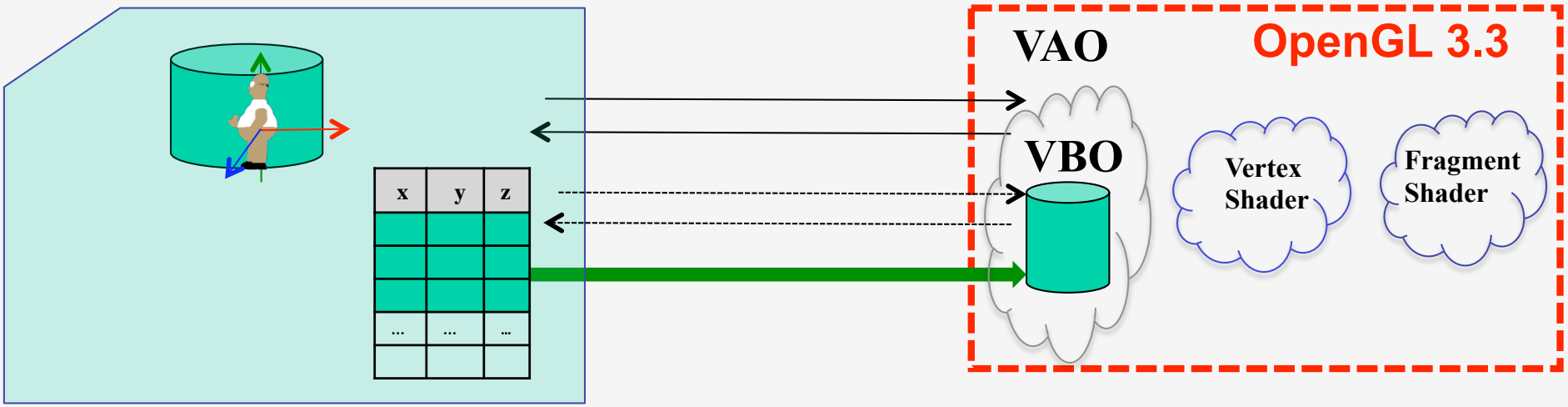
Laboratori OpenGL – Sessió 3

- Ús de *uniforms*
- Interacció directa a Qt
- Transformacions Geomètriques amb glm

Recordatori: Visualitzar en OpenGL 3.3: "core" mode

Haurem de programar el Vertex Shader i Fragment Shader que implementen una part del procés de visualització.



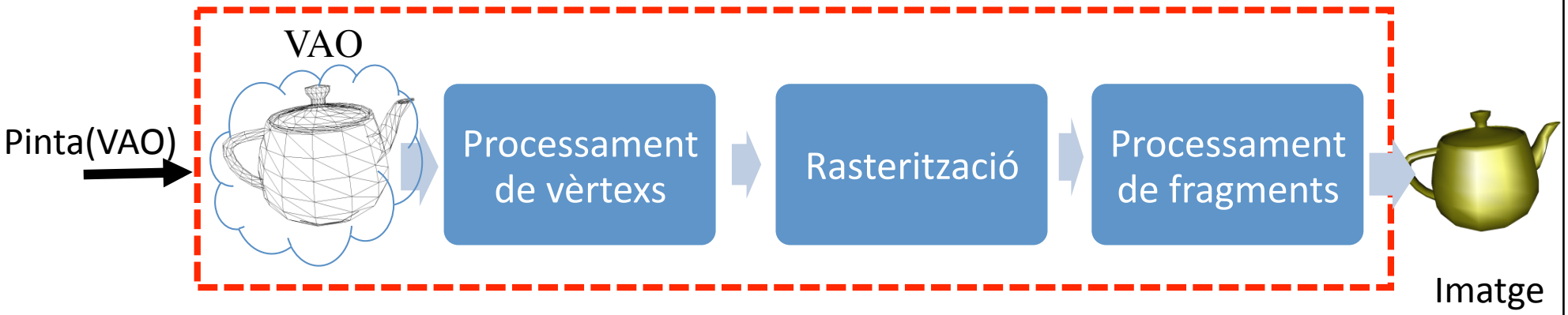
Aplicació. Model Geomètric

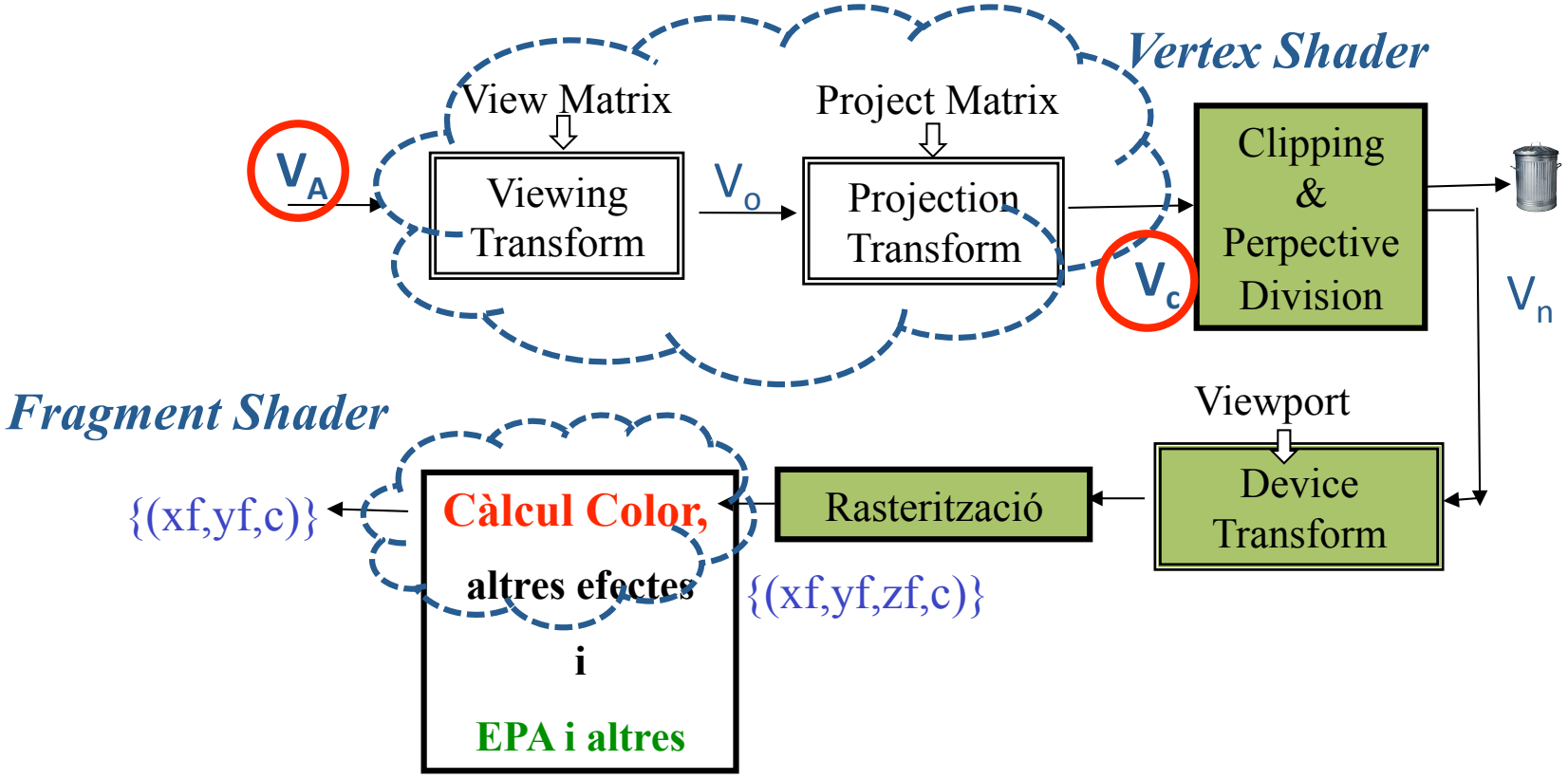
Haurem de:

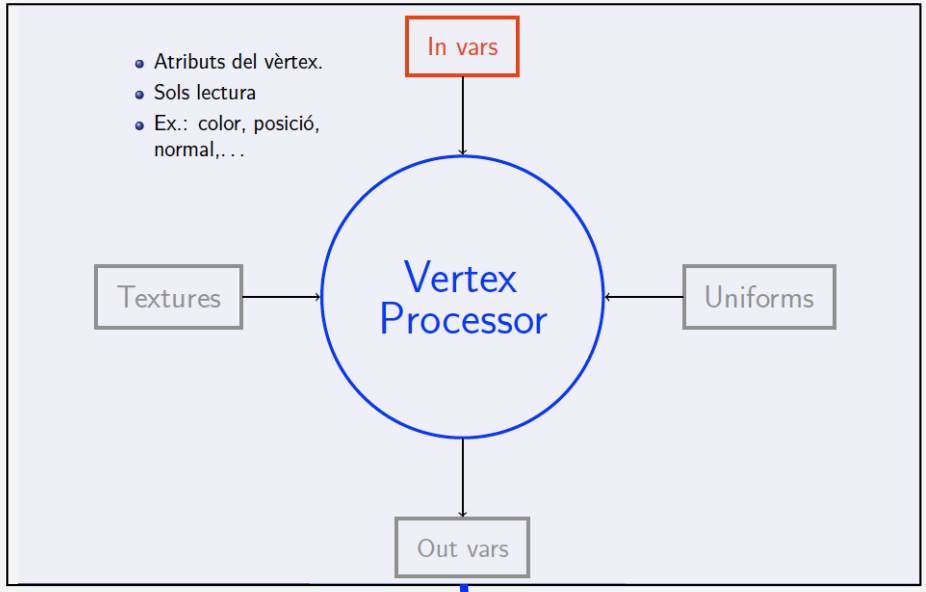
- **Generar un VAO per cada model 3D.**
- **Per a cada VAO, generar i omplir un VBO per a cada informació/ atribut dels vèrtexs.**

Per exemple: un per coordenades i altre pel color.

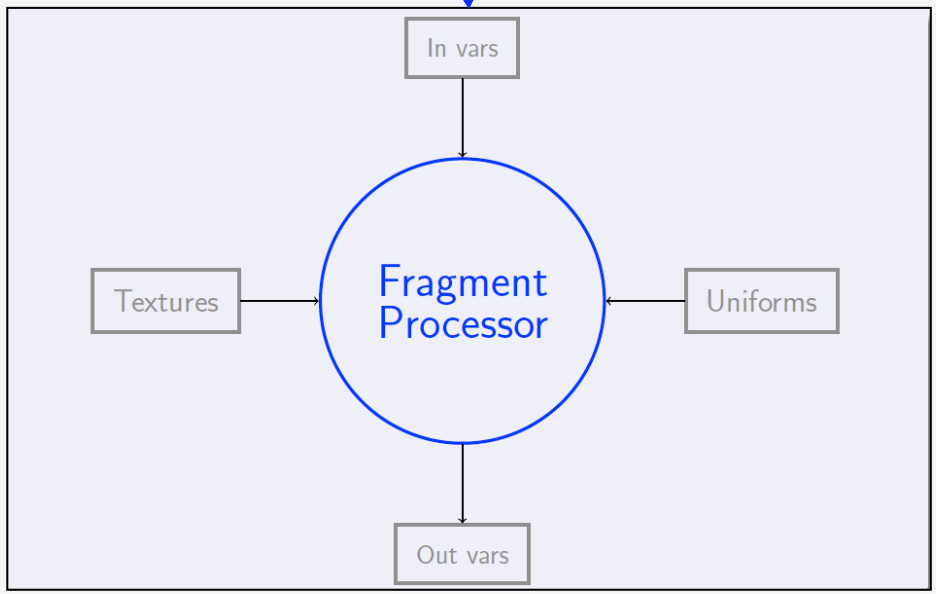
Paradigma projectiu bàsic amb OpenGL 3.3





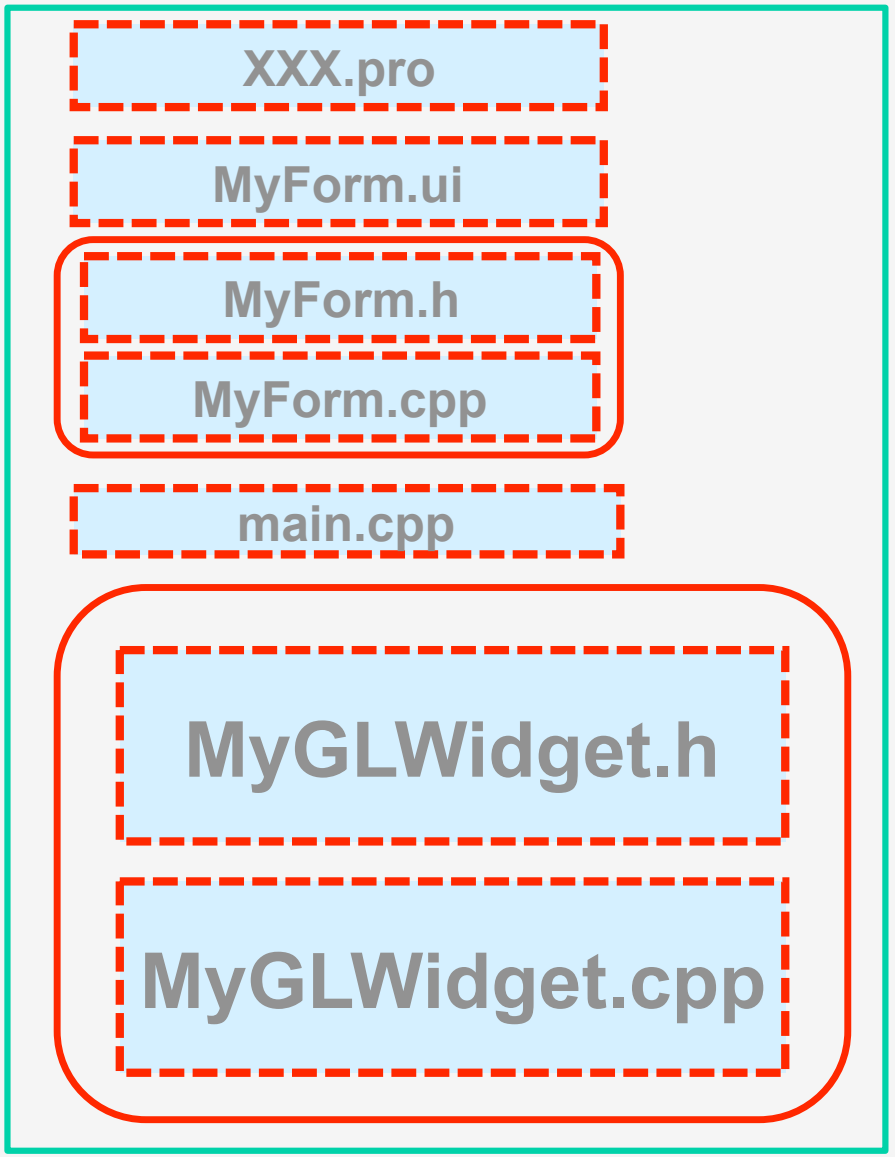


Valor interpolat a partir del seu valor en els vèrtexs



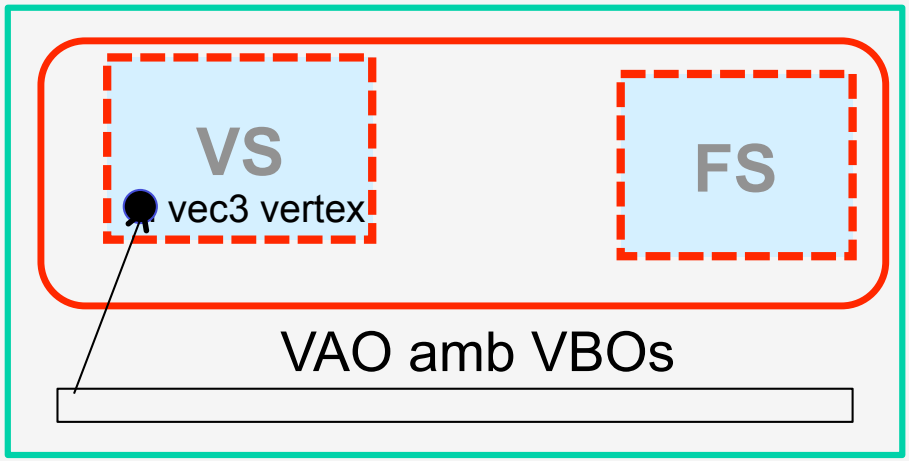

```
#version 330 core
in vec3 vertex;
in vec3 color;
out vec3 fcolor;
void main()  {
    gl_Position = vec4 (vertex, 1.0);
    fcolor = color;
}
```

```
#version 330 core
in vec3 fcolor;
out vec4 FragColor;
void main()  {
    FragColor=vec4(fcolor,1.0);
}
```



qmake-qt5
make

QOpenGLShader Program
QOpenGLShader

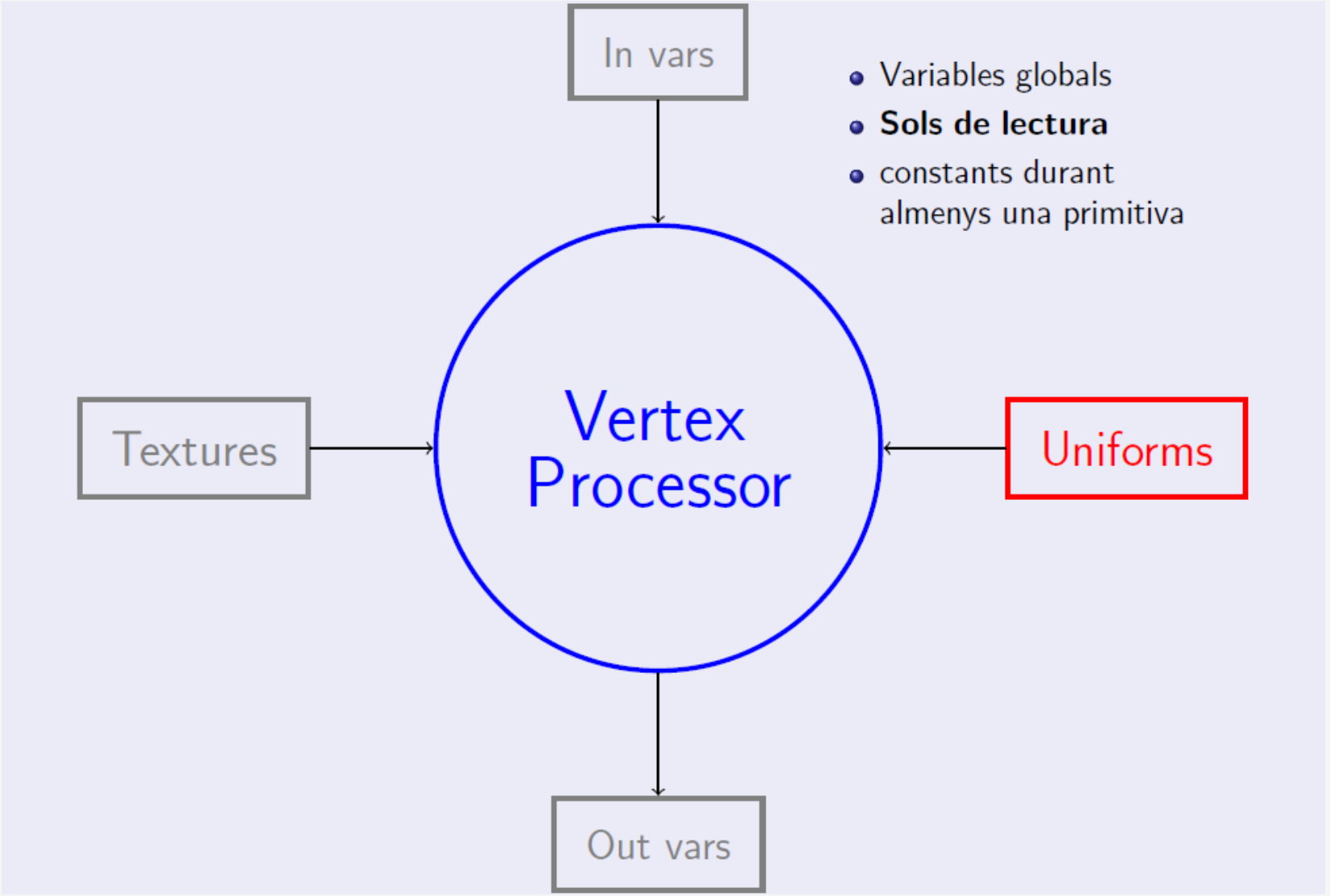


vertexLoc= **glGetAttribLocation**
(program ->programId(), "vertex");

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

glDrawArrays(…);

Uniforms


In vars
Variables globals
Sols de lectura
constants durant almenys una primitiva
Textures
Vertex Processor
Uniforms
Out vars

Uniforms

- Al vertex shader:

```
#version 330 core

in vec3 vertex;
uniform float val;

void main ()
{
        gl_Position = vec4 (vertex * val, 1.0);
}
```

XXX.pro

MyForm.ui

MyForm.h

MyForm.cpp

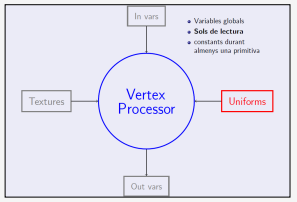
main.cpp

MyGLWidget.h

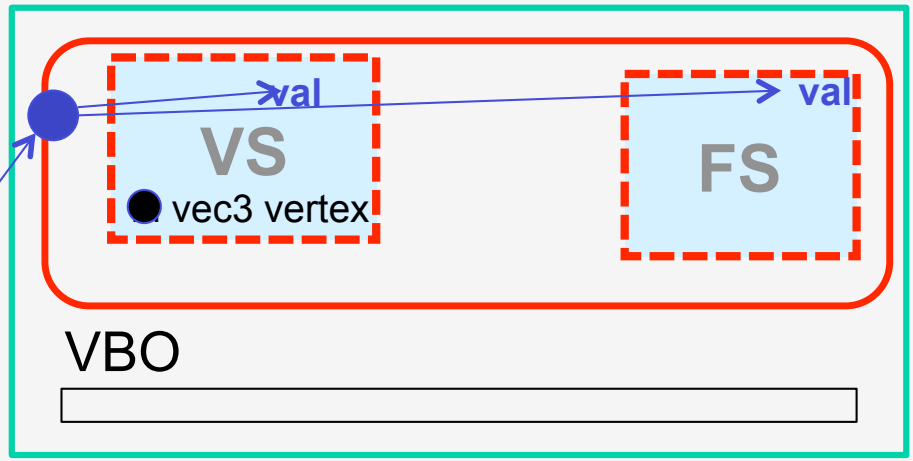
MyGLWidget.cpp

scl

qmake-qt5
make



QOpenGLShader Program
QOpenGLShader



vertexLoc= **glGetAttribLocation**
(program ->programId(), "vertex");
glVertexAttribPointer(vertexLoc, 3,
GL_FLOAT, GL_FALSE, 0, 0);
glDrawArrays(…);

varLoc =**glGetUniformLocation**(program->programId (), “val”);
glUniform1f (varLoc, scl);

Uniforms

- Al codi cpp de MyGLWidget:
  - Associar identificador al shader (només cal fer-ho un cop)

```
varLoc = glGetUniformLocation (program->programId (), “val”);
```

  - Donar valor al uniform (cal fer-ho cada cop que es vulgui canviar el valor del paràmetre *scl*)

```
glUniform1f (varLoc, scl);
// scl variable que conté el valor que es vol per “val”
```

Funcions OpenGL per a uniforms

GLint glGetUniformLocation (GLuint *program*, const GLchar **name*);

Obté la posició d'un uniform declarat al shader amb nom *name*

program : program al que està lligat el shader que conté el uniform

name : nom que identifica al uniform en el shader

void glUniform1f (GLint *location*, GLfloat *value*);

Especifica el valor *value* per al uniform identificat per *location*

location : identificador del uniform aconseguit amb glGetUniformLocation

value : valor que es passa cap al shader

Funcions OpenGL per a uniforms

Altres crides possibles:

glUniform{1|2|3|4}{f|i|ui} // nombre de paràmetres depenent de 1|2|3|4

1 – tipus float (f), int (i), unsigned int (ui), bool (f|i|ui)

2 – tipus vec2 (f), ivec2 (i), uvec2 (ui), bvec2 (f|i|ui)

3 – tipus vec3 (f), ivec3 (i), uvec3 (ui), bvec3 (f|i|ui)

4 – tipus vec4 (f), ivec4 (i), uvec4 (ui), bvec4 (f|i|ui)

glUniform{1|2|3|4}{f|i|ui}v (GLint *loc*, GLsizei *count*, const Type **value*);

{1|2|3|4} i {f|i|ui} – igual que crida anterior

count – nombre d'elements de l'array *value,* 1: un sol valor; >=1 array de valors

glUniformMatrix{2|3|4|2x3|3x2|2x4|4x2|3x4|4x3}fv
(GLint *loc*, GLsizei *count*, GLboolean transpose, const GLfloat **value*);

{2|3|4|2x3|3x2|2x4|4x2|3x4|4x3} – defineix les dimensions de la matriu

count – nombre de matrius de l'array *value*

transpose – si la matriu s'ha de transposar

Funcions OpenGL per a uniforms

Les que més usarem:

glUniform1{f|i|ui} // per a passar un únic valor

Exemple:

```
float scl = 0.5;
glUniform1f (varLoc, scl);
```

glUniform3fv // per a passar vectors de 3 components

Exemple:

```
glm::vec3 posLlum = glm::vec3 (1.0, 5.0, 0.0);
glUniform3fv (posLlumLoc, 1, &posLlum[0]);
```

glUniformMatrix4fv // per a passar les matrius de transformació

Exemple:

```
glm::mat4 TG = glm::mat4(1.0); // o glm::mat4 TG(1.0);
glUniformMatrix4fv (transLoc, 1, GL_FALSE, &TG[0][0])
```

Interacció directa amb Qt

- Per tal de tractar events de baix nivell en una aplicació OpenGL amb Qt cal re-implementar els mètodes virtuals corresponents (a la classe MyGLWidget):

```
virtual void mousePressEvent ( QMouseEvent * e )
virtual void mouseReleaseEvent ( QMouseEvent * e )
virtual void mouseMoveEvent ( QMouseEvent * e )
virtual void keyPressEvent ( QKeyEvent * e )
```

Interacció directa amb Qt

- Per tal de tractar events de baix nivell en una aplicació OpenGL amb Qt cal re-implementar els mètodes virtuals corresponents (a la classe MyGLWidget):

```
virtual void mousePressEvent ( QMouseEvent * e )
virtual void mouseReleaseEvent ( QMouseEvent * e )
virtual void mouseMoveEvent ( QMouseEvent * e )
virtual void keyPressEvent ( QKeyEvent * e )
```

Interacció directa amb Qt

- Exemple d'implementació (incloure en MyGLWidget.cpp):

```
void MyGLWidget::keyPressEvent (QKeyEvent *e) {
  makeCurrent ();
  switch ( e->key() ) {
    case  Qt::Key_S :
          scl += 0.1;
          glUniform1f (varLoc, scl);
          break;
    case  Qt::Key_D :
          scl -= 0.1;
          glUniform1f (varLoc, scl);
          break;
    default: e->ignore (); // propagar al pare
  }
  update ();
}
```

En MyGLWidget.h caldrà afegir:

#include <QKeyEvent>

i declarar el mètode virtual

virtual void keyPressEvent (QKeyEvent *e);

Interacció directa amb Qt

- Exemple d'implementació:

```
void MyGLWidget::keyPressEvent (QKeyEvent *e) {
  makeCurrent ();       // fa actiu el nostre context d'OpenGL
  switch ( e->key() ) {
    case  Qt::Key_S :
          scl += 0.1;
          glUniform1f (varLoc, scl);
          break;
    case  Qt::Key_D :
          scl -= 0.1;
          glUniform1f (varLoc, scl);
          break;
    default: e->ignore (); // propagar al pare
  }
  update ();     // provoca que es torni a pintar l'escena
}
```

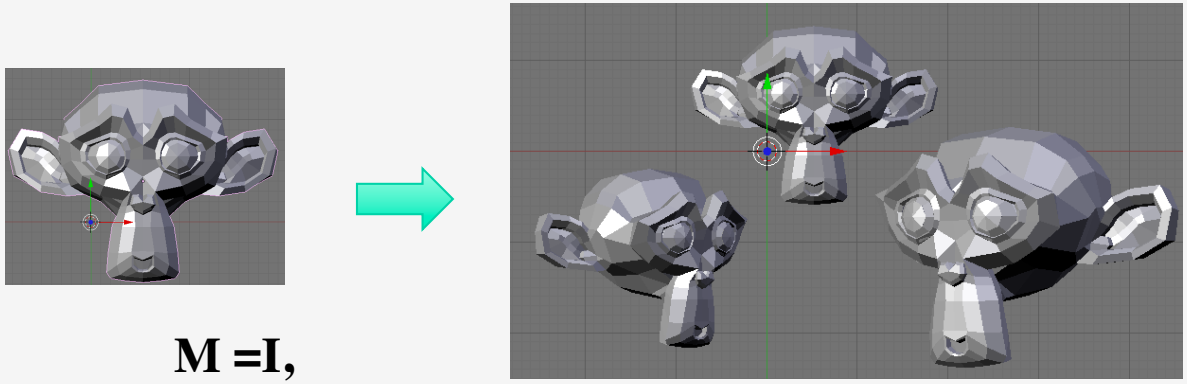
Primer Exercici sessió 3

1) Afegiu al vostre codi el uniform *scl* descrit en els exemples de codi vistos i feu, com hem vist, que amb les tecles ‘s’ i ‘d’ aquest valor del uniform augmenti o disminueixi respectivament.

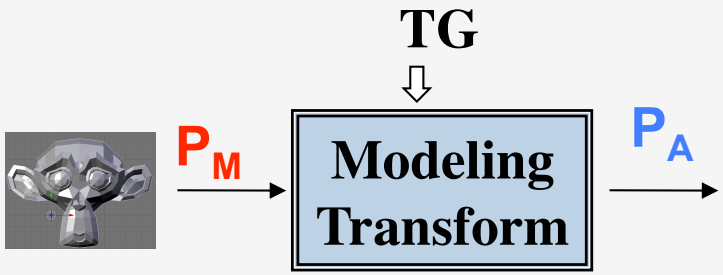
   Ho podeu fer tot amb el triangle del primer dia; però millor si teniu color per vèrtex; o sigui si teniu 1VAO amb 2 VBO un per les coordenades i altre pel color.

Matrius de transformació

- Hem de poder transformar els vèrtexs (pex, amb transformacions de model):

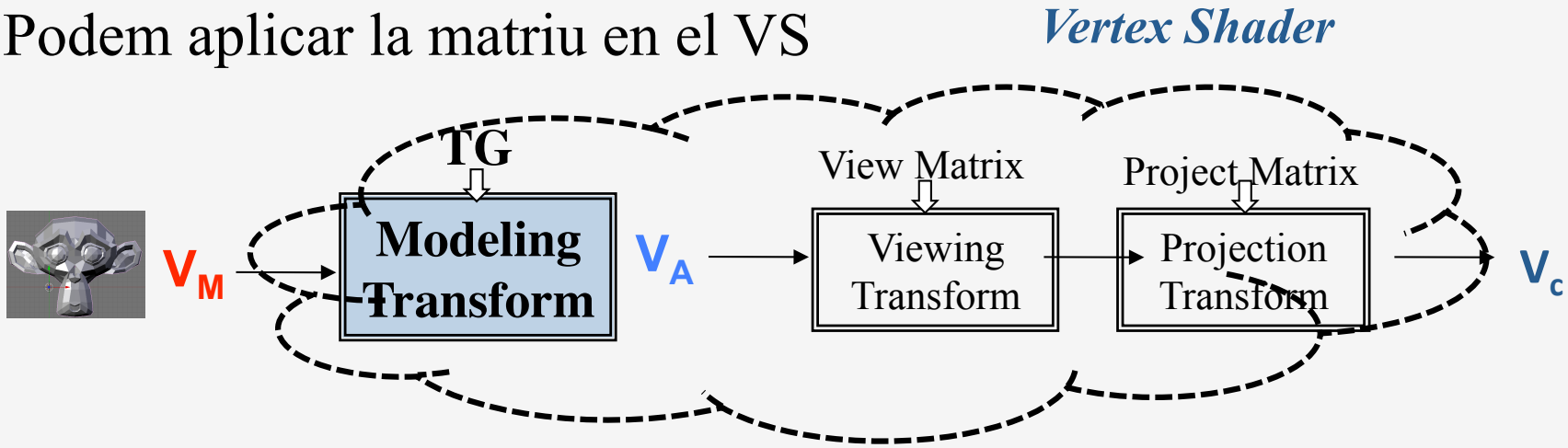


M =I,
TG_1, TG_2



Matrius de transformació

- Podem aplicar la matriu en el VS




```
#version 330 core

in vec3 vertex;
uniform mat4 TG;

void main ()
{
	gl_Position = TG*vec4 (vertex, 1.0);
}
```

Matrius de transformació

- Usarem glm per a construir la matriu de transformació:

- Exemple: ⇩

```
void MyGLWidget::modelTransform () {
    glm::mat4 TG (1.0);   // Matriu de transformació, inicialment identitat
    TG = glm::translate (TG, glm::vec3 (-0.5, 0.5, 0.0));
    glUniformMatrix4fv (transLoc, 1, GL_FALSE, &TG[0][0]);
}
```

- Al nostre programa (en els llocs corresponents de MyGLWidget):

```
GLuint transLoc;
transLoc = glGetUniformLocation (program->programId(), "TG");
```

Completant l'exemple

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

   modelTransform();
   glBindVertexArray(VAO);
      glDrawArrays(GL_TRIANGLES, 0, 3);
   glBindVertexArray(0);

}
```

Noteu que caldrà cridar a *modelTransform()* en el mètode *painGL()* abans d'enviar a pintar el model (*glDrawArrays*) per a actualitzar la TG que cal aplicar-li

Matrius de transformació

- Mètodes de transformacions geomètriques de la glm:

  translate (glm::mat4 *m_ant*, glm::vec3 *vec_trans*);

  // acumula a la matriu anterior *m_ant* la matriu resultant de fer una
  // translació pel vector *vec_trans*

  scale (glm::mat4 *m_ant*, glm::vec3 *vec_scale*);

  // acumula a la matriu anterior *m_ant* la matriu resultant de fer un
  // escalat en cada direcció segons els factors *vec_scale*

  rotate (glm::mat4 *m_ant*, float *angle*, glm::vec3 *vec_axe*);

  // acumula a la matriu anterior *m_ant* la matriu resultant de fer una
  // rotació de *angle* radians al voltant de l'eix *vec_axe*

- Per a poder incloure aquestes funcions de la glm:

  #include "glm/gtc/matrix_transform.hpp"

- Per a que els angles a usar a la rotació siguin en radians ens cal afegir al nostre codi (al fitxer MyGLWidget.h, abans includes de glm) el següent:

  #define GLM_FORCE_RADIANS

Exercicis sessió 3

El que cal que feu en aquesta sessió és:

0. Afegiu al vostre codi el uniform *scl* descrit en els exemples de codi vistos i feu, com hem vist, que amb les tecles 's' i 'd' aquest valor del uniform augmenti o disminueixi respectivament.

   Mireu el codi que teniu en transparència de *keyPressEvent(..)*

1. Feu els exercicis que teniu al guió per a aquesta sessió (de l'1 al 6).

- En exercicis 1 a 3, amb el teclat modificarem el vector de translació o l'angle de gir que hauran de servir en el mètode "modelTransform" per crear les matrius corresponents.

  Noteu que en keyPressEvent() ara areu de modificar variables que s'utilitzaran en la modelTransform() que implementar/crea i la TG.

  Recordeu de cridar a modelTransform abans de pintar el VAO

Exercicis sessió 3

1. Feu els exercicis que teniu al guió per a aquesta sessió (de l'1 al 6).
   - L'exercici 4 diu que afegiu una matriu d'escalat amb el valor scl. Cal substituir l'escalat que es feia directament en el shader (no fer servir scl) i incloure l'escalat en la TG de *modelTransform()*
   - Exercici 5: escalat no uniforme (fàcil ☺
   - Exercici 6: dos objectes i aplicar a cadascú una TG diferent o el model d'un triangle i pintar en dos llocs diferentes (una TG diferent per cada instància).
2. Ara feu escalat usant el moviment del ratolí, de manera que quan **el ratolí** es mou d'esquerra a dreta s'incrementa el factor d'escala i quan el ratolí es mou de dreta a esquerra es decrementa.

Interacció directa amb Qt

- Exemple d'implementació:

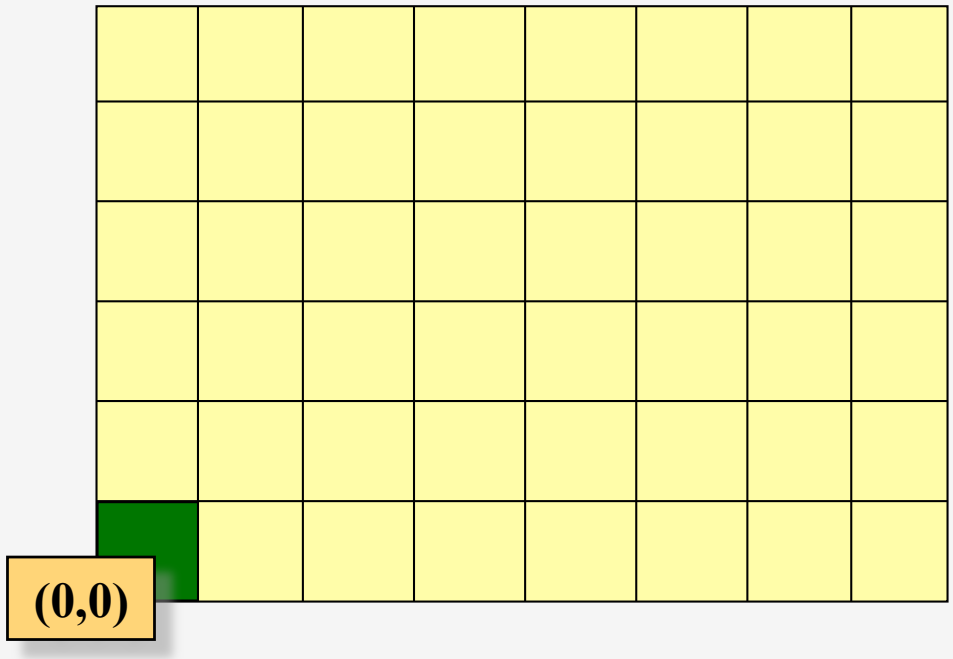
```
void MyGLWidget::mousePressEvent (QMouseEvent *e)
{
    xClick=e->x();
    yClick = e ->y();
}

void MyGLWidget::mouseMoveEvent (QMouseEvent *e)
{
      incx = e->x() – xClick;
      scl+=incx/600;
      xClick=e->x();
      …
      update();
}
```

En MyGLWidget.h caldrà:

#include <QMouseEvent> i declarar el mètodes virtuals

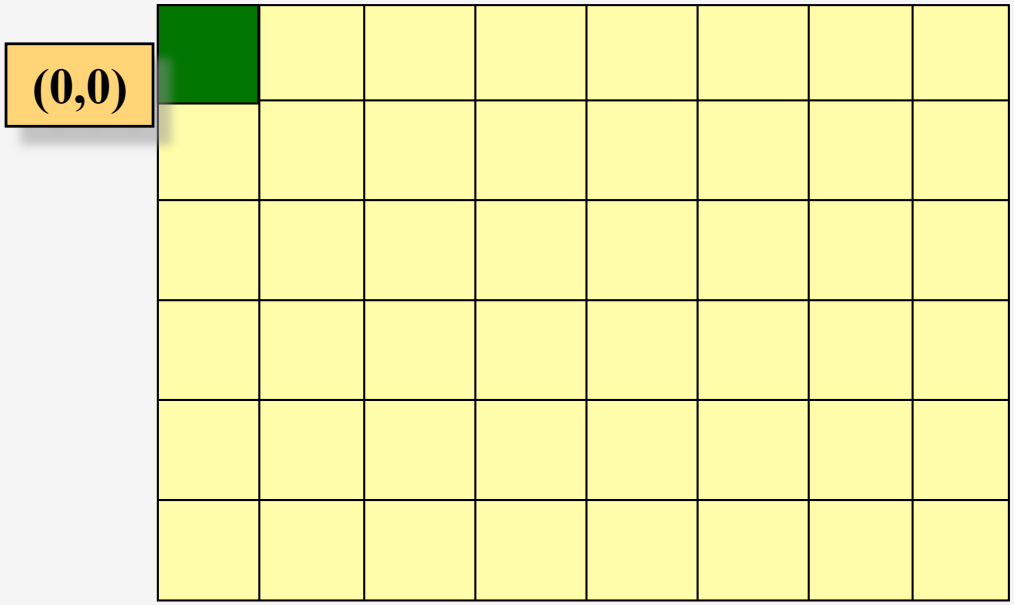
Consideració important

- OpenGL considera l'origen del SC de dispositiu a la cantonada inferior esquerra de la finestra gràfica.



Consideració important

- Qt considera l'origen del SC de dispositiu a la cantonada superior esquerra de la finestra gràfica.



Exercicis sessió 3

- Afegiu al vostre codi un uniform *vp,* de tipus *ivec2* que representi les dimensions del *viewport* (en píxels) que ens indica el métode resizeGL de la classe MyGLWidget. Cal tenir aquest uniform *vp* declarat en el fragment shader i usar-lo per a fer l'exercici dels 4 quadrants de manera que segueixi funcionant correctament després d'un redimensionament de la finestra d'OpenGL.