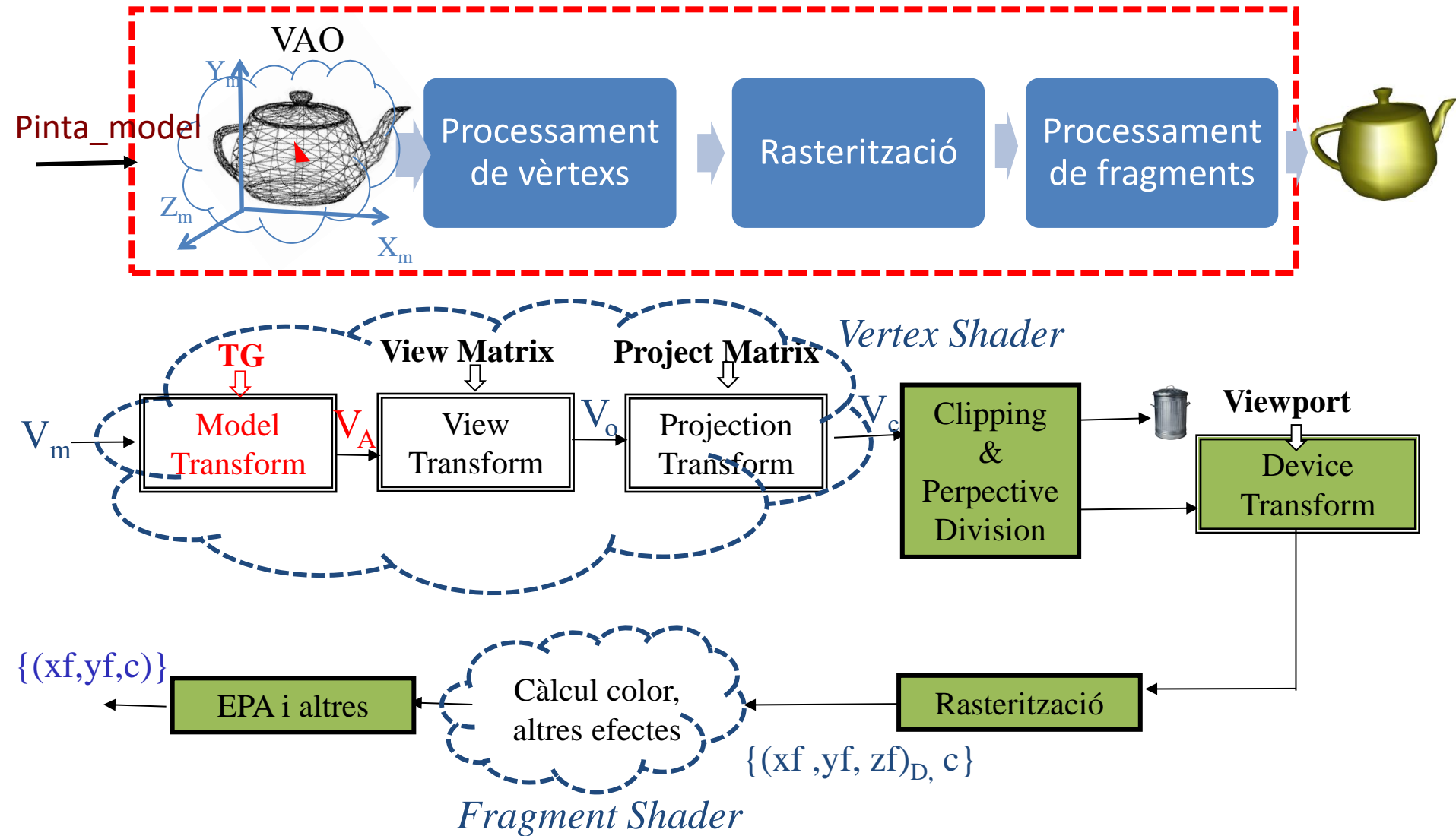
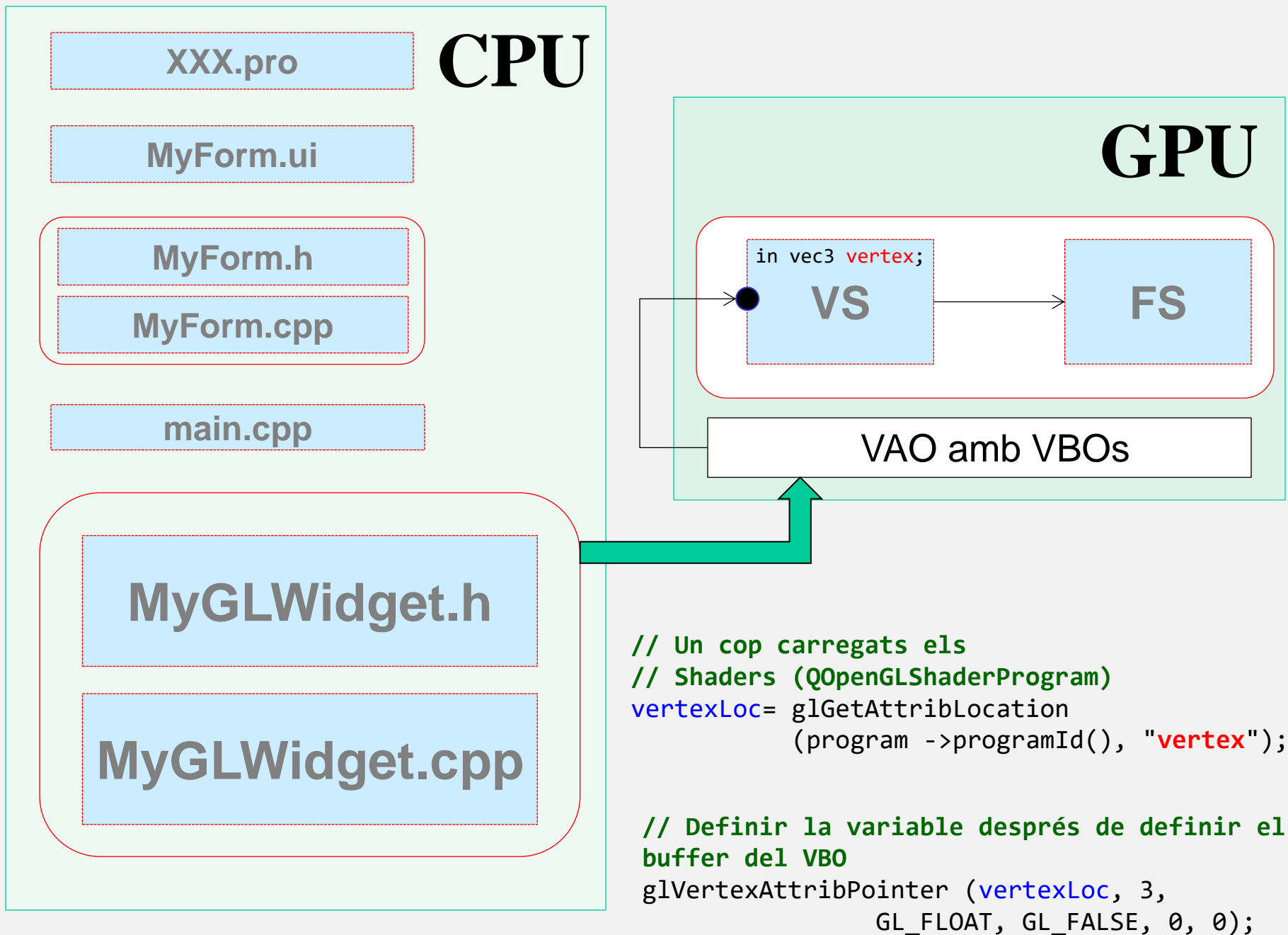


Laboratori OpenGL – Sessió 1.3

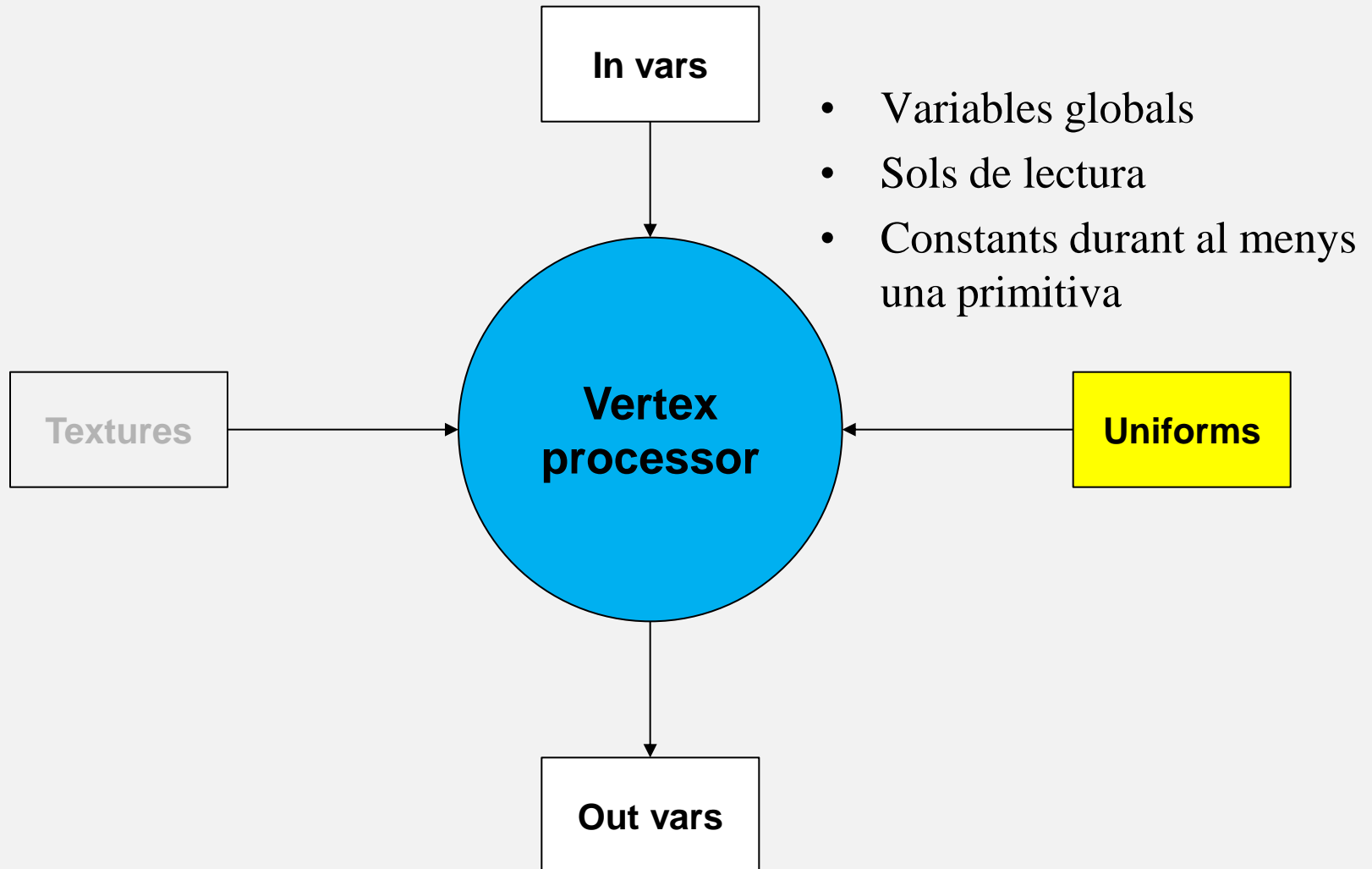
- Ús de *uniforms*
- Interacció directa a Qt
- Transformacions Geomètriques amb glm

Paradigma projectiu bàsic amb OpenGL 3.3





Uniforms



Uniforms

- All vertex shader:

```
#version 330 core
```

```
in vec3 vertex;
```

```
uniform float val;
```

```
void main ()  
{
```

```
    gl_Position = vec4 (vertex * val, 1.0);
```

```
}
```

CPU

XXX.pro

MyForm.ui

MyForm.h

MyForm.cpp

main.cpp

float **scl**;

MyGLWidget.h

MyGLWidget.cpp

GPU

```
in vec3 vertex;  
uniform float val;
```

VS

```
uniform float val;
```

FS

VAO amb VBOs

```
// Definir la variable un cop carregats els  
// Shaders (QOpenGLShaderProgram)  
varLoc= glGetUniformLocation  
        (program->programId(), "val");
```

```
// Enviar el valor de l'uniform  
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);  
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:

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 ();
}
```

MyGLWidget.h

Caldrà afegir:

```
#include <QKeyEvent>
```

i declarar el mètode virtual

```
virtual void keyPressEvent
(QKeyEvent *e);
```

Interacció directa amb Qt

MyGLWidget.cpp

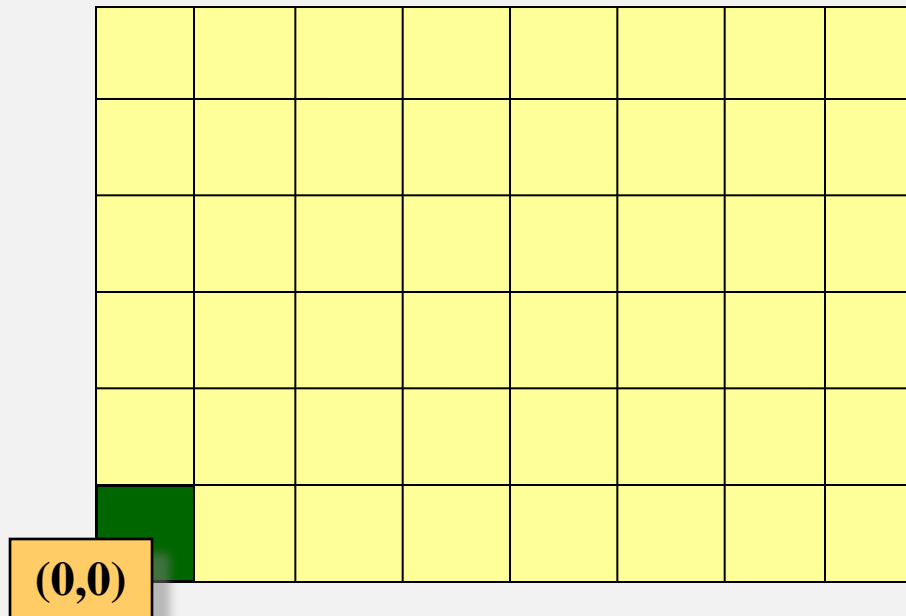
```
void MyGLWidget::keyPressEvent (QKeyEvent *e) {  
    makeCurrent ();  
    switch ( e->key() ) {  
        case Qt::Key_S :  
            scl += 0.1;  
            glUniform1f (varLoc, scl);  
            break;  
        case Qt::Key_D :  
            scl -= 0.1;  
            glUniform1f (varLoc,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

Cal cridar a *makeCurrent* quan hem de fer crides a **OpenGL** que no estiguin dins de *paintGL()*, *resizeGL()*, o *initializeGL()*

Update() força el repintat de l'escena

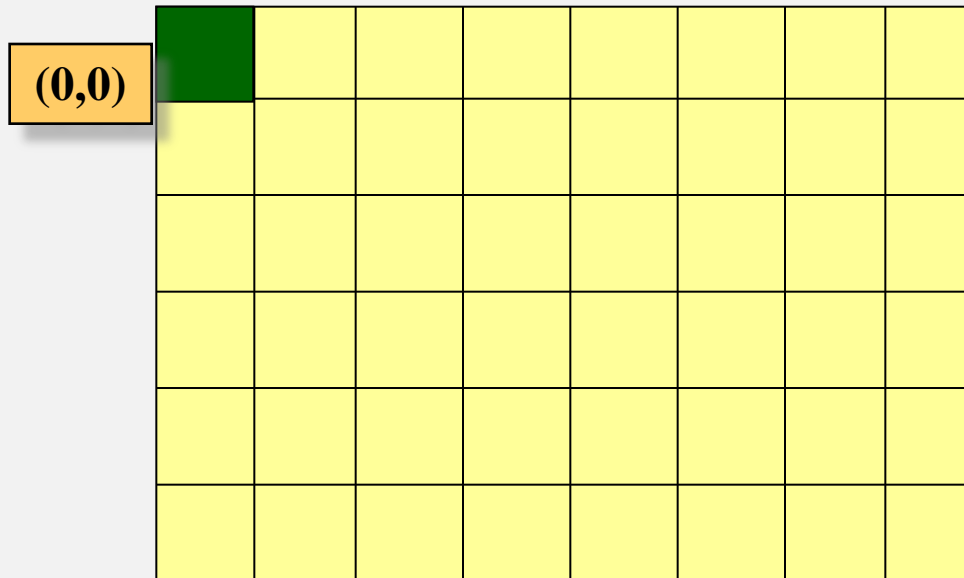
Consideració important

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



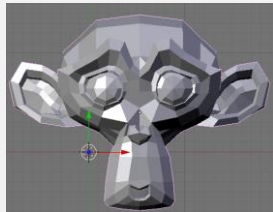
Consideració important

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

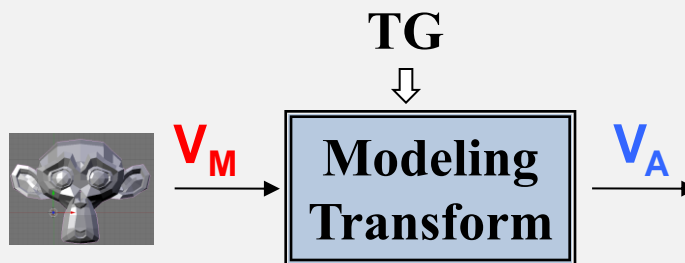
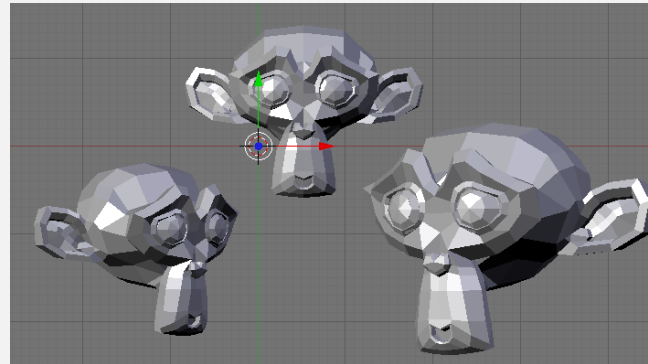


Matrius de transformació

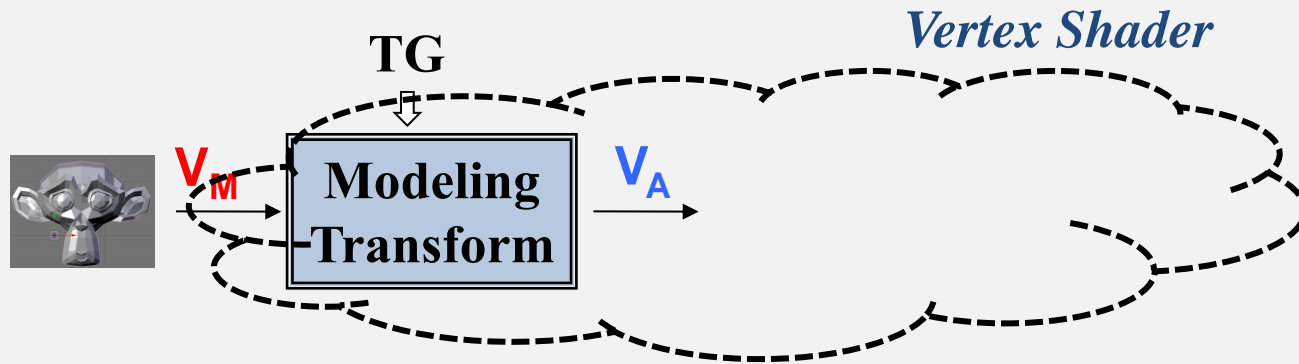
- Hem de poder transformar els vèrtexs (p.ex., amb transformacions de model):



$M = I,$
 TG_1, TG_2



Matrius de transformació



- Cal passar la matriu al Vertex Shader (com a uniform):

```
in vec3 vertex;  
uniform mat4 TG; ←  
void main () {  
    gl_Position = TG * vec4 (vertex, 1.0);  
}
```



- I lligar el uniform al nostre programa (en MyGLWidget):

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

Matrius de transformació

- Usarem glm per construir la matriu de transformació:
- Exemple:

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

- I completant l'exemple, per pintar farem:

```
void MyGLWidget::paintGL () {  
    glClear (GL_COLOR_BUFFER_BIT);    // Esborrem el frame-buffer  
  
    modelTransform (); // Abans del Draw !  
    glBindVertexArray (VAO1);  
    glDrawArrays (GL_TRIANGLES, 0, 3);  
    glBindVertexArray (0);  
}
```

Matrius de transformació

- Mètodes de transformacions geomètriques de la glm:

```
glm::translate (glm::mat4 m_ant, glm::vec3 vec_trans);  
    // retorna el producte de m_ant per una matriu que fa una  
    // translació pel vector vec_trans  
  
glm::scale (glm::mat4 m_ant, glm::vec3 vec_scale);  
    // retorna el producte de m_ant per una matriu que fa un  
    // escalat en cada direcció segons els factors vec_scale  
  
glm::rotate (glm::mat4 m_ant, float angle, glm::vec3 vec_axe);  
    // retorna el producte de m_ant per una matriu que fa una  
    // rotació de angle graus/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:

- 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.
- 2) 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 el que es feia al shader i construir la TG adient.
 - L'exercici 5 demana un escalat no uniforme. Podeu fer-lo també usant el moviment del ratolí, de manera que quan el ratolí es mou d'esquerra a dreta s'incrementa el factor d'escala en X (i es decrementa en anar de dreta a esquerra), i quan el ratolí es mou de baix a dalt s'incrementa el factor d'escala en Y (i es decrementa en anar de dalt a baix).

Exercicis sessió 3

3) Exercici extra:

- Afegiu al vostre codi un uniform *vp*, de tipus *ivec2* que representi les dimensions del *viewport* (en píxels) que ens indica el mètode `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.