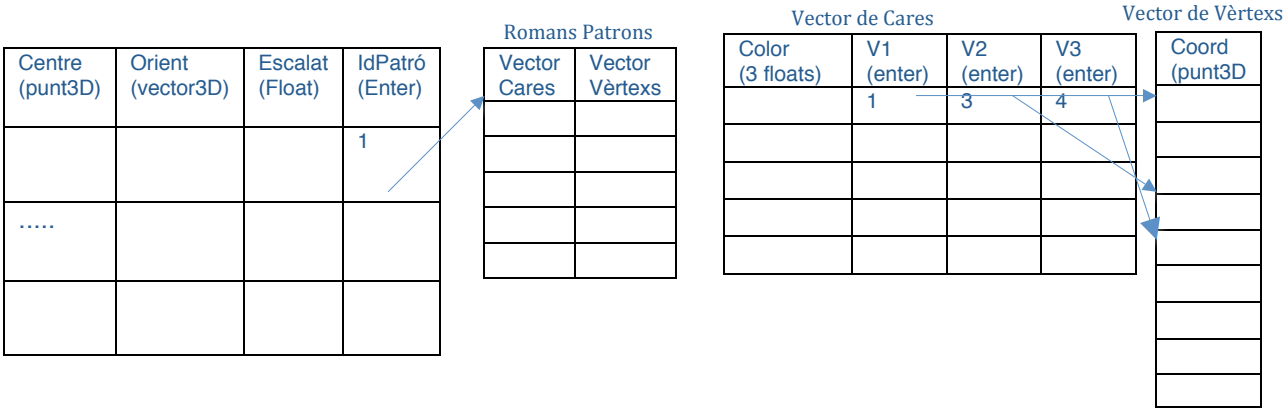
VIG - EXAMEN PARCIAL- QUADRIMESTRE PRIMAVERA- 2008-2009 TEMPS: 1h 30m

==========

1. (1.5 Punts) Un joc de computadors es basa en la recreació d'una ciutat romana. Es disposa del model de fronteres de 5 romans amb diferent indumentària, els anomerarem "romans patró". En l'escena poden haver-hi fins a 100 romans de cada tipus (és a dir, fins a 100 de cada "patró"). La ubicació de cada romà queda determinada pel centre de la seva capsa mínima contenidora, la seva orientació per un vector i la seva grandària serà la del romà patró corresponent escalat un cert %. Es demana el disseny d'una estructura de dades compacta que permeti emmagatzemar les dades requerides per a la visualització de la població dels romans de la ciutat. El model de fronteres dels romans "patró" està constituït per una malla de triangles on cada triangle pot ser d'un color diferent.

Una estructura apropiada per a representar la població és tenir una llista d'instàncies a romans patrons.
Per a emmagatzemar els romans patrons tindrem un vector de 5 components. Per cada romà patró (component) guardarem el seu model de fronteres que consisitirà en un vector de triangles i altre vector amb les coordenades (x,y,z) dels vèrtexs de les cares. Per cada trinagle guardarem els identificadors dels seus vèrtexs (components del vector de vèrtexs) i el seu color RGB.
Cada instància tindrà com a atributs la informació requerida per a la seva ubicació espaial i l'identificador del romà patró associat (component del vector de patrons).
Esquemàticament,



2. (1 Punt) Donat el color codificat en HSB com (0, 1, 0.5), contesta **raonadament** les següents preguntes: Quina seria la seva codificació en RGB? I en CMY?

Com H=0 és tracta d'un color amb tonalitat principal vermella. Com S=1 és un color saturat (pur); per tant, no té ni verd ni blau. Com B=0, la brillantor es mitjana. Per tant, es tracta d'un vermell de mitjana intensitat. RGB= (0.5,0,0).
El CMY serà el color complementari: (1-0.5, 1-0 ,1-0)=(0.5,1,1)

3. (2.5 Punts) Tenim una escena formada per un sol segment definit entre els punts (0,0,3) i (0,0,-3). Volem visualitzar la totalitat d'aquest segment en un *viewport* de 300x300 píxels usant una càmera perspectiva de manera que el segment travessi el *viewport* diagonalment des de la cantonada superior esquerra fins a la cantonada inferior dreta. Es demana:

- El tros de codi d'OpenGL requerit per a definir **tots** els paràmetres d'una càmera que produeixi la visualització indicada. La ubicació de la càmera s'ha de definir mitjançant transformacions geomètriques. Justifiqueu els procediment per decidir els valors escollits.
- Les coordenades d'observador, normalitzades i de dispositiu del vèrtex (0,0,3). Justifiqueu la resposta.

Donat que el segment (0,0,3)-(0,0,-3) es troba sobre l'eix de les Z, la visió des de la posició de l'observador ha de ser perpendicular a aquest eix per a què el segment es vegi com a tal. Decidim situar l'observador sobre l'eix de les X i mirant cap a l'origen de coordenades que és el centre del segment.

Un cop ubicada la càmera, per a què el segment es vegi creuant el window en diagonal de la cantonada de dalt a l'esquerra a la cantonada de baix a la dreta amb el punt (0,0,3) en la cantonada de dalt a l'esquerra, ens cal girar el vector vertical de la càmera 45º (sentit antihorari). Per tant, les transformacions a aplicar al sistema de coordenades de la càmera són: 1) gir sobre y de 90º; 2) gir sobre Z de 45º; i translació sobre Z de (0,0,d).

*La distància **d** a la que situem l'observador de l'origen de coordenades estarà lligada al valor que li donem a l'angle d'obertura de la càmera. Com que el segment que creua el window medeix 6 i el window és quadrat, el costat d'aquest quadrat (mida del window) és de **arrel(18)** o també **3*arrel(2)**. Si decidim posar un angle **α** de 45º tindrem que la distància d serà de **3*arrel(2)/2**, i l'angle d'obertura (FOV) serà de 2*α=90º.*

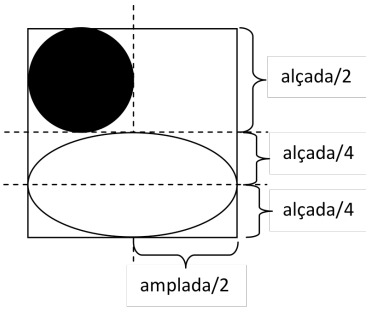
La relació d'aspecte pot ser la del viewport, i posarem com Znear la distància de l'observador al segment, i el Zfar una mica més lluny. Així doncs, el codi OpenGL per a inicialitzar aquesta càmera és el següent:

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
gluPerspective (90, 1, 3*arrel(2)/2, 3);
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
glTranslatef (0, 0, -3*arrel(2)/2);
glRotatef (-45, 0, 0, 1);
glRotatef (-90, 0, 1, 0);
```

Amb aquesta càmera, el punt (0,0,3) en SCA, tindria les següents coordenades en els altres sistemes:

- *En SCO seria el **(-3*arrel(2)/2, 3*arrel(2)/2, -3*arrel(2)/2),** perquè està en la cantonada de dalt a l'esquerra del window que fa 3*arrel(2) de costat i sobre el Znear que està allunyat de l'observador 3*arrel(2)/2.*
- *En SCN seria el **(-1, 1, -1)**, tenint en compte que el nostre espai normalitzat posa Znear a -1.*
- *I en SCD seria el **(0, 300, 0),** ja que l'origen del viewport està en la cantonada de baix a l'esquerra i aquest fa 300 x 300 pixels. La zd és 0 perquè el segment està sobre el Znear.*

3. (2 Punts) Tenim un mètode de visualització que defineix la càmera de la següent forma:




```
glMatrixMode (GL_PROJECTION);
glLoadIdentity();
glOrtho (-20,20,-20,20,5,25);
glMatrixMode (GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0,10,0,0,0,0,-1,0,0);
<codi Nou>
```

Tenim un mètode que pinta una esfera negra de radi 1 centrada a l'origen, anomenat *pintaEsferaNegra()*, i altre mètode que pinta una esfera blanca de radi 1 centrada a l'origen, i que s'anomena *pintaEsferaBlanca()*. Volem que el resultat de la visualització (en el viewport) sigui l'indicat en la figura (les línies discontínues i el text només pretenen servir-vos de guia; l'alçada i amplada es refereixen a les mides del *viewport* que es quadrat).
Escriviu, i justifiqueu, el codi que cal afegir a partir de <codi Nou> per a aconseguir la visualització que es demana. *Observació*: La càmera definida **no** es pot modificar.

*La càmera està posicionada de manera que l'observador es troba sobre l'eix Yapl, mirant cap a l'origen de coordenades i amb el vector **up** en la direcció de -Xapl, els eixos del sistema de coordenades de l'observador queden, per tant, orientats de manera que Xobs = –Zapl; Yobs = -Xapl; i Zobs=Yapl; i el window es troba en el pla Yapl=5, les seves mides són 40x40 i té la mateixa relació d'aspecte que el viewport.*

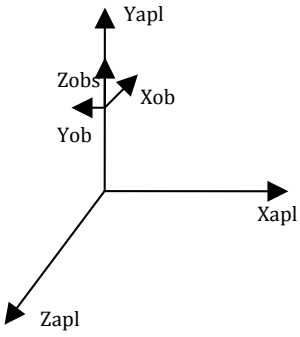
El diàmetre de l'esfera negra ocupa la meitat de l'amplada del viewport i, per tant, del window; per això val 20. El seu centre es projecta en el centre del quadrant superior-esquerra del window; per tant, tindrà coordenades d'observador (-10,10). Com l'esfera no és retallada, el seu centre estarà a una distància de, com a mínim, 10 del window; per tant, la Zobs del centre >= -15 (Yapl>=-5). Per tant, a l'esfera negra se li ha d'aplicar un escalat uniforme de valor 10 i traslladar el seu centre a (-10,-5,10).

El centre de l'esfera blanca es projecta en el punt (0,-10) del window en coordenades d'observador. La seva Zobs també serà -15 pel mateix motiu que l'altra esfera. Respecte la direcció Zapl (Xobs) se li ha aplicat un escalat el doble que respecte Xapl (Yobs). L'escalat respecte Yobs serà de 10 (mateix motiu que l'esfera negra). Per tant, a l'esfera blanca se li ha d'aplicar un escalat (10,10,20) i transladar a (10,-5,0)

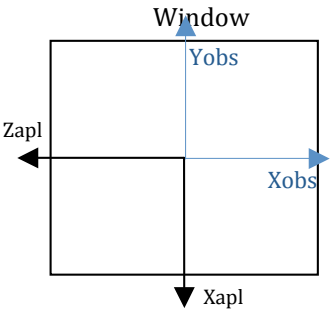
Així doncs, el codi que cal posar a partir de <codi Nou> serà el següent:

```
glPushMatrix ();
glTranslatef (-10, -5, 10);
glScalef (10, 10, 10);
pintaEsferaNegra ();
glPopMatrix ();
glPushMatrix ();

glTranslatef (10, -5, 0);
glScalef (10, 10, 20);
pintaEsferaBlanca ();
glPopMatrix ();
```

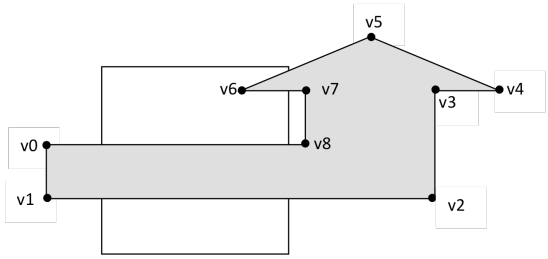


(a)



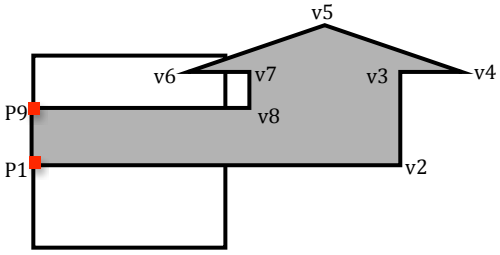
(b)

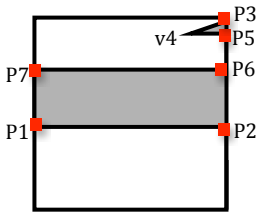
4. (1punt) Indiqueu com s'executaria el retallat del següent polígon i quin resultat donaria. Executeu i expliqueu l'algorisme pas a pas, no poseu només el resultat final.



El retallat de polígons és un procés iteratiu que compara la posició relativa de les arestes del model amb cadascun dels 6 plans que limiten el Volum de Visió. Per a cada pla, el que es fa és mirar, per cadascuna de les arestes del model, si els dos vèrtex estan a dins o fora respecte el pla, i en funció de les diferents combinacions es calculen els vèrtexs (i, implicitament, les arestes) del poligon retallat.

Aplicant l'algoritme mencionat, retallarem el polígon anterior. Només mostrarem el retallat respecte els plans (esquerra, dret, superior, inferior) del Volum de Visió. Per tal de facilitar la lectura, pintem al dibuix els vèrtex que es generen amb un quadradet vermell. El subíndex indica l'ordre en que quedaríen els vèrtexs en el polígon retallat. Després del segon retallat (en l'ordre escollit) el polígon no és modifica.





1 únic Polígon amb vèrtexs seguint l'ordre:
P1,P2,P3,V4,P5,P6,P7

5. (2 Punts) Volem dissenyar amb Qt4.0 una interfície (veieu la figura) per a una calculadora que permeti calcular potències de 2. Per fer això necessitem fer servir dos tipus de *widgets*:

- Un *QLCDNumber*: per fer la pantalla que ha de mostrar el número.
- Dos *QPushButtons*: un per multiplicar per 2 el número que hi ha a la pantalla, i un altre per dividir-lo per 2.

El valor inicial de la calculadora és 1. Cada vegada que es prem el botó de multiplicar per 2, el número mostrat a la pantalla es multiplica per 2. El botó de dividir per 2 funciona de manera anàloga. Quan el número mostrat a la pantalla és 1, la calculadora no permet dividir més per 2 (el botó queda desactivat, veure figura esquerra). Quan el número mostrat és 2 elevat a 30, el botó també queda desactivat. En qualsevol altre cas, es poden realitzar totes dues operacions.

Indiqueu clarament i detalladament com implementaríeu aquesta interfície per tal que tingui l'aparença i comportament especificats. Concretament, especifiqueu en detall els *widgets* propis que necessiteu definir (si s'escau). Indiqueu també clarament quins *signals* i *slots* necessiten aquests nous *widgets* i les connexions entre ells

Hi ha diverses possibles solucions. Una de les més senzilles és la següent: es necessita fer un nou widget ***meuLCDNumber*** *que heredi de QLCDNumber que tingui els següents slots:* ***multiplicaPerDos()*** *i* ***divideixPerDos()****. I els següents signals:* ***activatInferior(bool)*** *i* ***activatSuperior(bool).***

L'slot ***multiplicaPerDos()*** *agafa el valor del LCD i el multiplica per 2. Això serà sempre possible, ja que una vegada fet això, comprovarà si el valor és igual a 2^30 i si és així, emetrà un signal (activatSuperior) que permet desactivar el botó de multiplicar. El de dividir funciona de forma l'anàloga. Els slots han de fer el control de les fites i emetre els signals corresponents, com es veu a continunació:*

```
void meuLCDNumber::multiplicaPerDos()
{
    long val = 2*intValue();
    emit activatSuperior (val != 2^30);
    emit activatInferior(true);
    display(val);
}

void meuLCDNumber::divideixPerDos()
{
    int val = intValue()/2;
    emit activatSuperior (true);
    emit activatInferior(val != 1);
    display(val);
}
```

Inicialment s'ha de posar el valor a 1:

```
meuLCDNumber::meuLCDNumber(QWidget* parent):QLCDNumber(parent)
{
  display(1);
}
```

També cal desactivar el botó de dividir, es pot fer amb una funció d'inicialització per garantir que l'emissió del signal corresponent es realitzi quan l'objecte QPushButton que representa al botó de dividir estigui efectivament creat:

```
void meuLCDNumber::init()
{
  emit activatInferior(false);
}
```

Suposant que el botó de multiplicar es digui mul2, que el de dividir tingui com a nom div2 i que l'objecte que instancia la meva classe derivada de QLCDNumber es diu qlcd, podríem establir les connexions de la següent forma:

```
connect(mul2, SIGNAL(clicked()), qlcd, SLOT(multiplicaPerDos()));
connect(div2, SIGNAL(clicked()), qlcd, SLOT(divideixPerDos()));
connect(qlcd, SIGNAL(activatInferior(bool)), div2, SLOT(setEnabled(bool)));
connect(qlcd, SIGNAL(activatSuperior(bool)), mul2, SLOT(setEnabled(bool)));
```