Laboratori OpenGL – Sessió 2.1
Bloc 2

- Nou esquelet de base (~/assig/idi/blocs/bloc-2)
- Transformacions de càmera amb glm (view i projection)
- Classe Model – càrrega d'objectes OBJ
- Z-buffer
- Control d'errors d'OpenGL

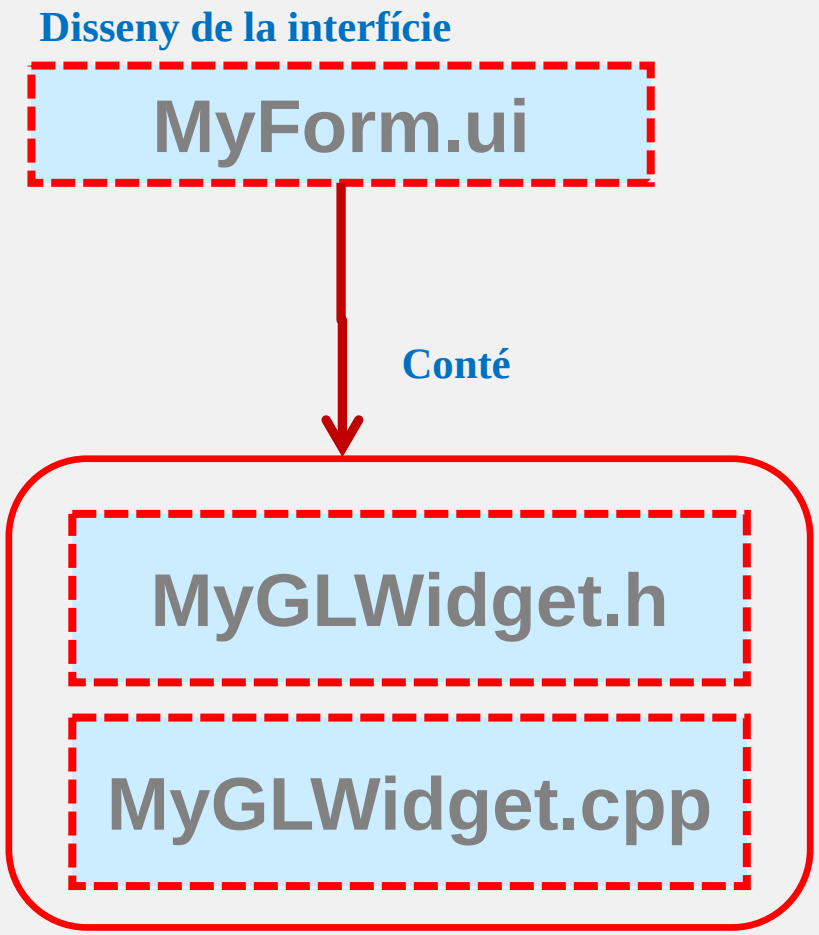
Nou esquelet de base

- Classe BL2GLWidget:
  - Hereta de QOpenGLWidget
  - Inclou el codi ja implementat per l'esquelet
  - No es pot modificar!
- Classe MyGLWidget:
  - Hereta de BL2GLWidget
  - Pot reimplementar mètodes virtuals de BL2GLWidget
  - Pot accedir als atributs i mètodes protected de BL2GLWidget

Classes BL2GLWidget i MyGLWidget

Classes BL2GLWidget i MyGLWidget

- La interfície de l'aplicació de Qt (MyForm.ui) conté un element de tipus MyGLWidget
- Quan Qt inicia l'aplicació crida als mètodes initializeGL() i resizeGL(...) de MyGLWidget
- initializeGL(), paintGL() i resizeGL() no estan implementats, per ara, en MyGLWidget, s'executen els de BL2GLWidget.



Nou esquelet:
BL2GLWidget.h

```
…….
class BL2GLWidget : public QOpenGLWidget, protected QOpenGLFunctions_3_3_Core
{
  Q_OBJECT
  public:
    BL2GLWidget (QWidget *parent=0);
    ~BL2GLWidget ();
  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
    …….   // Tots els mètodes virtuals són susceptibles de ser reimplementats en la
                classe derivada

    QOpenGLShaderProgram *program;   // Els atributs protected es veuen des de la
    glm::mat4 View, legoTG;                    // classe derivada
private:
    …….  // Tot el que és privat no es veu des de la classe derivada
};
```

Nou esquelet:
MyGLWidget.h

```
#include “BL2GLWidget.h"

Class MyGLWidget : public BL2GLWidget
{
    Q_OBJECT
    public:
       MyGLWidget (QWidget *parent=0) : BL2GLWidget(parent) {}
       ~MyGLWidget ();

     protected:
        …     // mètodes que calgui reimplementar en MyGLWidget

    private:
        int printOglError(const char file[], int line, const char func[]);    // mètode privat
};
```

Nou esquelet:
MyGLWidget.cpp (1)

```
#include “MyGLWidget.h"
…
#define printOpenGLError() printOglError(__FILE__, __LINE__)
#define CHECK() printOglError(__FILE__, __LINE__, __FUNCTION__)
#define DEBUG() std::cout << __FILE__ << “ “ << __LINE__ << “ “ << __FUNCTION__ << std::endl;

// Mètode per al control d’errors d’OpenGL
void MyGLWidget::printOglError (const char file[], int line, const char func[])
{
    …   // implementa el necessari per facilitar el control d’errors d’OpenGL
}

MyGLWidget::~MyGLWidget ()  {
}
```

Nou esquelet:
Com implementar a MyGLWidget

Opcions:

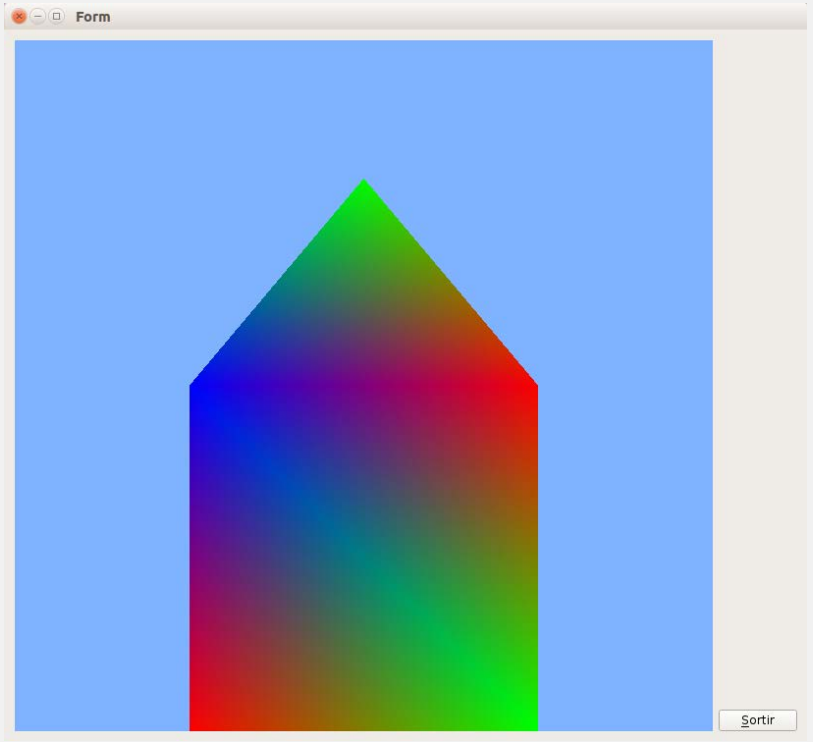
a) Mètode nou de MyGLWidget (no existeix en BL2GLWidget):
   - Declarar i implementar el mètode com sempre

b) Mètode que existeix en BL2GLWidget i que es vol reescriure sencer:
   - Declarar-lo com a virtual a MyGLWidget i implementar-lo

c) Mètode que existeix en BL2GLWidget i que es vol ampliar per davant o per darrera (no es pot modificar pel mig):
   - Declarar-lo com a virtual a MyGLWidget
   - En la implementació fer crida al mètode de la classe BL2GLWidget abans o després del codi afegit (depenent si es vol ampliar per darrera o per davant respectivament).

Què fa l'esquelet del Bloc 2?

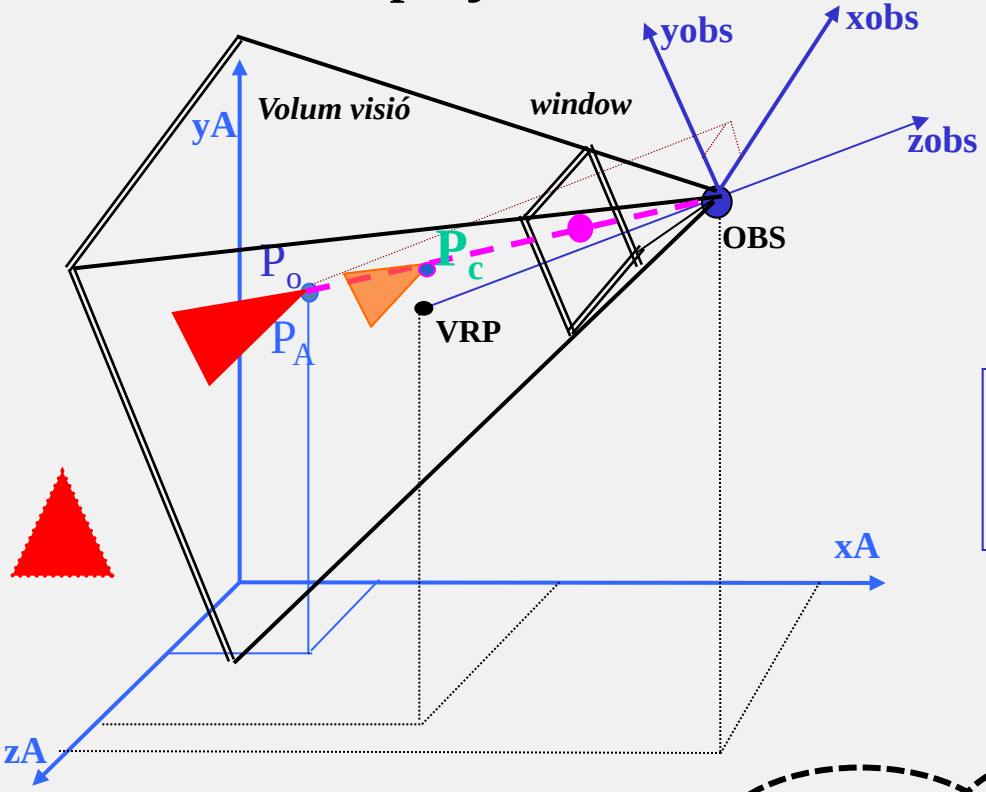
- Pinta un objecte
- Inclou una transformació de model
- El Vertex i Fragment Shaders pinten amb color per vèrtex



Laboratori OpenGL – Sessió 2.1
Bloc 2

- Nou esquelet de base (~/assig/idi/blocs/bloc-2)
- Transformacions de càmera amb glm (view i projection)
- Classe Model – càrrega d'objectes OBJ
- Z-buffer
- Control d'errors d'OpenGL

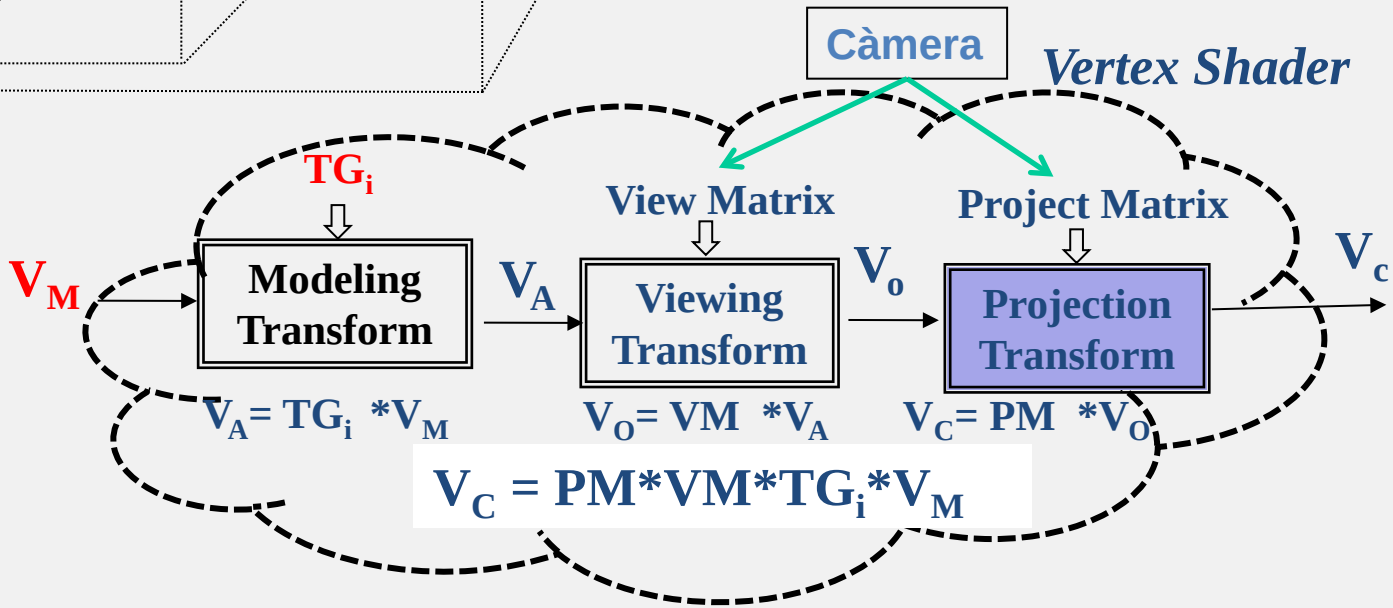
Transformació de projecció



FOV, ra_w, zNear, zFar

PM = perspective (FOV, ra_w, zN, zF)

projectMatrix(PM);



Transformació de projecció
(exercici 1)

- Al codi de MyGLWidget cal:
  1. Demanar un uniform location per al uniform de la matriu

     projLoc = glGetUniformLocation (program->programId(), “proj”)

Ho afegirem al mètode (ja impementat a BL2GLWidget): carregaShaders

```
void MyGLWidget::carregaShaders() {  // declarem-lo també en MyGLWidget.h
    BL2GLWidget::carregaShaders();  // cridem primer al mètode de BL2GLWidget
    projLoc = glGetUniformLocation (program->programId(), “proj”);
}
```

Transformació de projecció
(exercici 1)

- Al codi de MyGLWidget cal:

2. Definir un mètode que ens calculi la transformació de projecció i enviï el uniform amb la matriu cap al vertex shader (cal que els paràmetres siguin floats)

```
void MyGLWidget::projectTransform () {
    // glm::perspective (FOV en radians, ra window, znear, zfar)
    glm::mat4 Proj = glm::perspective (float(M_PI)/2.0f, 1.0f, 0.4f, 3.0f);
    glUniformMatrix4fv (projLoc, 1, GL_FALSE, &Proj[0][0]);
}
```

Aquest mètode és nou a la classe MyGLWidget.

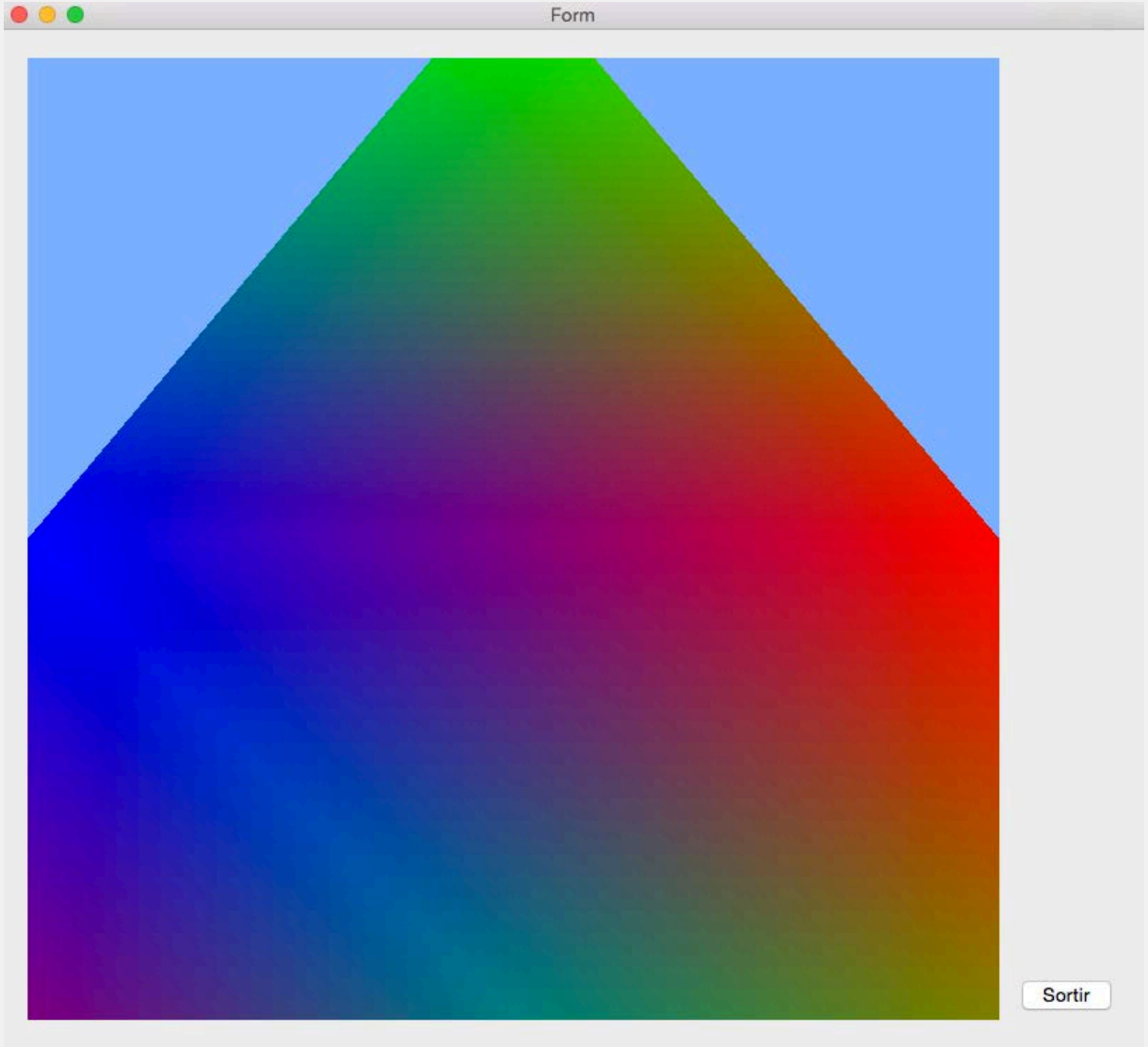
Transformació de projecció
(exercici 1)

- Al vertex shader (afegir):

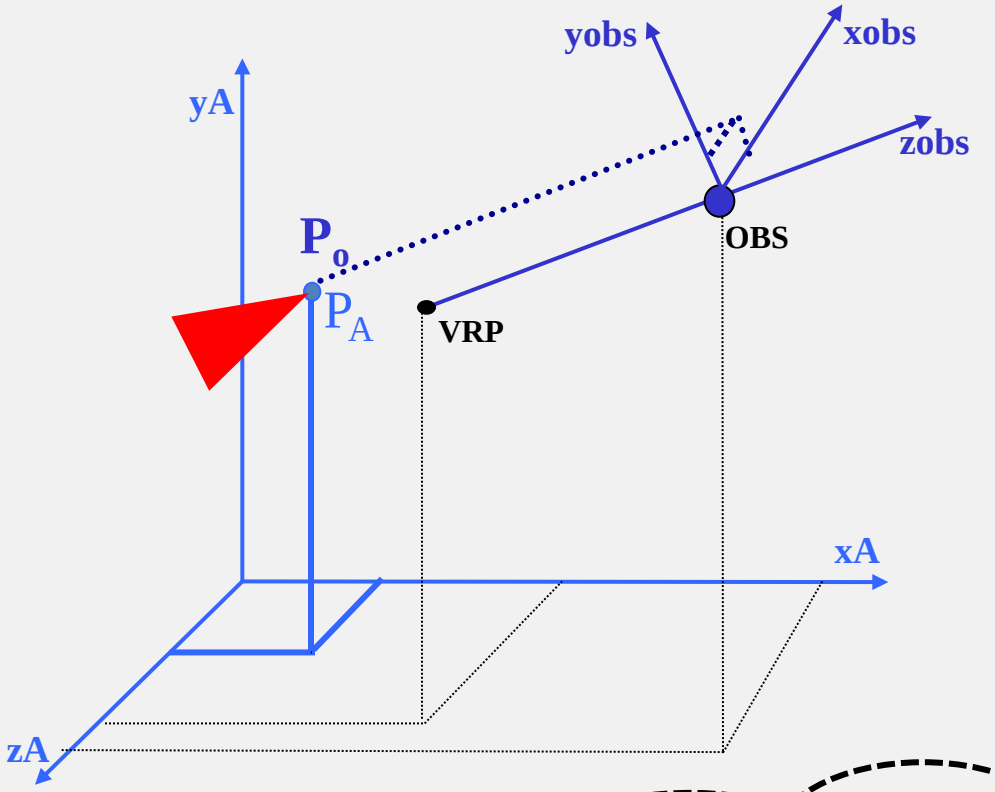
```
...
uniform mat4 proj;
...
void main () {
    ...
    gl_Position = proj * ... * vec4 (vertex, 1.0);
}
```

Transformació de projecció

(exercici 1)

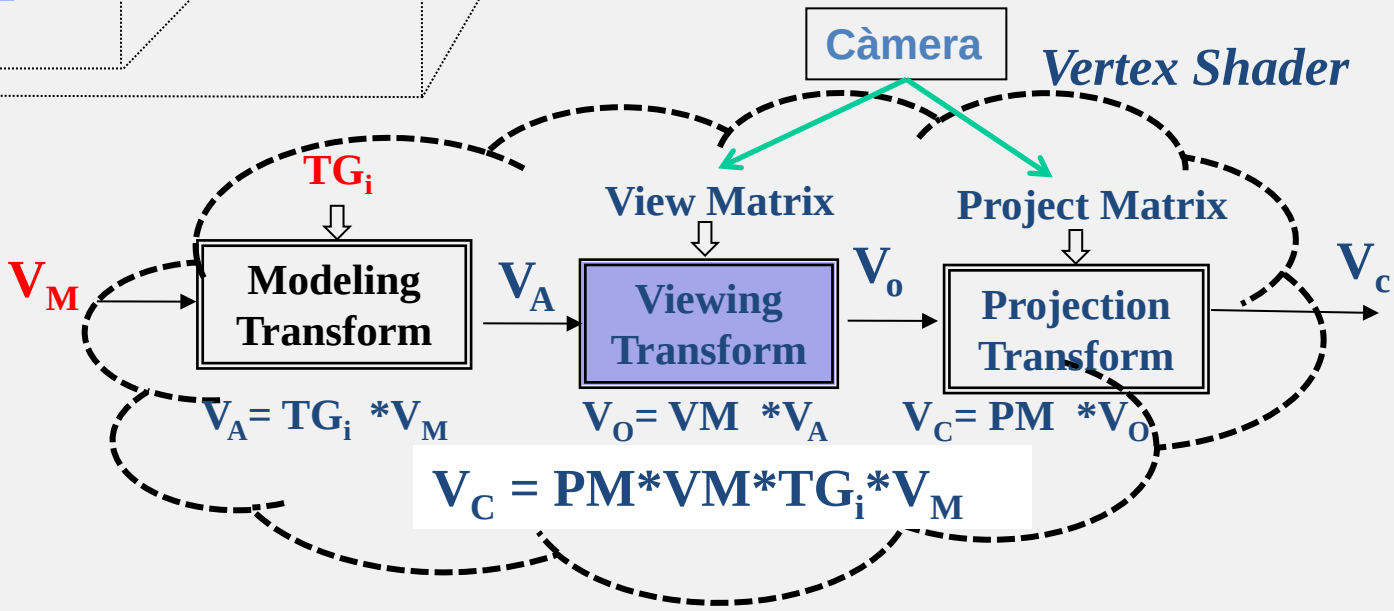


Transformació de punt de vista (view)



OBS , VRP, up

```
VM = lookAt (OBS,VRP,up);
viewMatrix (VM);
```



Transformació de punt de vista (view)
(exercici 2)

- Al codi de MyGLWidget cal (igual que abans):
  - Demanar un uniform location per al uniform de la matriu

```
viewLoc = glGetUniformLocation (program->programId(), “view”)
```

  - Definir un mètode que ens calculi la transformació de punt de vista (view) i enviï el uniform amb la matriu cap al vertex shader

```
void MyGLWidget::viewTransform () {
    // glm::lookAt (OBS, VRP, UP)
    glm::mat4 View = glm::lookAt (glm::vec3(0,0,1),
                                  glm::vec3(0,0,0), glm::vec3(0,1,0));
    glUniformMatrix4fv (viewLoc, 1, GL_FALSE, &View[0][0]);
}
```

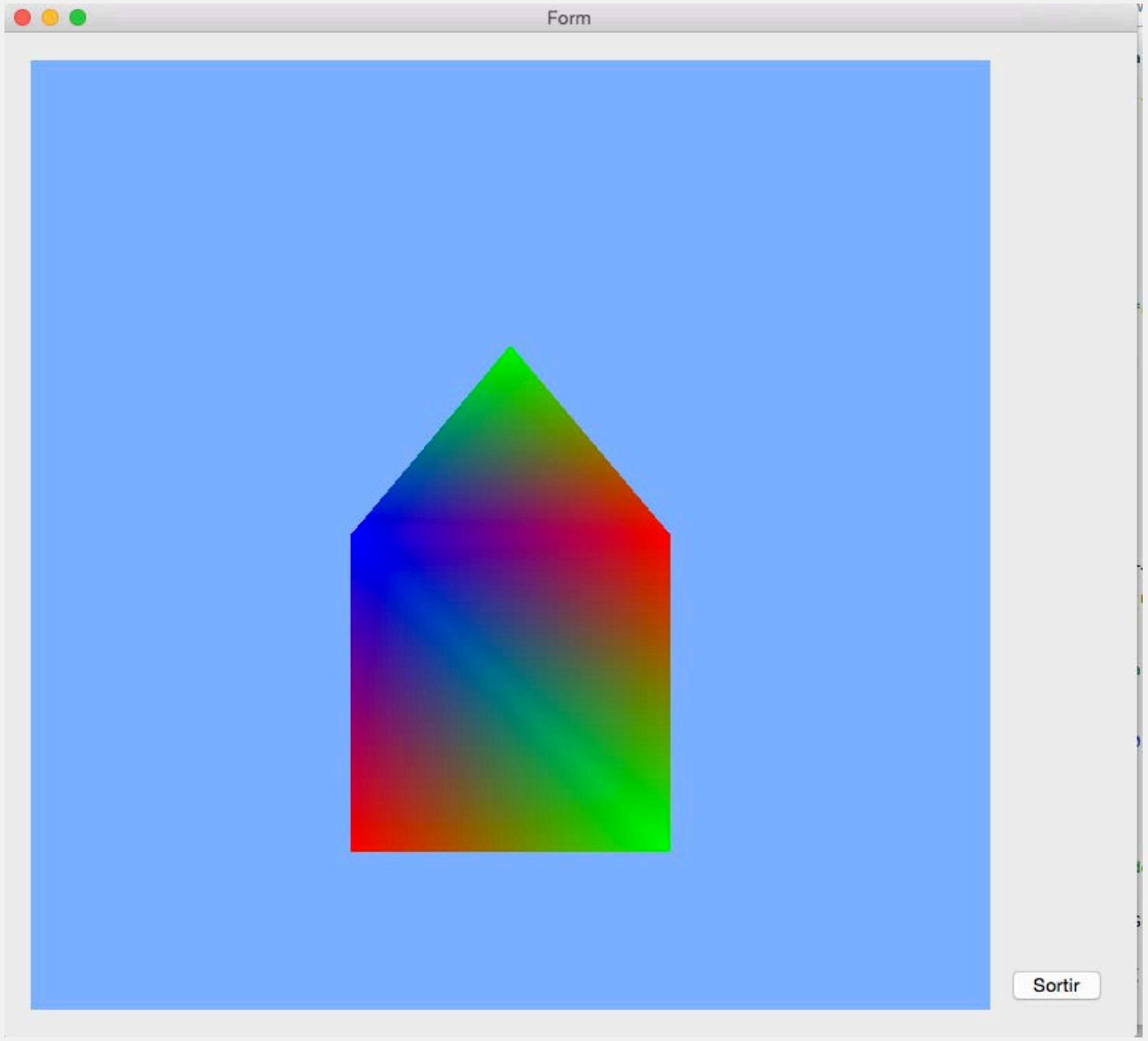
Transformació de punt de vista (view)
(exercici 2)

- Al vertex shader (afegir):

```
...
uniform mat4 view;
...
void main () {
    ...
    gl_Position = proj * view * ... * vec4 (vertex, 1.0);
}
```

Transformació de punt de vista (view)

(exercici 2)



Laboratori OpenGL – Sessió 2.1
Bloc 2

- Nou esquelet de base (~/assig/idi/blocs/bloc-2)
- Transformacions de càmera amb glm (view i projection)
- Classe Model – càrrega d'objectes OBJ
- Z-buffer
- Control d'errors d'OpenGL

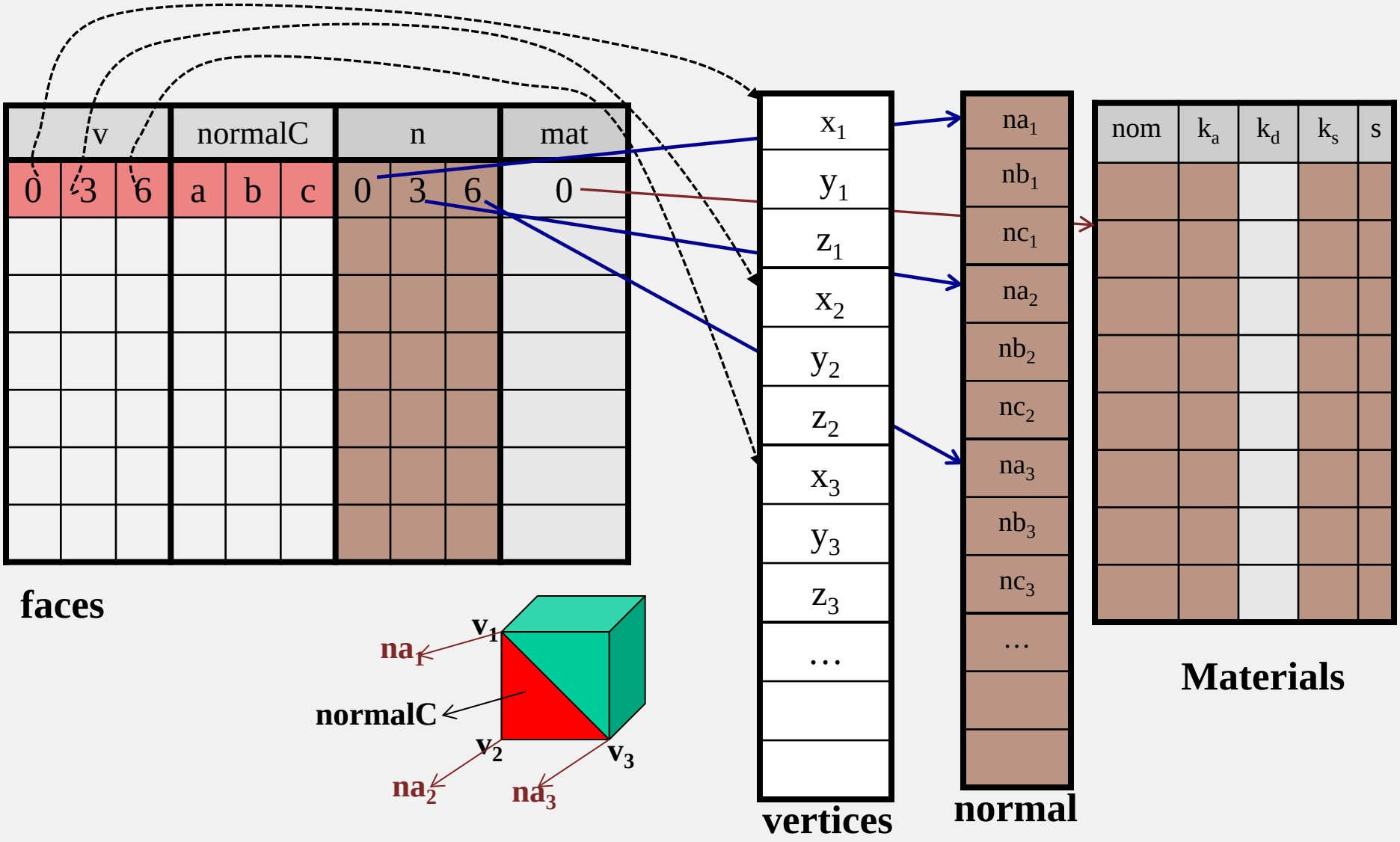
Càrrega de models OBJ (exercici 4)

- Classe Model: permet carregar *objecte.obj*
  - ~/assig/idi/Model (copieu-vos la carpeta en un directori vostre)
  - Analitzeu el model.h (classe Model)
  - Mètode Model::load(std::string filename)
    Inicialitza les estructures de dades a partir d'un model en format OBJ-Wavefront en disc
- Modifiqueu el fitxer .pro afegint

  INCLUDEPATH += <el-vostre-directori>/Model

  SOURCES += <el-vostre-directori>/Model /model.cpp
- En *~/assig/idi/models* trobareu models d'objectes.
  - Si els copieu a un directori local, per cada .obj copieu també (si existeix) el .mtl → definició dels materials corresponents.
  - Fins la propera sessió usarem el **HomerProves**
- Més models els podeu trobar a la xarxa.

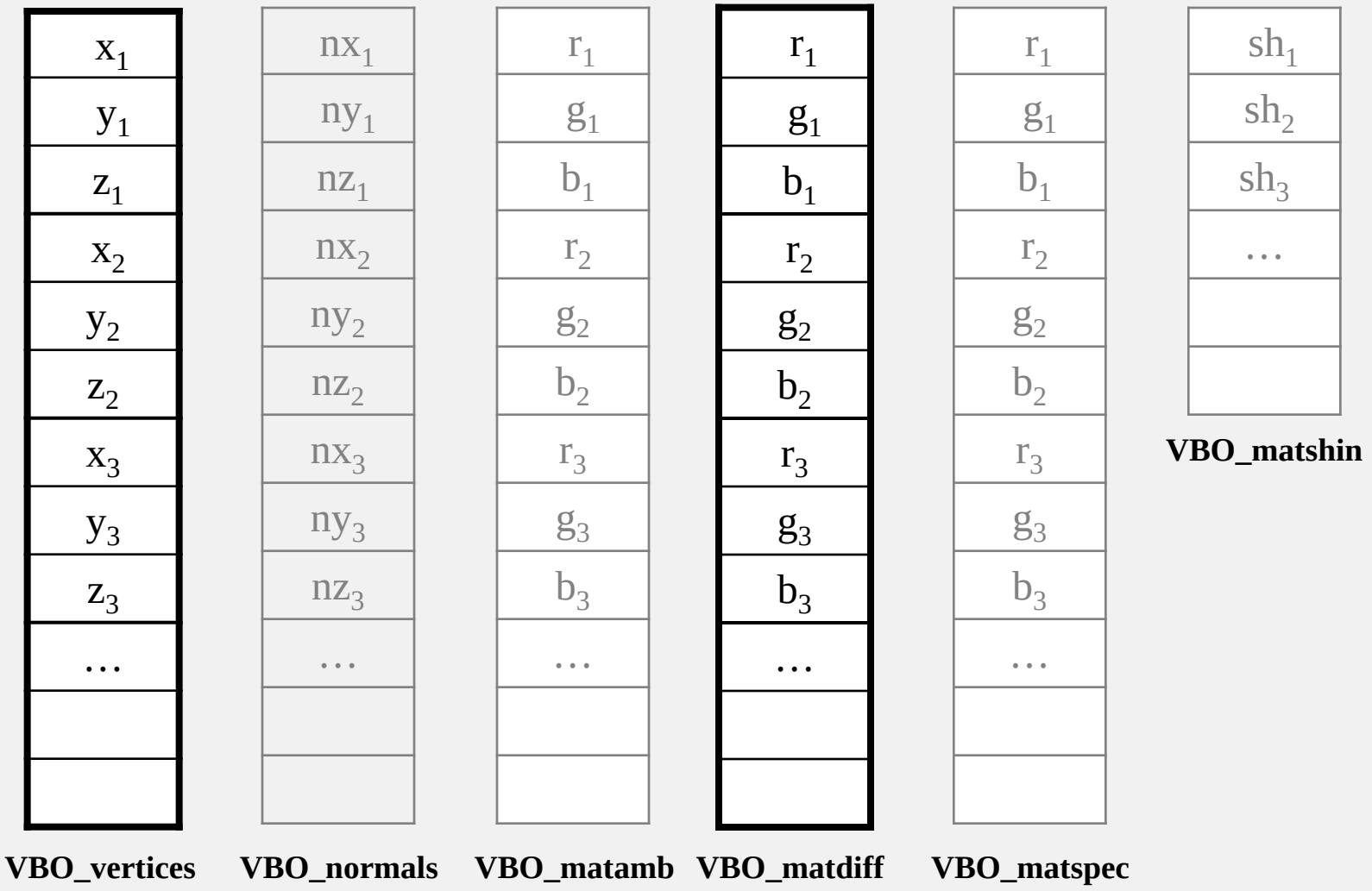
Representació classe Model



Analitzeu l'arxiu model.h

Compte!! amb el nom dels camps de Material que en l'esquema són simbòlics; p.e. k_d és float diffuse[4]

Representació auxiliar de la classe Model



Ús de la classe Model (exercici 4)

- Construcció d'un objecte de tipus Model (declaració)

```
Model m;  // un únic model
Model vectorModels[3];  // array de 3 models
vector<Model> models;  // vector stl de models
```

- Càrrega d'un arxiu (model) .obj

```
m.load ("../models/HomerProves.obj");
```

- Accés als seus VBOs (els genera la propia classe Model)

```
glBufferData (..., m.VBO_vertices (), GL_STATIC_DRAW); // posició
glBufferData (..., m.VBO_matdiff (), GL_STATIC_DRAW);  // color
```

- Per saber el nombre de cares (totes les cares són triangles)

```
m.faces().size()
```

- Mida en bytes dels buffers

```
sizeof(GLfloat) * m.faces ().size () * 3 * 3
```

Exemples

- Pas de dades del buffer de posicions cap a la GPU

```
glBufferData (GL_ARRAY_BUFFER,
              sizeof(GLfloat) * m.faces ().size () * 3 * 3,
              m.VBO_vertices (), GL_STATIC_DRAW);
```

- Pintar l'objecte

```
glDrawArrays (GL_TRIANGLES, 0, m.faces ().size () * 3);
```

- Recorregut de la taula de vèrtexs

```
for (unsigned int i = 0; i < m.vertices().size(); i+=3) {
   // escric per pantalla les coordenades del vèrtex
   std::cout << "(x, y, z) = (" << m.vertices()[i] << ", "
                                  << m.vertices()[i+1] << ", "
                                  << m.vertices()[i+2] << ")" << std::endl;
}
```

Laboratori OpenGL – Sessió 2.1
Bloc 2

- Nou esquelet de base (~/assig/idi/blocs/bloc-2)
- Transformacions de càmera amb glm (view i projection)
- Classe Model – càrrega d'objectes OBJ
- Z-buffer
- Control d'errors d'OpenGL

Z-buffer (exercici 4)

- Algorisme de Z-buffer:
  - Activar el z-buffer (només cal fer-ho un cop!)

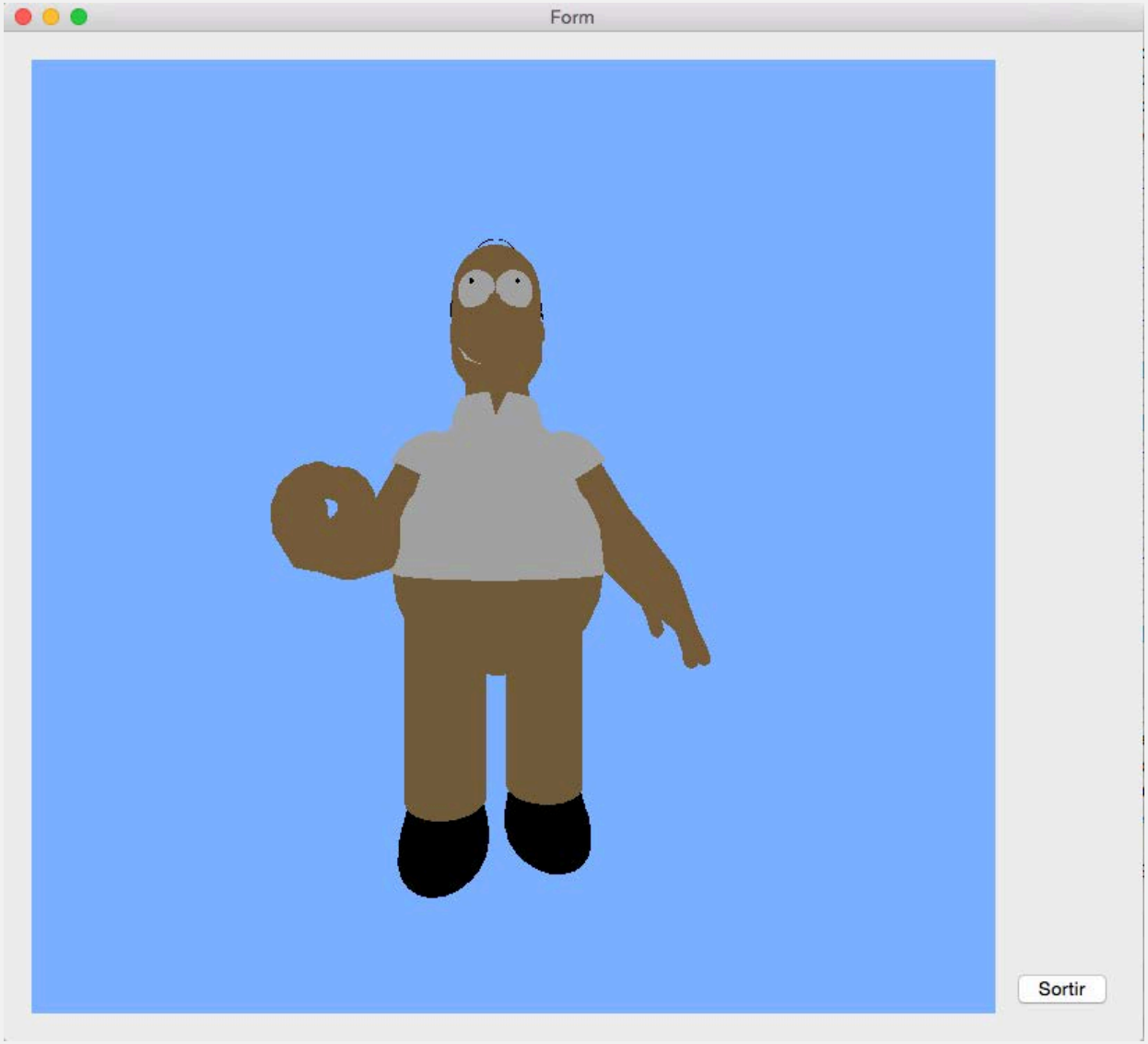
    ```
    glEnable (GL_DEPTH_TEST);
    ```

  - Esborrar el buffer de profunditats a la vegada que el frame buffer

    ```
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    ```

Càrrega i pintat del HomerProves

(exercici 4)



Laboratori OpenGL – Sessió 2.1
Bloc 2

- Nou esquelet de base (~/assig/idi/blocs/bloc-2)
- Transformacions de càmera amb glm (view i projection)
- Classe Model – càrrega d'objectes OBJ
- Z-buffer
- Control d'errors d'OpenGL

Control d'errors d'OpenGL

- Per saber si una crida a OpenGL ha donat error:
  - En cas d'error en una de les seves funcions, OpenGL assigna un codi que identifica l'error a una variable d'entorn

    GL_ERROR

  - En el MyGLWidget podem usar la crida

    CHECK()

    allà on vulguem controlar si una crida a OpenGL ha produït un error

    Si s'ha produït un error des de la darrera comprovació, ens escriurà per pantalla:

    glError in file <file> @ line <line>: <error> function: <function>

    on <error> és la constant d'error (per exemple: GL_INVALID_ENUM)
    i cal anar al manual de la crida d'OpenGL per saber què l'ha produït

Exercicis sessió 2.1

El que cal que feu en aquesta sessió és:

1) **Mirar codi esquelet bloc 2** (~/assig/idi/blocs/bloc-2) i entendre tot el que està programat.
2) **Feu TOTS els exercicis** que teniu al guió per a aquesta sessió. És important que els feu **en l'ordre** que es presenten.
   - Feu ús del que necessiteu del codi que s'ha presentat en aquestes transparències, però vigileu si feu *copy&paste* perquè copiar de pdf us pot portar problemes.