1. (1 punt) Imaginem que tenim definida una càmera perspectiva que permet veure tota l'escena. Com sabeu, per tal d'actualitzar adequadament la visualització de l'escena quan la finestra GLWidget pateix algun canvi de dimensions, s'ha d'actualitzar, si s'escau, l'angle d'obertura de la càmera (*fovy*). Completeu el següent mètode `setProjection()` per a implementar aquesta actualització si no volem deformacions i volem continuar veient tota l'escena en la vista.

```
void GLWidget::setProjection()
{
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     double ar = (double) width()/height();
     double fovy;

     if (ar>=1) fovy=anglecam;
     else {

     ...

     }
     gluPerspective(fovy,ar,znear,zfar);
}
```

```
double DEG2GRAD = M_PI/180.;
double RAD2DEG = 180./M_PI;
fovy = 2*atan(tan((anglecam/2)*DEG2RAD)/ar)*RAD2DEG;
```

2. (1 punt) Quan modifiquem el tipus de càmera de perspectiva a axonomètrica, ¿quines matrius del procés de visualització es modifiquen? Justifiqueu i argumenteu la vostra resposta explicant la missió de les dos matrius que s'utilitzen en el procés de visualització.

Les dues matrius del procés de visualització són la matriu de transformació de coordenades i la matriu de transformació de projecció. La matriu de transformació de coordenades permet fer el canvi de base per a què els vèrtexs de l'escena es referenciïn respecte del sistema de coordenades de l'observador. Aquest canvi és independent del tipus de càmera. La matriu de transformació de projecció s'encarrega de la projecció de cadascun d'aquests vèrtexs depenent del tipus de càmera que s'utilitzi (perspectiva o axonomètrica), donat que en un cas, perspectiva, es projecten en la direcció de la recta que uneix el vèrtex amb l'observador i en l'altre, axonomètrica, en la de la recta que passa pel vèrtex i és paral·lela a la direcció de visió (Zobs).
Per tant, només es modifica la matriu de projecció.

3. (1.5 punts) Tenim la informació geomètrica (cares i vèrtexs) d'un objecte bàsic i un mètode `PintaObjecte()` que implementa el seu recorregut enviant a pintar les primitives gràfiques corresponents. Coneixem els punts de coordenades mínimes (MinX,MinY,MinZ) i màximes (MaxX,MaxY,MaxZ) de la seva capsa mínima contenidora (amb les seves arestes orientades segons els eixos de coordenades). Volem visualitzar una instància d'aquest objecte tal que l'amplada (mida segons l'eix X) de la seva capsa sigui *AmplaX* (escalat uniforme) i el centre de la base d'aquesta estigui ubicat en el punt (10,0,10).
   Indiqueu:
   a) L'expressió de la composició de les transformacions geomètriques que cal efectuar a l'objecte bàsic per tal de visualitzar la instància utilitzant el mètode `PintaObjecte()`.
   b) El codi OpenGL que hauríeu d'implementar. Ha de quedar clarament indicat quin és el contingut de la matriu de ModelView en el moment que comença el codi que proposeu.

Per a ubicar i escalar l'objecte adequadament, traslladarem el centre de la base de la capsa contenidora a l'origen de coordenades, seguidament realitzarem un escalat uniforme respecte l'origen de valor AmpleX/(MaxX-MinX) i, per últim, traslladarem l'objecte a la posició final (com que tenim el centre de la base de la capsa a l'origen i aquest és el punt que volem situar al punt donat, directament farem la translació al punt donat).

a) Si considerem escalat = AmplaX/(MaxX-MinX) tenim que la composició de transformacions és:
*TG = T(10, 0, 10) * S(escalat, escalat, escalat) * T(-(MaxX+MinX)/2, -MinY, -(MaxZ+MinZ)/2)*

b) Suposem que el contingut de la matriu ModelView en aquest punt és la matriu que permet portar els vèrtexs del sistema de coordenades de l'aplicació al sistema de coordenades de l'observador, és a dir, en aquest punt del codi la matriu ModelView conté la transformació de coordenades:

```
escalat = AmplaX/(MaxX-MinX);
glPushMatrix ();
glTranslatef (10, 0, 10);
glScalef (escalat, escalat, escalat);
glTranslatef (-(MaxX+MinX)/2, -MinY, -(MaxZ+MinZ)/2);
PintaObjecte ();
glPopMatrix ();
```

4. (1 punt) Imaginem que enviem a pintar un triangle. Quins blocs funcionals del procés de visualització requereixen només les coordenades de vèrtexs i quins tota la informació del triangle (els tres vèrtexs ordenats)? Nota: òbviament, cada bloc funcional tindrà la informació geomètrica referida respecte del sistema de coordenades que li correspon.

Requereixen només les coordenades de vèrtexs:
- *La transformació de coordenades*
- *La transformació de projecció*
- *La transformació món-dispositiu*

Totes aquestes transformacions, es poden implementar multiplicant les coordenades dels vèrtexs per una matriu.

Requereixen tota la informació del triangle:
- *El retallat*
- *La rasterització*

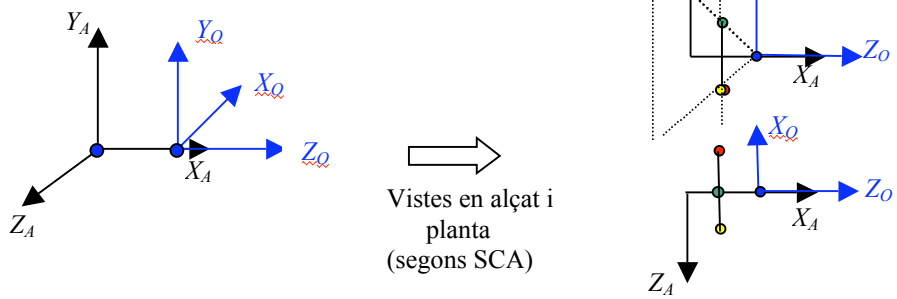
Aquests blocs s'implementen de manera específica d'acord amb la primitiva a tractar (triangle, segment, punt).

5. (1.5 punts) Donat el codi OpenGL següent:

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
gluPerspective (90, 1, 50, 150);
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
glTranslatef (0, 0, -100);
glRotatef (-90, 0, 1, 0);
glBegin (GL_POLYGON);
glVertex3f (50, -50, 50);  // vèrtex v1
glVertex3f (50, -50, -50); // vèrtex v2
glVertex3f (50, 50, 0);    // vèrtex v3
glEnd ();
```

Si tenim una vista que abarca del píxel (0,0) al (500,500), indiqueu quins són els valors dels vèrtexs v1, v2 i v3 en el sistema de coordenades d'observador (SCO), en el sistema de coordenades normalitzat (SCN) i en el sistema de coordenades del dispositiu (SCD).
Com generaríeu la mateixa matriu ModelView mitjançant una crida a gluLookAt? Justifiqueu la resposta.

a) El sistema de coordenades de l'observador és tal que el seu origen es troba al punt (100, 0, 0) i està orientat de manera que el seu eix X_O coincideix amb l'eix $-Z_A$ de l'aplicació, el seu eix Y_O coincideix amb l'eix Y_A de l'aplicació i el seu eix coincideix amb l'eix X_A de l'aplicació.
El triangle és troba en el pla x=50 (paral·lel al pla ZY de l'aplicació).
Com que l'angle d'obertura de la càmera és de 45° i la distància de l'observador al pla del triangle és 50, tenim que els vèrtexs queden respectivament a les cantonades esquerra-abaix, dreta-abaix i al centre de la part de dalt del window.



Per tant:
V1 en SCO = (-50, -50, -50); V1 en SCN = (-1, -1, -1) / (-1, -1, 1); V1 en SCD = (0, 0, 0);
V2 en SCO = (50, -50, -50); V2 en SCN = (1, -1, -1) / (1, -1, 1); V2 en SCD = (500, 0, 0);
V3 en SCO = (0, 50, -50); V3e n SCN = (0, 1, -1) / (0, 1, 1); V3 en SCD = (250, 500, 0);

Nota: La coordenada Z en el SCN serà -1 si s'ha considerat el volum normalitzat amb el Znear a -1 i serà 1 si s'ha considerat el volum normalitzat amb el Znear a 1.

b) D'acord amb la posició i orientació del SCO (veure dibuix) tenim que:
OBS = (100, 0, 0); VRP = (0, 0, 0); up = (0, 1, 0)
gluLookAt (OBS.x, OBS.y, OBS.z, VRP.x, VRP.y, VRP.z, up.x, up.y, up.z);

6. (1 punt) Quin bloc funcional genera els *fragments*? Quina informació té associada (es guarda en) un *fragment*? Indiqueu el domini (conjunt de valors) que pot prendre cada element d'informació. Justifiqueu la resposta.

Els fragments es generen amb la rasterització.
La informació associada a un fragment és: (xd, yd, zd, c), on (xd, yd) són les coordenades de dispositiu (píxel) en què es projecta un punt de la primitiva; zd és proporcional a la seva distància a l'observador, i c és el color.
- *xd ∈ [0, resolucióX-1] resolucióX és el nombre de píxels de la vista en amplada*
- *yd ∈ [0, resolucióY-1] resolucióY és el nombre de píxels de la vista en alçada*
- *zd ∈ [0, 2^{nz}-1] nz és el nombre de bits per a codificar la profunditat en cada píxel*
- *c és un color (r, g, b) on r, g, b ∈ [0, 2^{nc}-1] nc és el nombre de bits per a codificar cada color R, G, B*

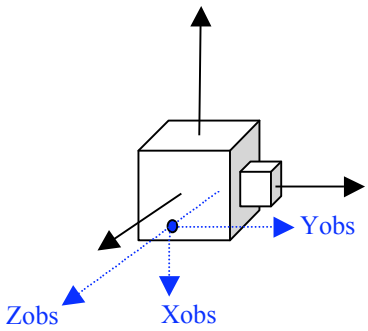
7. (1.5 punts) Una escena està formada per dos cubs, un de costat 20 centrat al punt (0,0,0), i l'altre de costat 10 centrat al punt (15,0,0). Indiqueu TOTS els paràmetres d'una càmera que permeti veure a la vista dos quadrats, un damunt de l'altre (el més gran a sota), de manera que ocupin el màxim de la vista (*viewport*). Cal que indiqueu la posició i orientació de la càmera especificant **VRP**, **OBS** i **up**.

Els dos cubs estan situats de costat respecte l'eix X_A de l'aplicació, de manera que el més petit està més a la dreta. Si volem veure dos quadrats, el millor és definir una càmera axonomètrica, perquè així no veurem la deformació de perspectiva. Si volem veure el quadrat petit damunt del quadrat gran (és a dir, volem veure el cub petit damunt del gran) hem de girar la càmera respecte de la direcció de visió, per tant hem de definir el vector up de manera que indiqui que la verticalitat va en direcció a la part positiva de l'eix X_A de l'aplicació.
La direcció de visió la podem definir perpendicular al pla XY de l'aplicació i ens interessa que el punt mig dels dos cubs ens quedi projectat al mig de la vista, per tant podem definir el VRP com el punt central entre els dos cubs i l'OBS en la direcció paral·lela a Z_A a una distància que permeti veure l'escena des de fora.

Els paràmetres de la càmera podrien ser:

OBS = (5, 0, 20);
VRP = (5, 0, 0);
up = (1, 0, 0);

left = -10
right = 10
bottom = -15
top = 15
znear = 10
zfar = 30



8. (1.5 punts). Completeu el següent codi de Qt per a crear un nou widget, anomenat *semàfor*, que ha de tenir un slot per a canviar de color (de manera similar a com ho veu fer en la Pràctica_0).
En una interfície es dessitja que al fer un "clic" en un *boto_1* (de tipus *QPushButton*) de la interfície, es modifiqui el color del *semàfor*. Indiqueu les connexions que necessitaríeu a efectes d'obtenir aquest comportament.

```
#include <qpushbutton.h>
class MyButton : _ public _ _ QPushButton _
{
  _ Q_OBJECT _

  int color; // 0 - verd, 1 - groc, 2 - vermell

  public:
     MyButton (QWidget * parent, const char * name = 0);

  public slots:
  _ void CanviaColor(); _

};
```

Com que volem que en fer un "clic" en el boto_1 el semàfor canviï de color i hem definit un slot en MyButton (`CanviaColor()`) que farà aquest canvi de color, l'únic que ens cal és connectar el signal `clicked()` del boto_1 amb aquest nou slot `CanviaColor()` del semàfor:

```
connect (boto_1, SIGNAL(clicked()), semàfor, SLOT(CanviaColor()));
```