

Sessió OpenGL

- Introducció a OpenGL
- Integració d'OpenGL amb Qt
- Visualització amb OpenGL
- Primitives gràfiques
- Transformacions geomètriques
- Matrius modelview i projection
- Pila de matrius
- Operacions amb matrius OpenGL
- Transformacions de vista
- Exemple a modificar

Intro a OpenGL

- API per visualització de gràfics 3D
 - Només visualització 3D
 - Cap funció de gestió d'entrada/events
 - Cap funció de gestió de finestres
 - OpenGL = GL core + GLU
- Aspectes bàsics
 - A cada frame es redibuixa tota l'escena.
 - Animació via doble-buffering
 - Màquina d'estats
- Tres tipus de funcions
 - Dibuix de primitives gràfiques
 - Modificació del *context gràfic* (variables que determinen com es dibuixen les primitives)
 - Consulta de valors de context gràfic.

Integració amb Qt

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

Mètodes virtuals a 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 per redefinir l'àrea de dibuix (viewport).
 - Qt la cridarà quan es creï la finestra, i cada cop que es canviï la mida de la finestra.

Integració amb Qt (ii)

```
class MyGL : public QGLWidget
{
    Q_OBJECT

public:
    MyGL( QWidget *parent):QGLWidget(parent) {}

protected:
    void initializeGL() {
        glClearColor( 0, 0, 0, 0);
        ...
    }

    void resizeGL( int w, int h ) {
        glViewport( 0, 0, w, h );
        ...
    }

    void paintGL() {
        // dibuixar l'escena
    }
};
```

Integració amb Qt (iii)

- Per tal de tractar un event de baix nivell cal re-implementar el mètode virtual corresponent de QWidget.
- Mètodes virtuals QWidget:

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

...

- Exemple d'implementació:

```
void MyGL::keyPressEvent (QKeyEvent *e) {  
    switch (e->text () [0] .toAscii ()) {  
        case 'X' :  
            rotx += 5;  
            updateGL ();  
            break;  
        default: e->ignore (); // propagar pare  
    }  
}
```

Visualització OpenGL

```
void MyGL::paintGL(void) {
    // 1. Netejar finestra
    glClear(GL_COLOR_BUFFER_BIT | ...);

    // 2a. Model-view Transformation
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(X, Y, Z, VRPX, VRPY, VRPZ, uX, uY, uZ);

    // 2b. Projection Transformation
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(fovy, ratio, near, far); //o
    gluOrtho(left, right, bot, top, near, far);

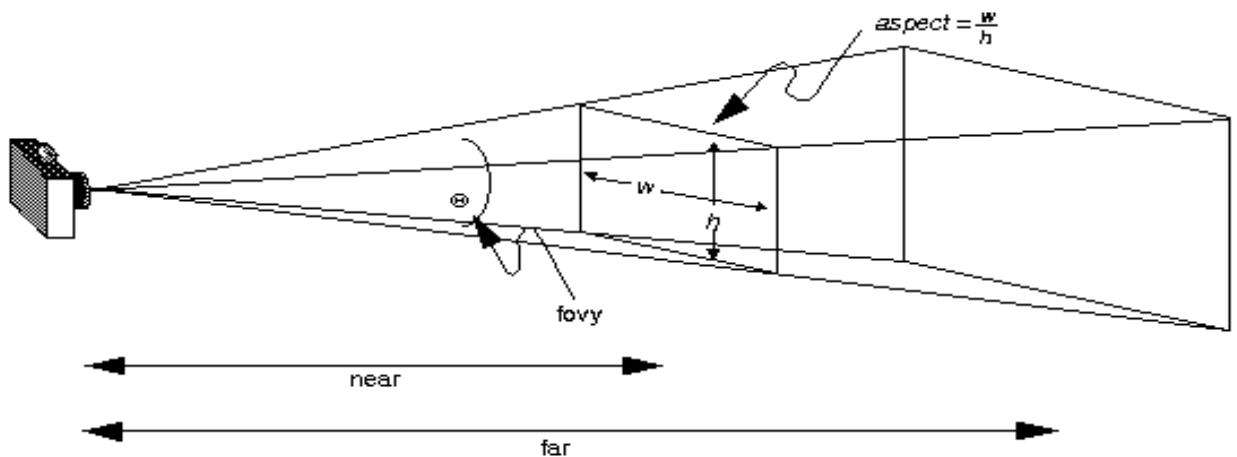
    // 3. Recorregut de l'escena
    per cada primitiva de l'escena {
        glBegin(GL_POLYGON);
        glVertex3f(v.x, v.y, v.z);
        glVertex3f(w.x, w.y, w.z);
        glVertex3f(u.z, u.y, u.z);
        glEnd();
    }
}
```

} Laboratori VIG. Primera sessió d'OpenGL

Tipus de càmera

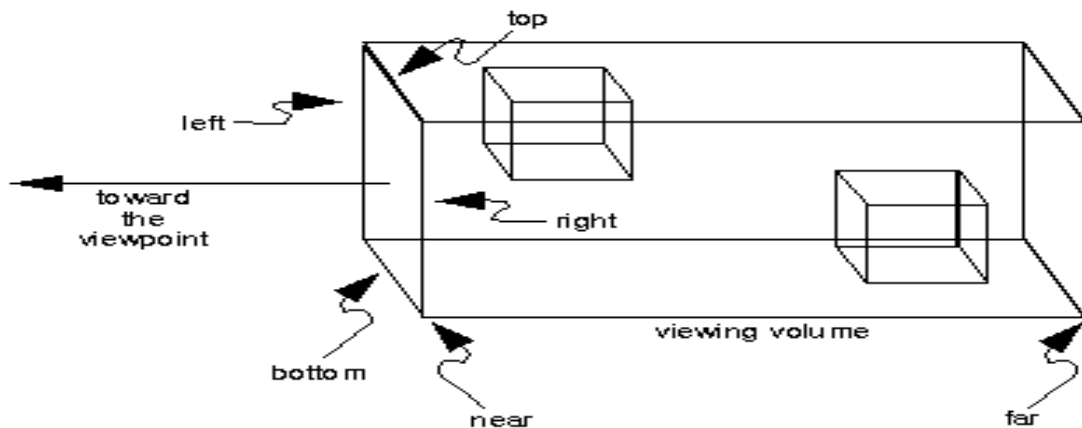
void **gluPerspective**(

GLdouble *fovy*, GLdouble *aspect*,
GLdouble *zNear*, GLdouble *zFar*);



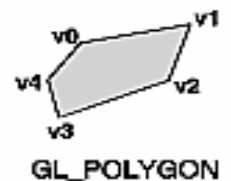
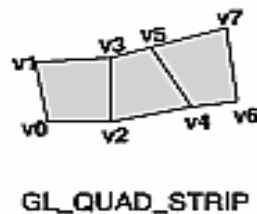
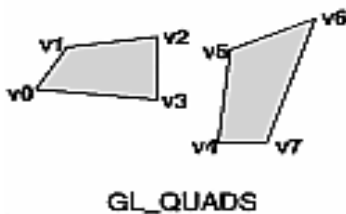
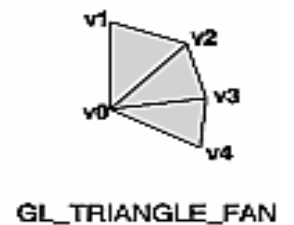
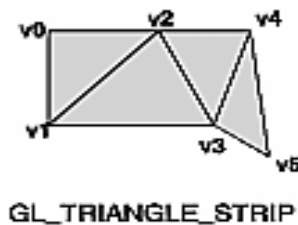
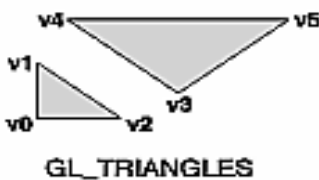
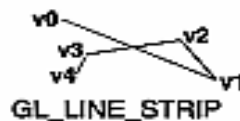
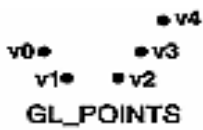
void **glOrtho**(

GLdouble *left*, GLdouble *right*, GLdouble *bottom*,
GLdouble *top*, GLdouble *near*, GLdouble *far*);



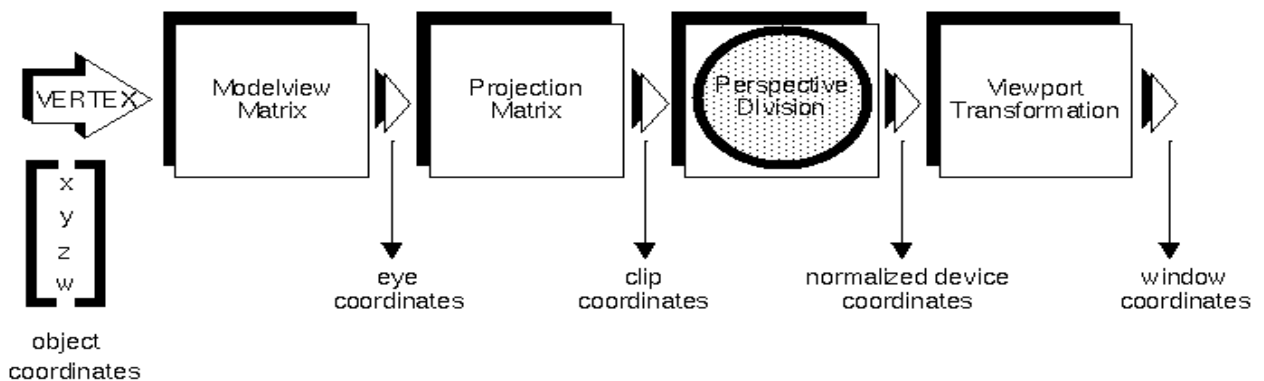
Primitives

```
glBegin(tipus de primitiva);  
glVertex3f(x1, y1, z1);  
glVertex3f(x2, y2, z2);  
glVertex3f(x3, y3, z3);  
...  
glEnd();
```



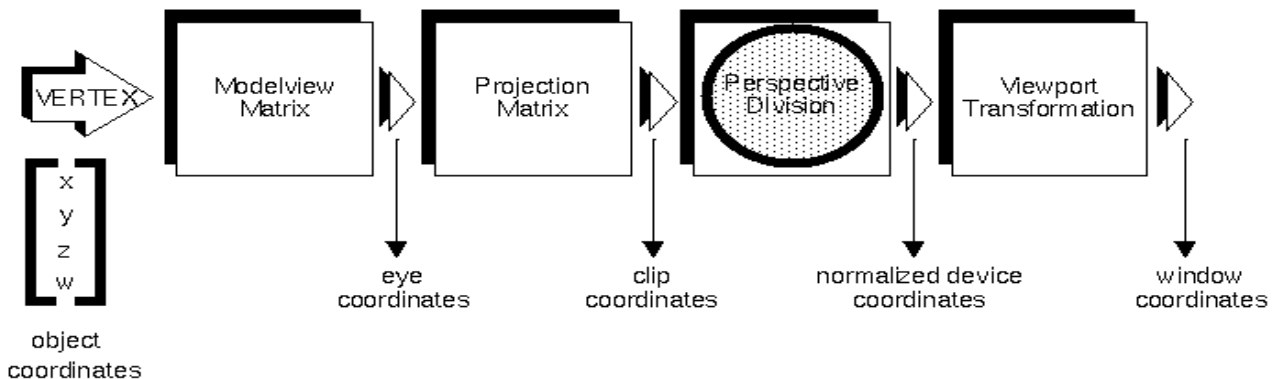
Transformacions

- OpenGL aplica als vèrtexs dues transformacions geomètriques en consonància amb la càmera definida.
- Aquestes transformacions estan representades per les matrius *Modelview* i *Projection*.
- La matriu *Modelview* és la que fa el pas de coordenades de món a coordenades de l'observador.
- La matriu *Projection* fa la projecció segons el tipus de càmera.



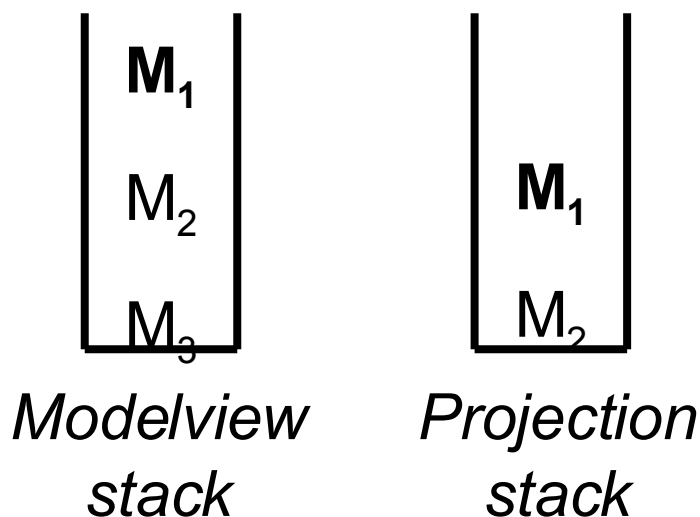
Modelview i Projection

- Definir el valor de cada matriu:
 - Indicar la matriu a modificar:
`glMatrixMode(GL_MODELVIEW);`
 - Indicar la matriu amb el qual s'ha de carregar (`glLoadIdentity`, `glLoadMatrix`) o s'ha de multiplicar (`glTranslate`, `glRotate`, `glScale`) la matriu activa.
- Aplicar les matrius als vèrtexs:
 - Automàtica: els vèrtexs enviats amb `glVertex` es multipliquen per la MODELVIEW i la PROJECTION



Pila de matrius

- Durant la visualització d'una escena, sovint ens pot interessar aplicar una matriu diferent a cada objecte.
- Per tant és útil poder guardar els valors d'una matriu per restaurar-los més endavant.
- Per això OpenGL manté *dues piles de matrius* (una per la modelview i una altra per la projection):



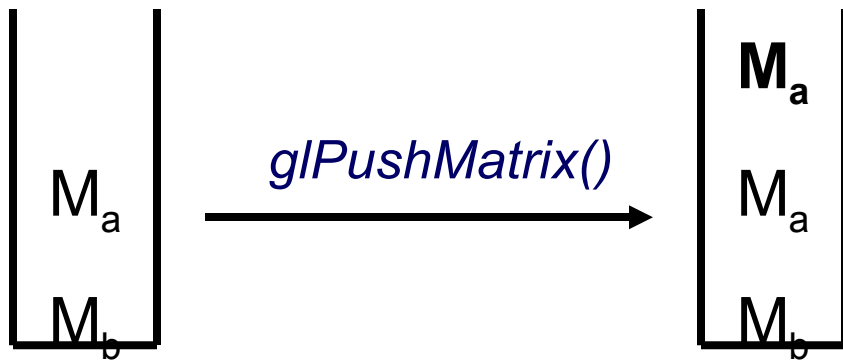
- OpenGL sempre aplica als vèrtexs la matriu del *top* de cada pila.

Operacions (i)

Primer cal indicar quina matriu volem modificar:

```
glMatrixMode(GL_MODELVIEW);  
glMatrixMode(GL_PROJECTION);
```

- Duplicar la matriu al top de la pila:



- Eliminar la matriu al top de la pila:

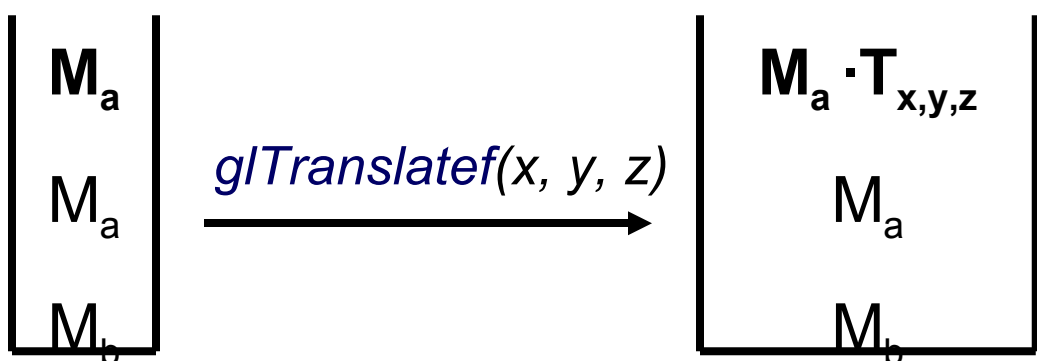


Operacions (ii)

- Substituir la matriu al top de la pila: (*glLoadIdentity*, *glLoadMatrix**). Exemple:



- Multiplicar la matriu al top de la pila per una altra matriu (*glTranslatef**, *glRotate**, *glScale**, *gluLookAt*, *gluPerspective...*). Exemple:



View transformations

- Aquestes transformacions corresponen a la definició de la posició i orientació de la càmera.

Amb gluLookAt:

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
gluLookAt(X,Y,Z, VRPX,VRPY,VRPZ, uX,uY,uZ);
```

Amb rotacions i translacions:

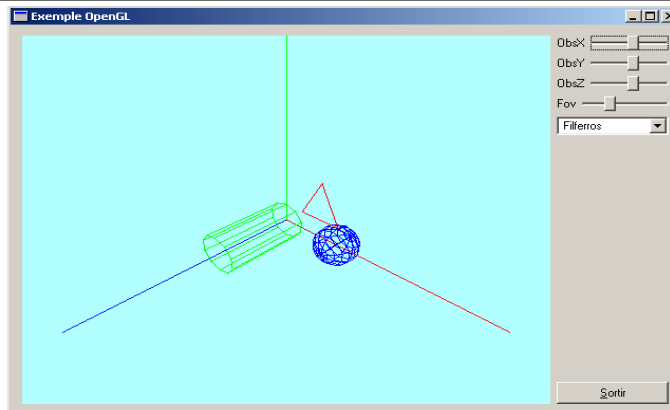
```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glRotatef(...)  
glRotatef(...)  
glRotatef(...)  
glTranslatef(...)
```

Model transformations

- Transformacions del model: corresponen al pas de coordenades locals de l'objecte (coordenades de modelat) a coordenades de l'aplicació.
- Les transformacions de modelat d'OpenGL s'han d'escriure en ordre invers ($p'=Mp$) i després de les view transformacions:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
// view transformation
gluLookAt(X,Y,Z, VRPX,VRPY,VRPZ, uX,uY,uZ);
Per cada objecte obj
{
    glPushMatrix();
    // modeling transformation
    glTranslatef(...);
    glRotatef(...);
    glScalef(...);
    Dibuixar(obj);
    glPopMatrix();
}
```

Exemple



mygl.h

Definició de la subclasse de QGLWidget.

mygl.cpp

Implementació de la subclasse.

mywindow.ui

Interfície generada amb QtDesigner

mywindow.h i mywindow.cpp

Implementats amb un editor de text.

openGLplugin.h i openGLplugin.cpp

Implementació del plugin

main.pro

Implementació del projecte.

gl.pro i openGLplugin.pro

Implementació dels projectes per a compilar l'aplicació i el plugin.

main.cpp

Implementació de main().