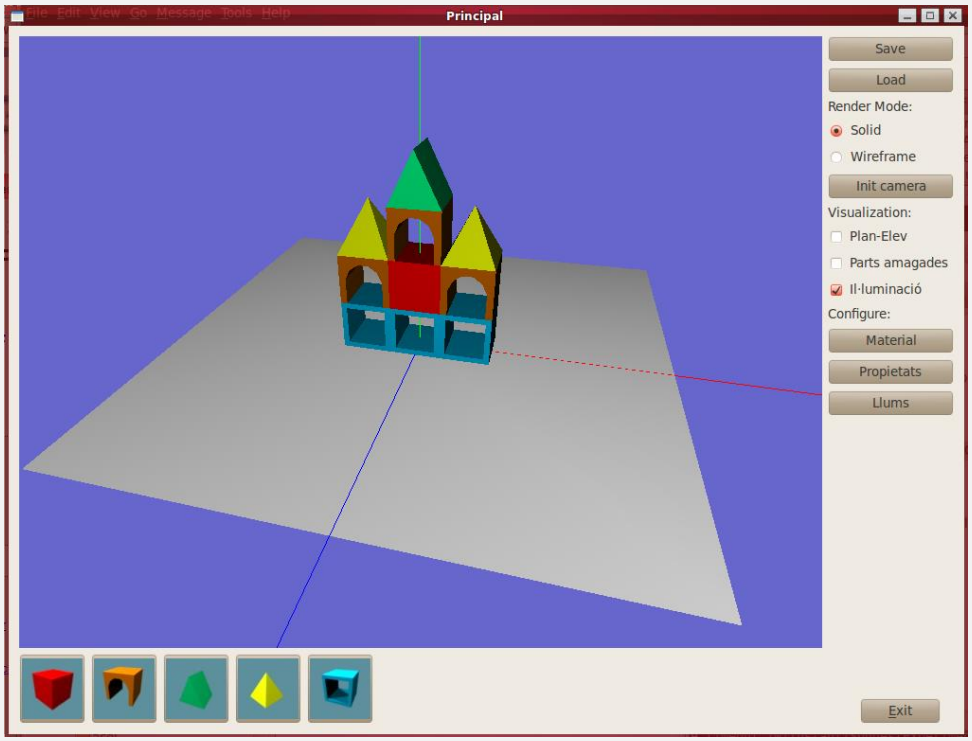
Introducció a OpenGL

- 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
  - OpenGL és una Màquina d'estats

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



Projecte Qt usant OpenGL

- Donat que farem servir la llibreria glm (OpenGL Mathematics), cal descarregar-se-la i posar-la a una ruta " a prop" dels projectes on treballem.
- Per usar-la només cal tenir al fitxer **.pro** la sentència:
  INCLUDEPATH += {ruta a la llibreria glm}

  (canvieu la ruta relativa/absoluta per la que correspongui en el vostre cas)

A l'esquelet de codi proporcionat l'arxiu **.pro** s'espera trobar el glm a la ruta **/usr/include/glm**. Si no hi és, poseu-li.

OpenGL amb Qt

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

Mètodes virtuals que cal implementar:

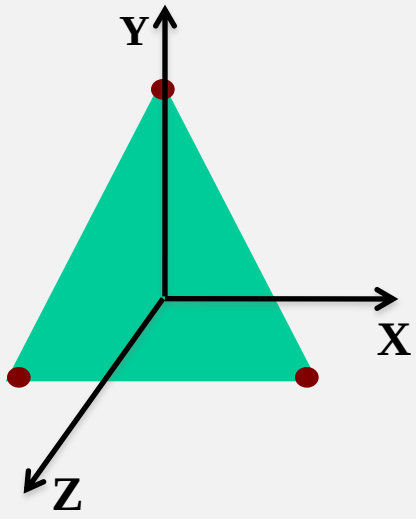
Les funcións ***initializeGL ()***, ***paintGL ()*** **i** ***resizeGL ()***

NO s'han de cridar mai directament.

Si voleu refrescar/repintar l'escena, heu de cridar el mètode ***update();***

- *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 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.

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).

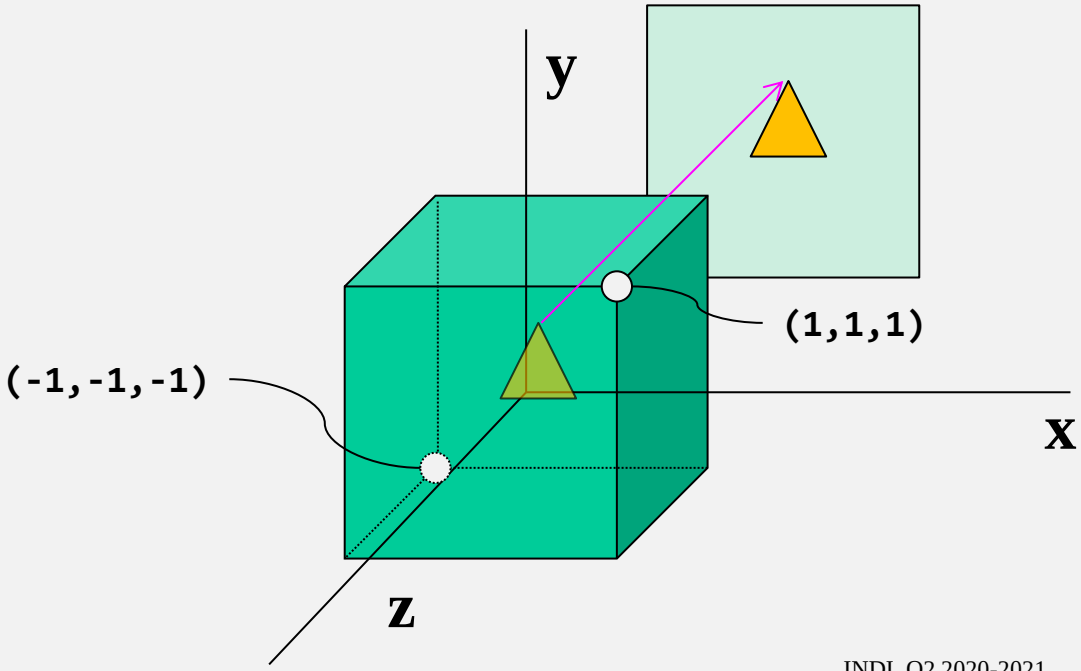
Pintarem els VAOs.

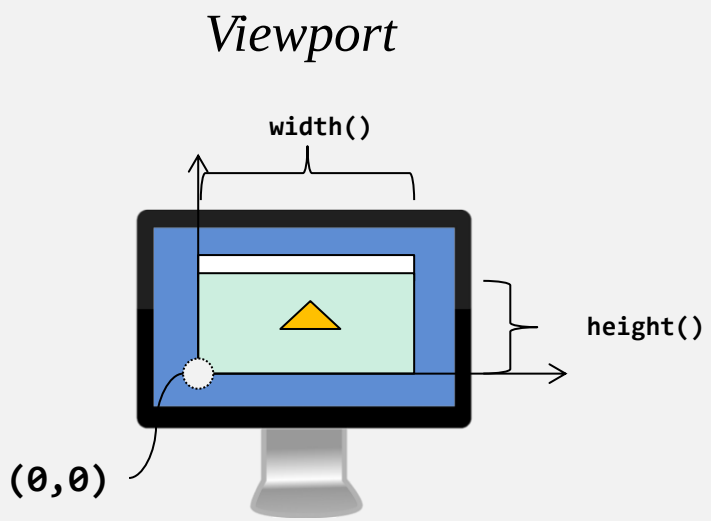
Sistema de coordenades i finestra de visió

• En el sistema de coordenades(SC) inicial l'origen està centrat al *Viewport*. Anomenarem aquest SC el *Clip Space*.

• La mida del nostre univers de visualització és un cub de radi 1 centrat a (0,0,0)

• Veiem en el *Viewport* la projecció plana (ortogonal) del cub contra el pla Z=-1



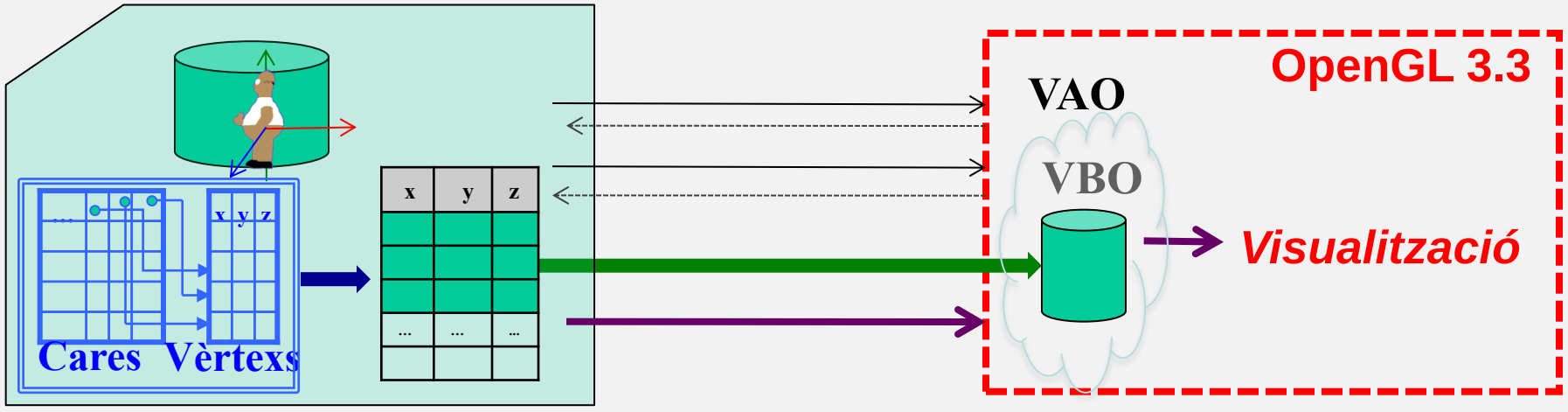


Pintar en OpenGL 3.3: "core" mode

1. **Crear en GPU/OpenGL un *VAO* que encapsularà dades del model.**
2. **Crear VBO que guardarà les coordenades dels vèrtexs (potser cal altres per normal, color,…)**
3. **Guardar llista de vèrtexs (amb repetició o explícit) en el VBO**

(i si cal, color i normal en els seus *VBO, però avui no*)

4. **Cada cop que es requereix pintar, indicar el *VAO* a pintar i dir que es pinti: *glDrawArrays(…)*. Acció pinta_model() a teoria.**



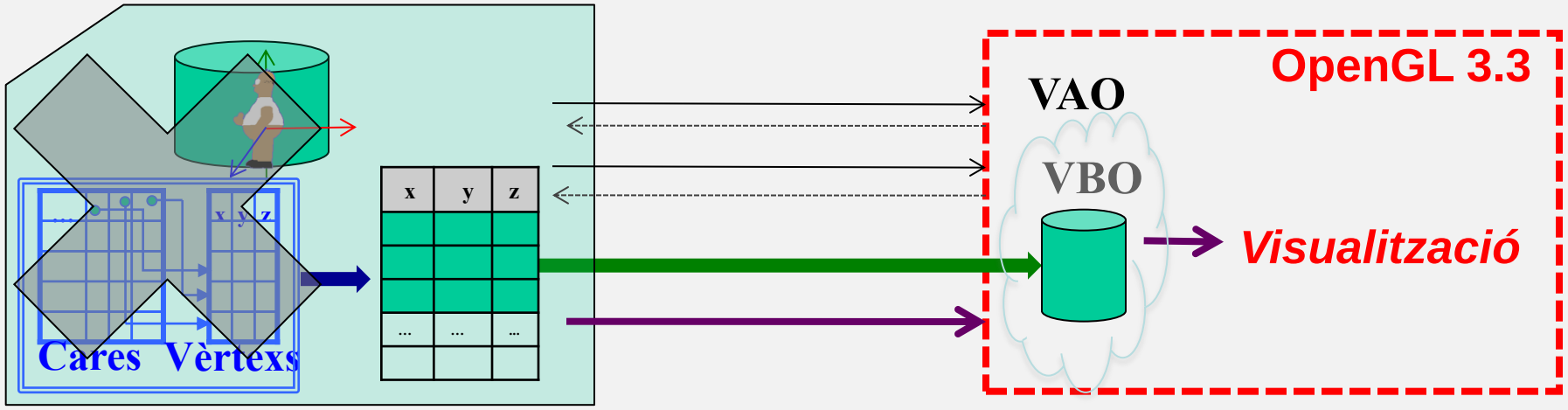
Model Geomètric

Pintar en OpenGL 3.3: "core" mode

1. **Crear en GPU/OpenGL un *VAO* que encapsularà dades del model.**
2. **Crear VBO que guardarà les coordenades dels vèrtexs (potser cal altres per normal, color,…)**
3. **Guardar llista de vèrtexs (amb repetició o explícit) en el VBO**

(i si cal, color i normal en els seus *VBO, però avui no*)

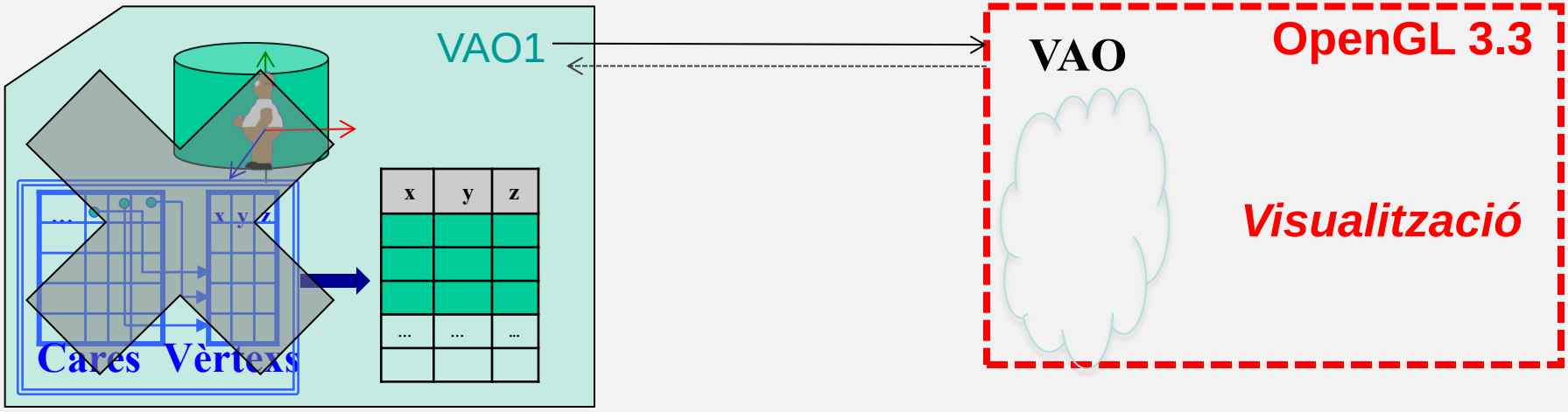
4. **Cada cop que es requereix pintar, indicar el *VAO* a pintar i dir que es pinti: *glDrawArrays(…)*. Acció pinta_model() a teoria.**



Model Geomètric

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

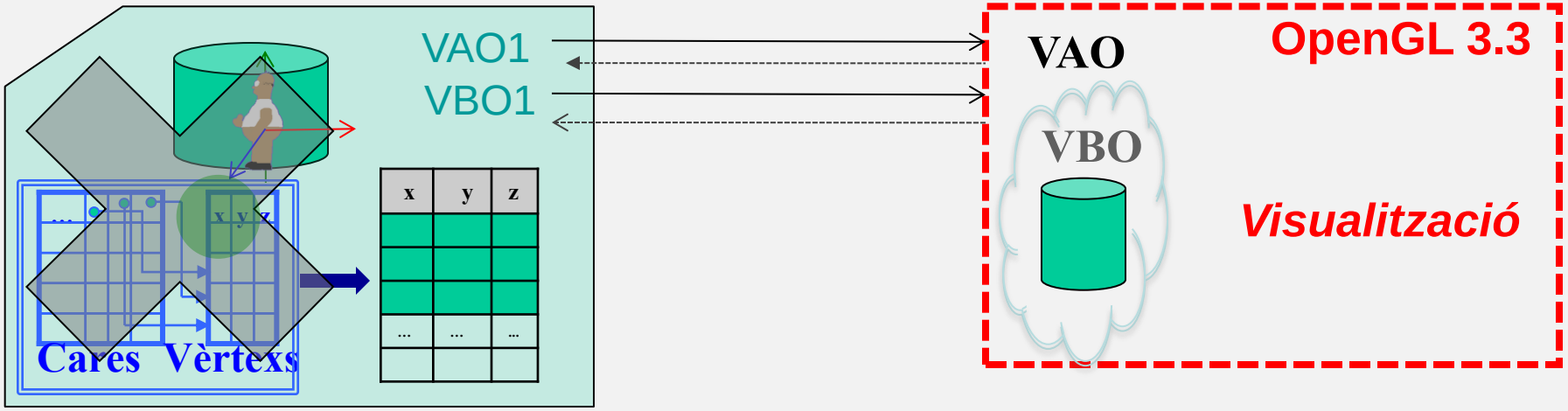
```
void glBindVertexArray (GLuint array);
```

Activa el VAO identificat per *array*

array : nom del VAO a activar

Pintar en OpenGL 3.3: “core” mode

2. Crear VBO que guardarà les coordenades dels vèrtexs



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*);

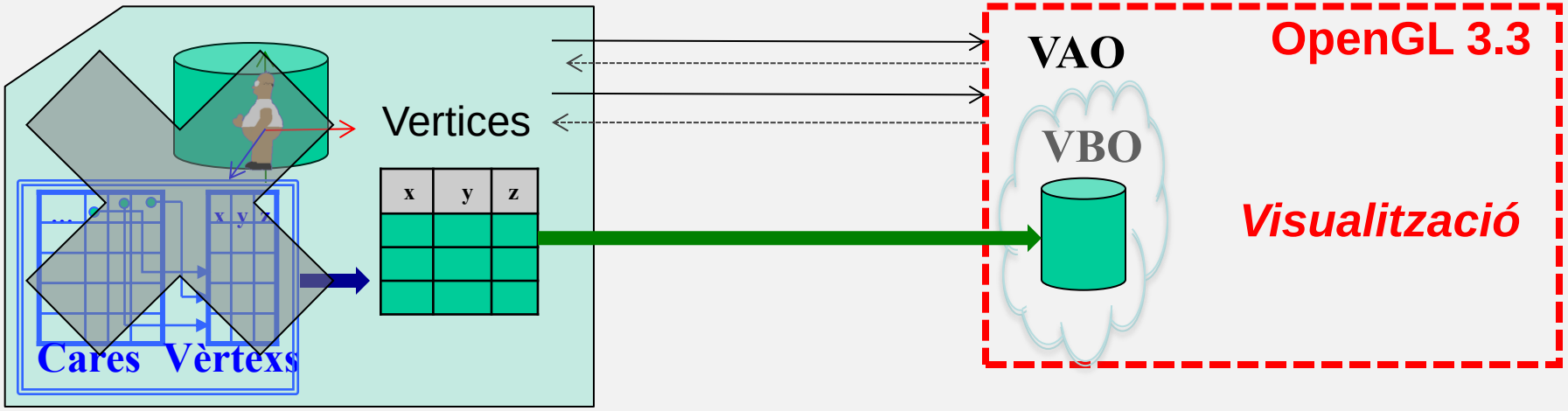
Activa el VBO identificat per *buffer*

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

3. Guardar llista de vèrtexs (amb repetició o explícit) en el VBO



Model Geomètric

```
 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);
...

glBufferData (GL_ARRAY_BUFFER, sizeof(Vertices), Vertices,
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 (veure documentació)

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

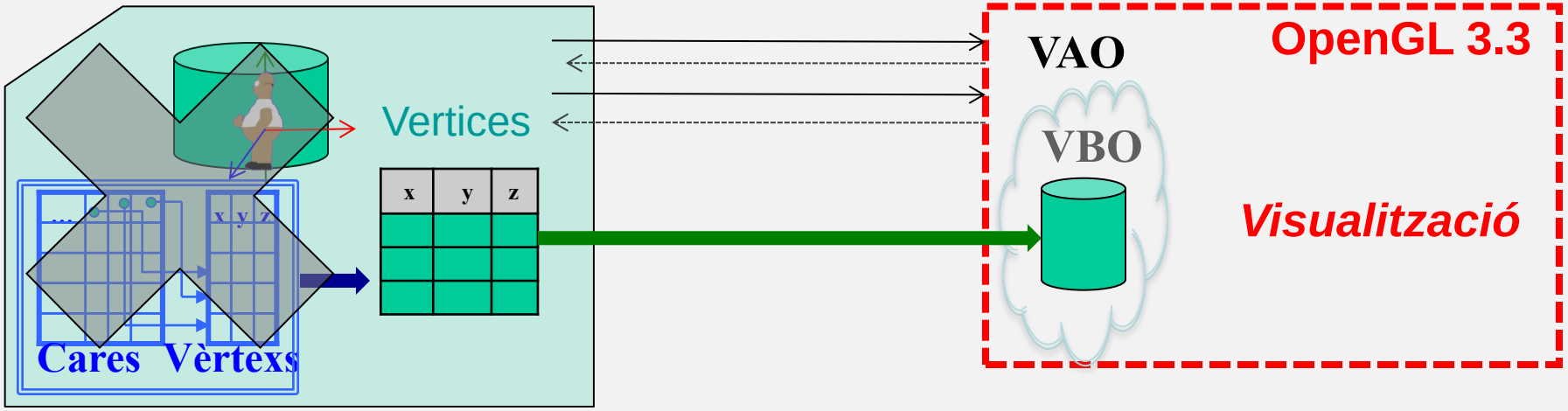
size : mida en bytes de les dades

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

3. Guardar llista de vèrtexs (amb repetició o explícit) en el VBO



Model Geomètric

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

– **Si el VBO conté dades dels vèrtex, normals, materials, etc. barrejades (*interleaved*), entren en joc els paràmetres *stride* i *pointer*.**

```
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

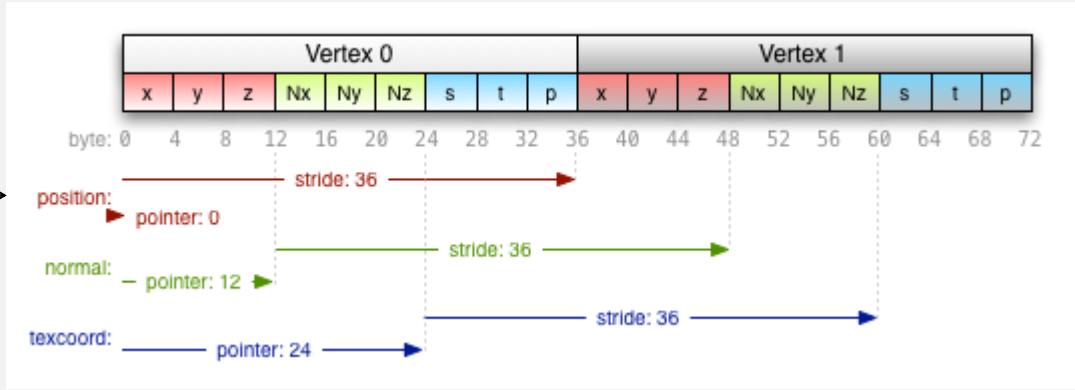
size : nombre de components que composen l'atribut

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

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

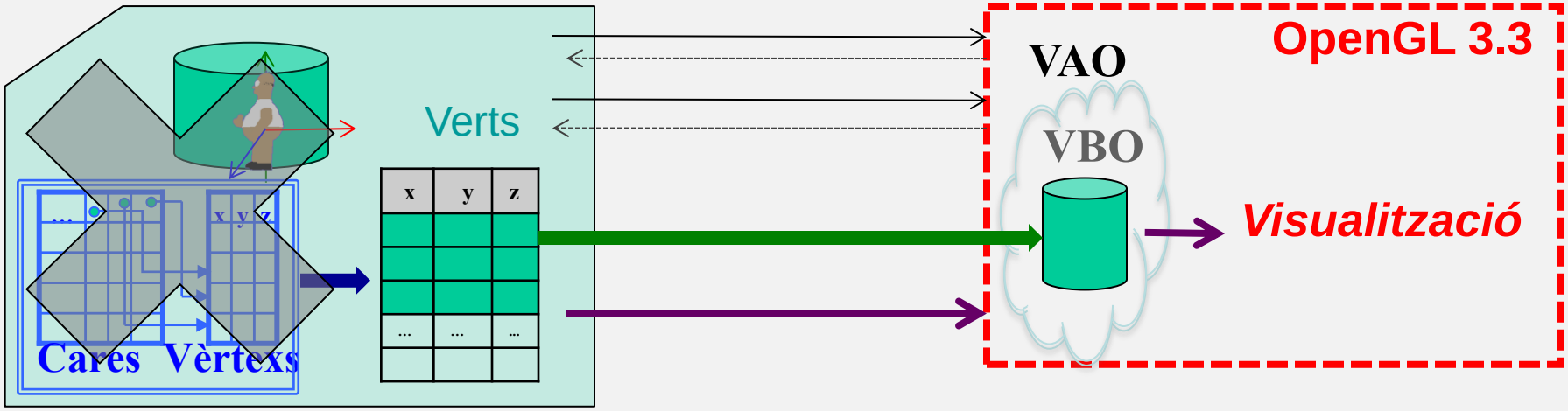
stride : offset en bytes entre dos atributs consecutius (normalment 0)

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



Pintar en OpenGL 3.3: "core" mode

4. **Cada cop que es requereix pintar, indicar el *VAO* a pintar i dir que es pinti: *glDrawArrays(…)*. Acció pinta_model() a teoria.**



Model Geomètric

```
glBindVertexArray (VAO1); // Activa el VAO1
glDrawArrays (GL_TRIANGLES, 0, 3); // Llança a
dibuixar els vèrtex associats al VAO (dins del VBO)
com a llista de triangles
```

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);
```

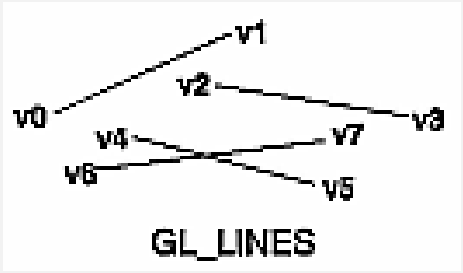
mode : tipus de primitiva a pintar (GL_TRIANGLES, …)

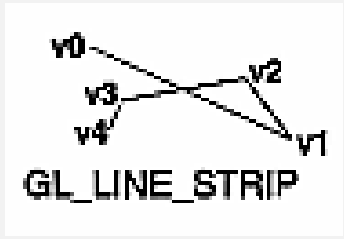
first : índex del primer element de l'array

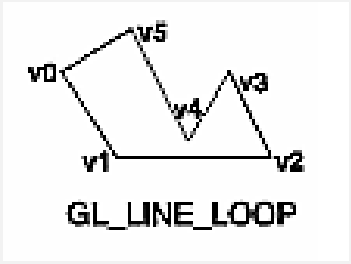
count : nombre de vèrtex a tenir en compte de l'array

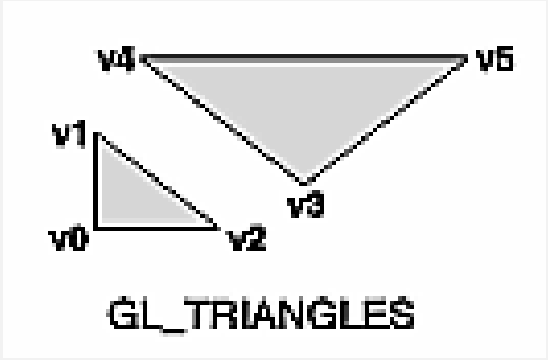
Primitives en OpenGL

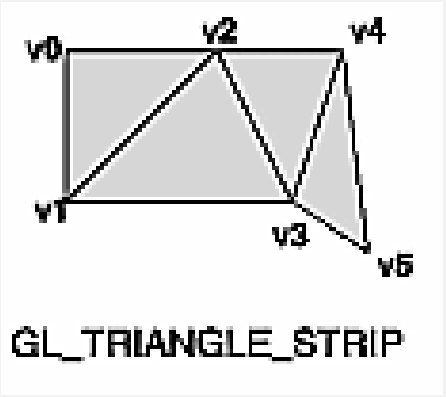
- Totes les primitives s'especifiquen mitjançant vèrtexs:



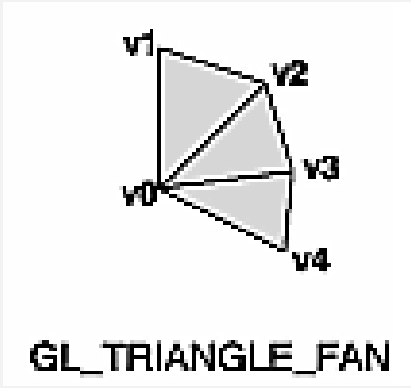




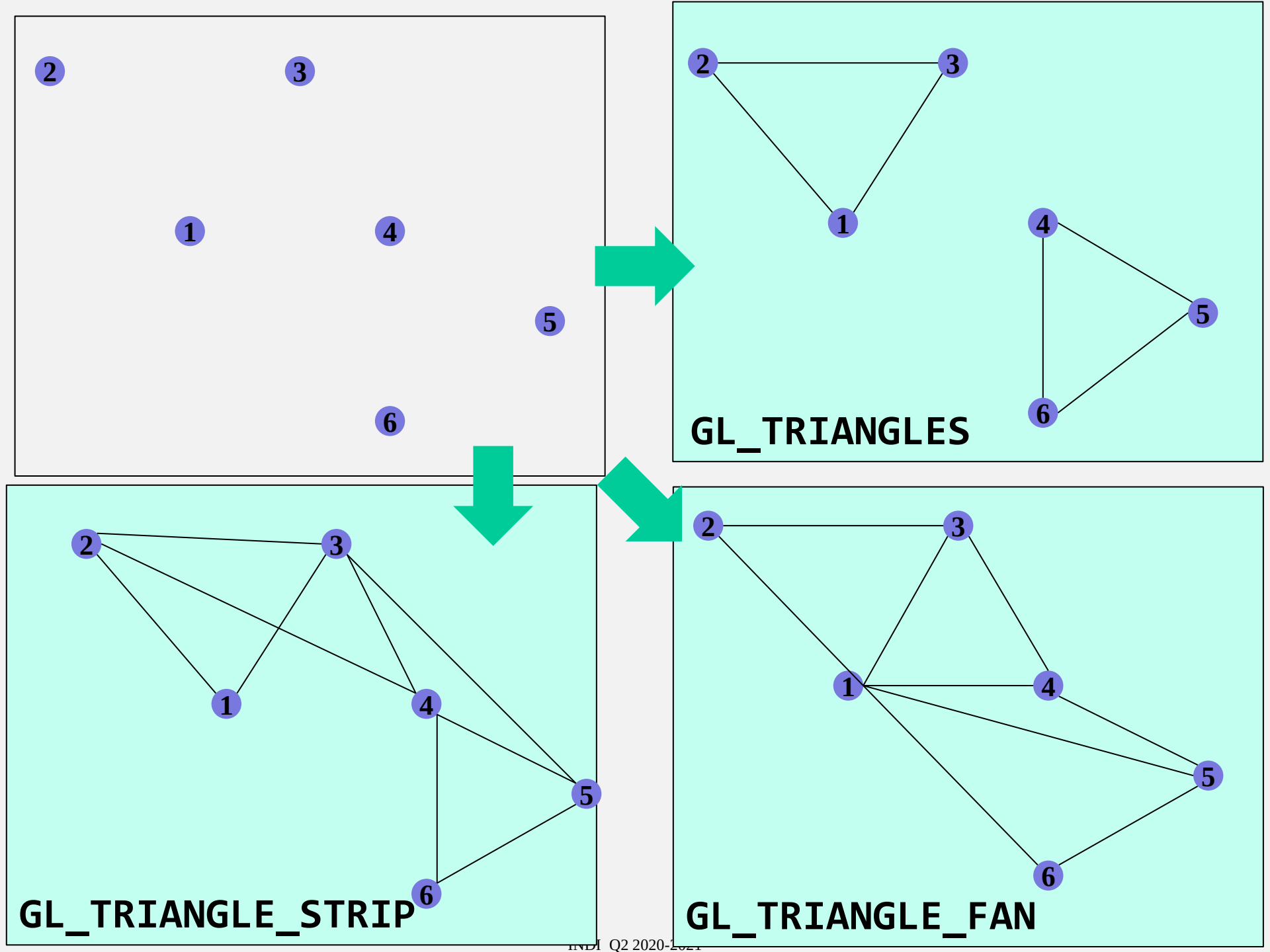




Cada triangle és sempre $V_i - V_{i-1} - V_{i-2}$

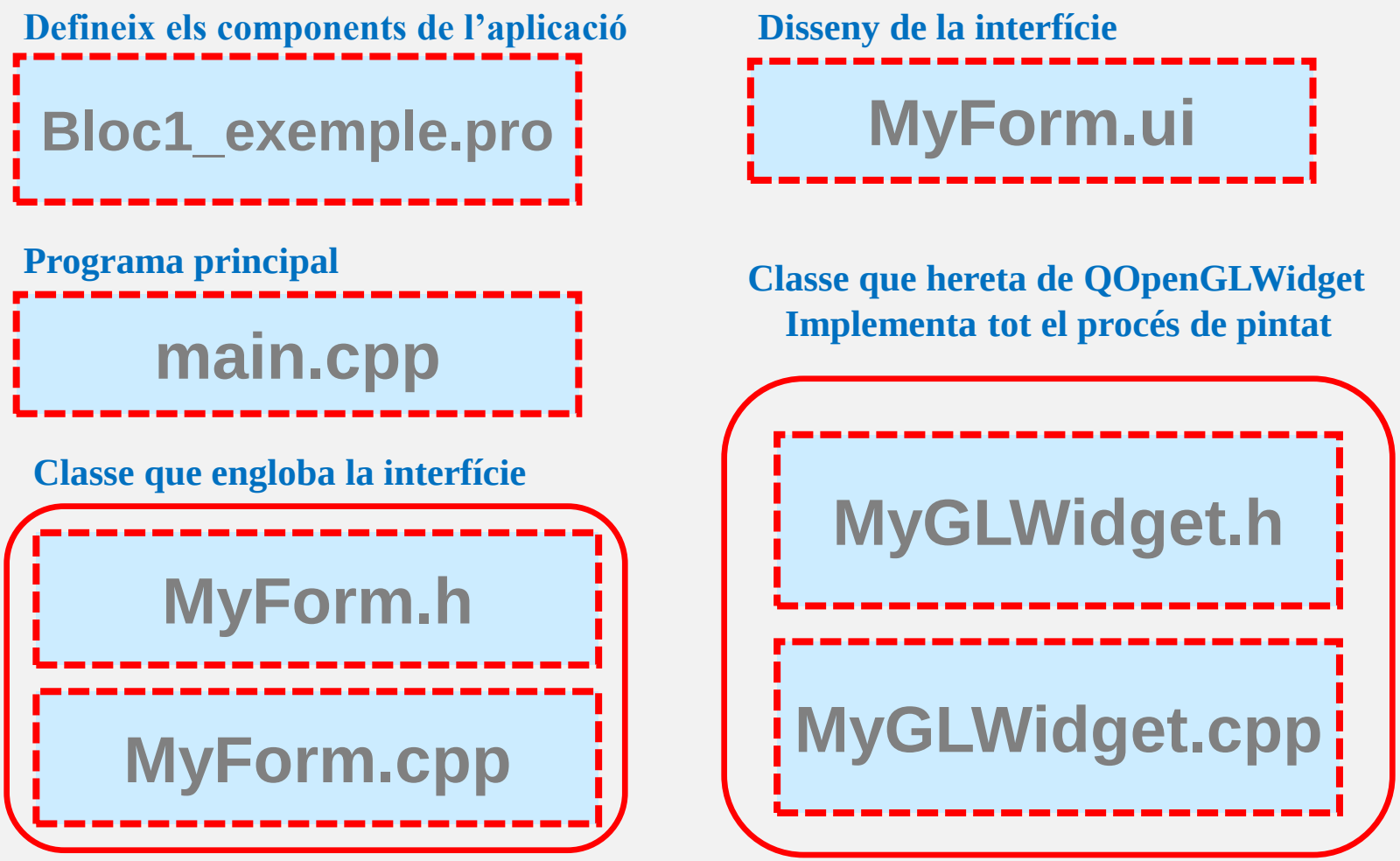


V_0 està a tots els triangles


2
3
1
4
5
6
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN

Exemple complet

- Exemple que teniu a Atenea:



Exemple complet:
Bloc1_exemple.pro

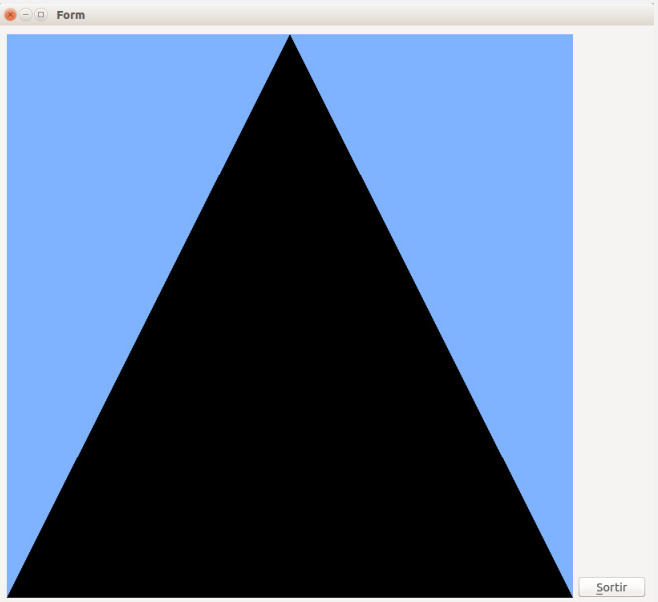
```
TEMPLATE = app
QT += opengl

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 (); //  Creació dels buffers de l'escena
        ……. //  ho explicarem el proper dia
    GLuint VAO1, vertexLoc; //  Identificadors del VAO i de la variable de vertex
};
```

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; // alliberar recursos
}

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(); // preparar els buffers de l'escena

}
```

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
}
```

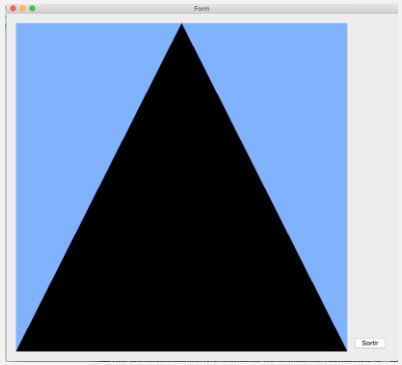
Exercicis sessió 1.1

El que cal que feu en aquesta sessió és:

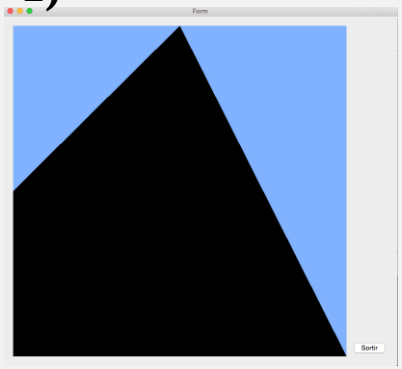
1) Copieu-vos l'exemple, compileu-lo i proveu-lo. **Compte amb el path al glm** !
2) Feu els exercicis que teniu al guió per a aquesta sessió:
    1) Jugueu amb les coordenades dels vèrtexs, tingueu en compte que el món que estem veient és aquell en què x, y, i z pertanyen a [-1, 1].
    2) Feu que pinti un quadrat (usant triangles).
    3) Feu que pinti una caseta (3 triangles).
    4) Pinteu 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

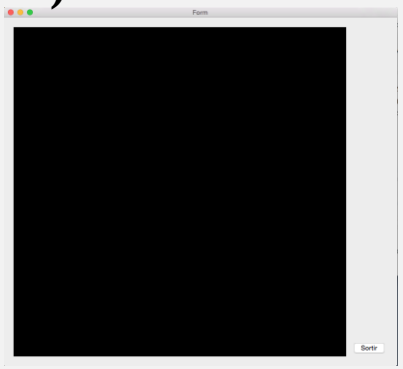
Punt de partida



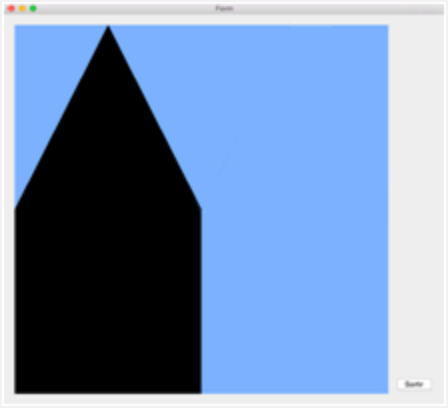
1)



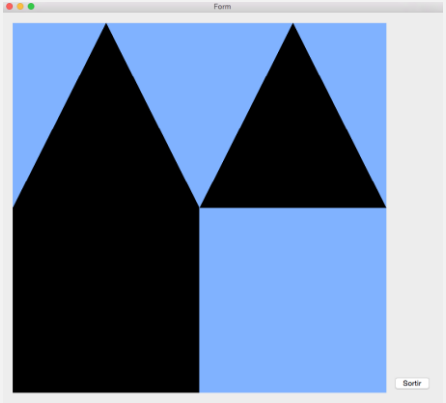
2)



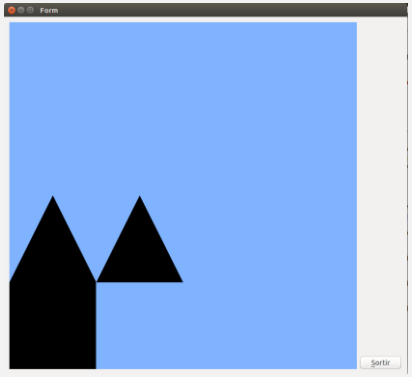
3)



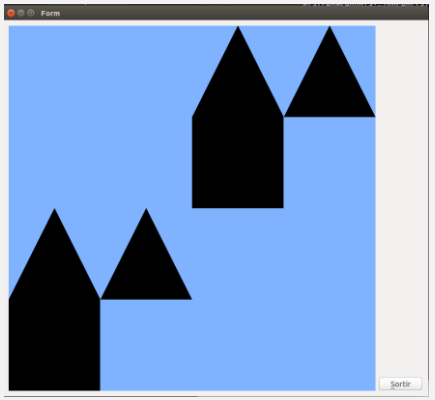
4)



5.a)



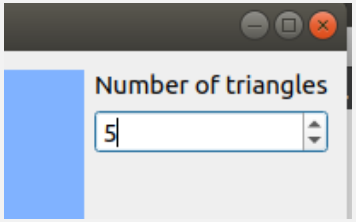
5.b)

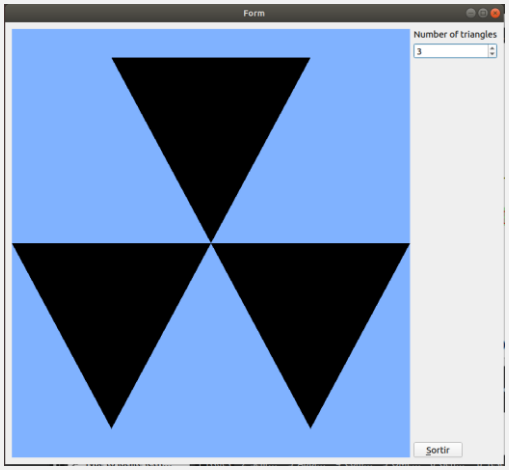


Exercicis sessió 1.1

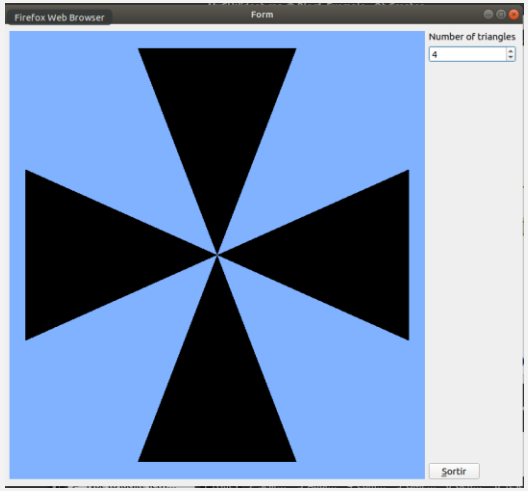
(bonus track)

Una mica d'OpenGL i de QT. Useu un QSpinBox per controlar l'aspecte de la figura que renderitzem tal i com es mostra a les figures:

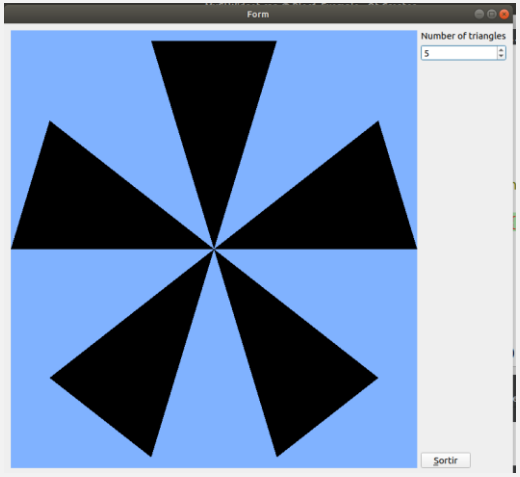




N=3



N=4



N=5

Exercicis sessió 1.1

(bonus track)

Una mica d'OpenGL i de QT. Useu un QSpinBox per controlar l'aspecte de la figura que renderitzem tal i com es mostra a les figures.

a) **Definiu un SLOT a *MyGLWidget.h* per rebre el signal *valueChanged* del *QSpinBox***

```
public slots:
    void onValueChanged(int pValor);
```

b) **Implementeu l'slot a MyGLWidget.cpp. Actualitzeu el valor de la variable que controla el nombre de costats i demaneu que es redibuxi el widget:**

```
void MyGLWidget::onValueChanged (int pValor)
{
    this->val = pValor; // ens desem el valor a un atribut de la classe
    qDebug()<<"DIAL!"<<pValor;
    if(this->isInitialized()){
        makeCurrent();
        createBuffers(); // actualitza buffers
        update(); // actualitza interfície (repinta)
    }
}
```

Exercicis sessió 1.1

(bonus track)

Finalment, connecteu l'slot al signal valueChanged(int) del *QSpinBox*: