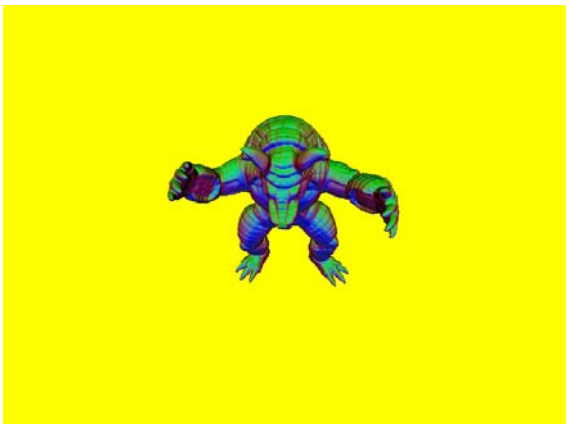
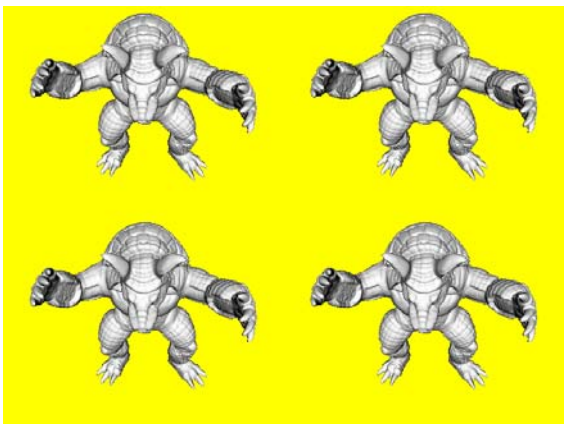
Quads (quads.*) 2.5 punts

Escriu **VS+GS+FS** que dibuixin cada triangle del model quatre cops; un a cada quadrant de la finestra. Aquí teniu un exemple, amb els shaders per defecte (*esquerra*) i amb els shaders que es demanen (*dreta*):

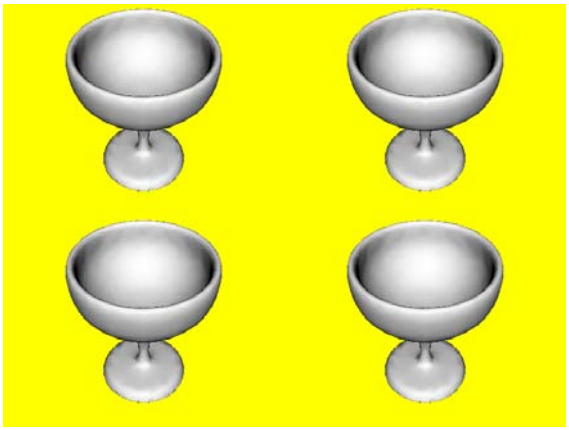
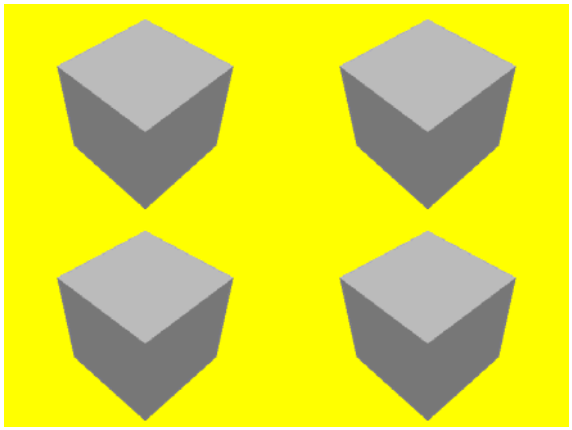



El **VS** haurà d'escriure gl_Position com habitualment. El **color del vèrtex** tindrà per components RGBA la **component z de la normal en eye space.**

El **GS** haurà d'emetre quatre còpies de cada triangle. Aquest exercici és senzill si treballeu en **NDC** (Normalized Device Coordinates). La única diferència entre les dues còpies és la translació en X i Y (en NDC), que serà -0.5 o 0.5. És obligatori que apliqueu la translació en NDC.

El **FS** simplement escriurà el color que li arriba del VS.

Aquí teniu més exemples:

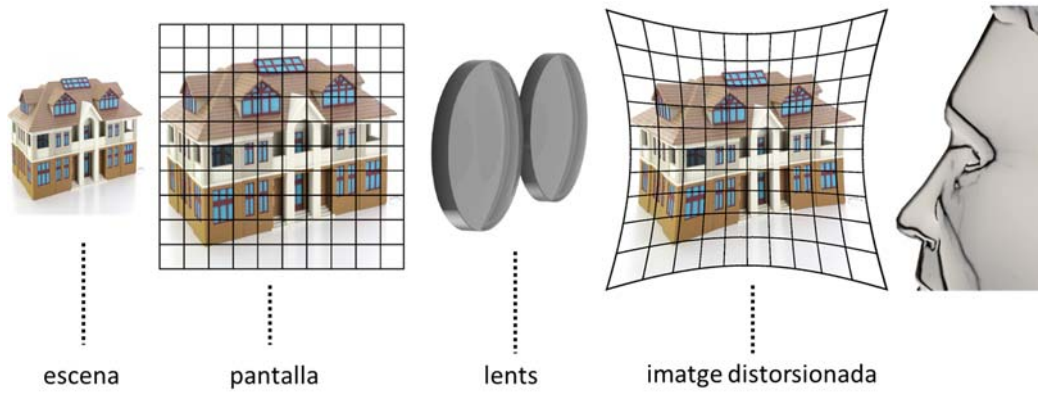



Fitxers i identificadors (ús obligatori):

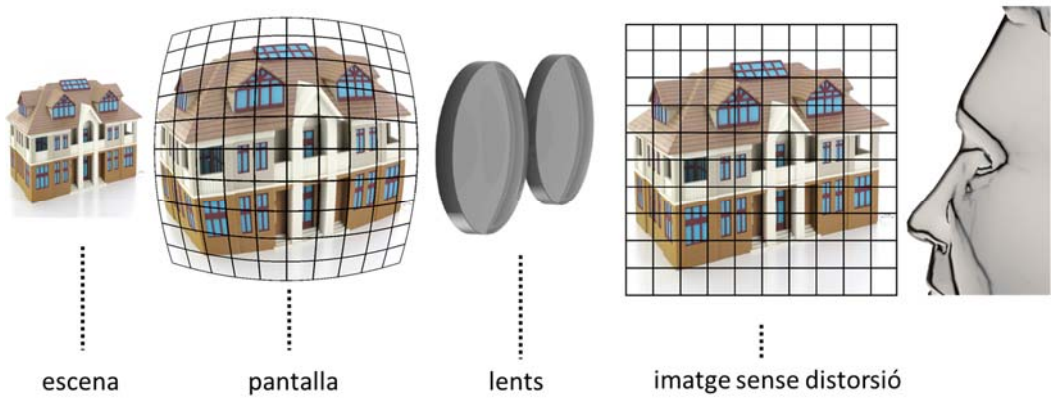
`quads.vert, quads.geom, quads.frag`

Undistort (undistort.*) 2.5 punts

Les ulleres de realitat virtual (Oculus Rift, GearVR...) fan servir lents que produeixen una distorsió radial a les imatges. Si la imatge que es mostra a la pantalla del dispositiu no es corregeix, l'usuari veurà la imatge distorsionada:

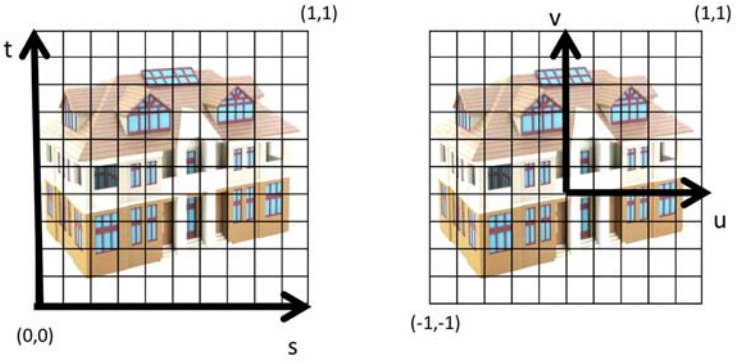


La solució més habitual consisteix a aplicar la distorsió inversa a la imatge que volem mostrar, de forma que les lents desfan aquesta distorsió i l'usuari veu la imatge correcta:



Escriu un VS i un FS que apliquin una distorsió radial. Aquest problema està pensat per l'objecte Plane, que té coordenades de textura (s,t) dins l'interval [0,1]. La imatge que volem distorsionar la carregarem com una textura **uniform sampler2D colorMap**.

El VS farà les tasques per defecte. El FS accedirà a la textura amb unes coordenades (s,t) modificades per tenir en compte la distorsió radial. El primer que haureu de fer és obtenir les coordenades de textura (u,v) respecte uns eixos centrats a l'espai de textura, com es mostra a la figura.



Sigui Q=(u,v) el punt amb les coordenades de textura del fragment, i sigui r la longitud del vector posició de Q. La distorsió radial que volem (inspirada en la del Oculus Rift DK1) consisteix a calcular un nou punt Q'=(u',v') amb un nou mòdul r' calculat com:

$$r' = (r + 0.22r^3+0.24r^5)$$

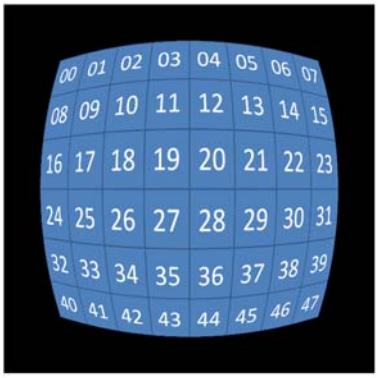
La distorsió només canvia la distància a l'origen, i per tant els vectors posició de Q i Q' són paral·lels, amb Q' = r' * normalize(Q).

Un cop calculats (u',v'), heu de calcular les coordenades finals de textura (s', t') desfent el canvi de coordenades del començament.

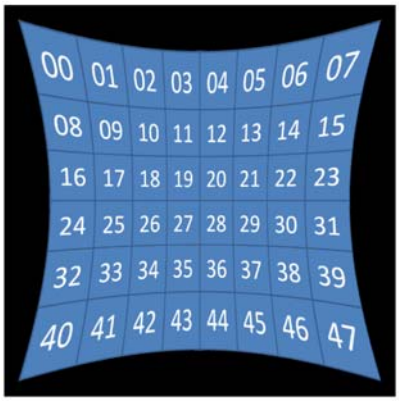
Si s' i t' pertanyen a l'interval [0,1], el color final del fragment serà el color de la textura al punt (s',t'). Altrament el color serà negre.

Aquí teniu alguns exemples (textura carregada i imatge esperada):

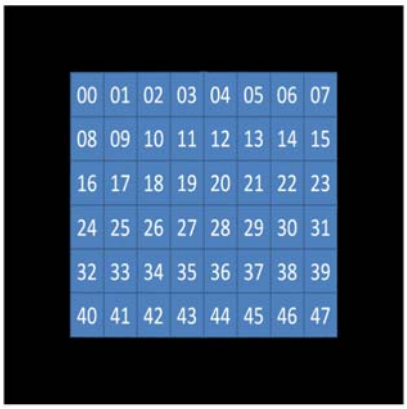
frames.png



imatge esperada



frames-pincushion.png



imatge esperada

Fitxers i identificadors (ús obligatori):

```
undistort.vert, undistort.frag
uniform sampler2D colorMap;
```

XRays (xrays.*) 2.5 punts

Escriu un VS i un FS per simular una mena de lupa, controlada amb el mouse, que permeti veure les capes interiors d'un dibuix d'anatomia. Farem servir quatre textures (foot0.jpg ... foot3.jpg):

```
uniform sampler2D foot0;
uniform sampler2D foot1;
uniform sampler2D foot2;
uniform sampler2D foot3;
```

El VS farà les tasques per defecte, però aplicarà un escalat S(0.5, 1, 1) al vèrtex (abans de passar-lo a clip space), de forma que l'objecte **plane.obj** passi a ser rectangular.

Pel FS, us proporcionem un **xrays.frag** que heu de completar. El que ha de fer el FS és:

1. Calcular la distància d (en píxels) del fragment a les coordenades actuals del mouse. Feu servir obligatòriament la funció mouse() que us proporcionem, que retorna les coordenades del mouse en window space.

2. Usar les coordenades de textura habituals per accedir a la textura **foot0** (pell). Sigui C el color resultant. Si **d >= R** (R és una constant que ja teniu declarada al exemple), el color del fragment serà directament C. Altrament, el color final es calcula com segueix.

3. Accedir a la textura indicada pel **uniform int layer=1** per obtenir un altre color D. Per exemple, si layer = 1, cal obtenir el color de la textura foot1. Podeu assumir que layer sempre tindrà valor 0, 1, 2 o 3.

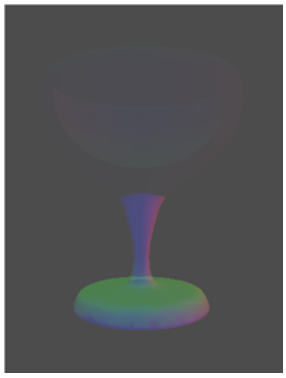
4. El color final del fragment (cas d<R) serà el resultat de fer la interpolació lineal entre D i C, on el paràmetre d'interpolació lineal serà d/R (és a dir, la distància al mouse normalitzada per R). D'aquesta manera el centre del cercle al voltant del mouse mostrarà el color D de la capa interior (indicada per layer) mentre que a mesura que ens allunyem de la posició del mouse es mostrarà gradualment el color C de pell.

Identificadors (ús obligatori):

```
xrays.vert, xrays.frag
uniform sampler2D foot0, foot1, foot2, foot3;
uniform int layer;
```

Ghostlight (ghostlight.*) 2.5 punts

Escriu un **plugin** que dibuixi el model tant transparent que sigui gairebé invisible, excepte en una regió de 100 píxels al voltant del punter del ratolí. Els resultats esperats són aquests (glass):



Prengueu com a punt de partida el plugin d'alpha blending que ve com a exemple amb l'aplicació viewer.

El FS ha de comprovar la distància entre la posició del fragment i la del ratolí (que caldrà passar via uniform) i:

- Aplicar una opacitat de 0.025 si la distància és major que 100 píxels.
- Aplicar una opacitat de 0.25 altrament.

Per a poder obtenir la posició del ratolí en cada moment podeu utilitzar la crida mouseMoveEvent de la interfície dels plugins.

Fitxers i identificadors (ús obligatori):

`ghostlight.pro, ghostlight.h, ghostlight.cpp`