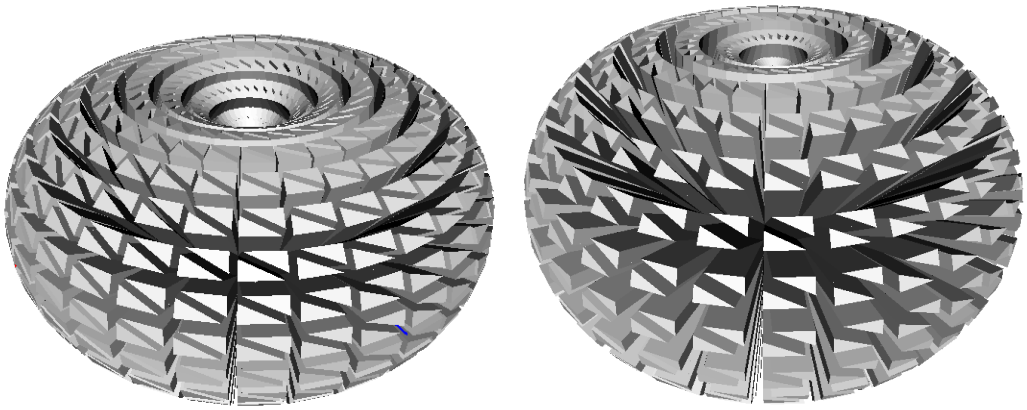
Extrude 2.5 punts

Escriu un **VS, GS** i **FS** per obtenir l'efecte de la figura (torus amb d=0.5, d=1.0):

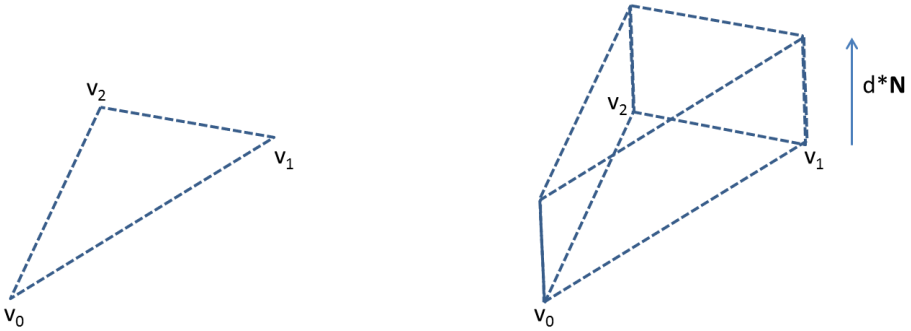


Tasques del **VS**:

- Escriure a gl_Position la posició del vèrtex en *object space*.
- Passar al GS la normal del vèrtex, també en *object space*.

Tasques del **GS**:

- El GS haurà de crear, per cada triangle, un prisma de base triangular (vegeu figura). Siguin V_0, V_1 i V_2 els vèrtexs del triangle original, i N_0, N_1, N_2 les seves normals. Sigui N el promig normalitzat de N_0, N_1, N_2. Sigui *d* un uniform float definit per l'usuari. Els vèrtexs del prisma seran V_0, V_1, V_2, i (V_0+d***N**), (V_1+d***N**) i (V_2+d***N**).



- Per cadascun dels vèrtexs del prisma que creï el GS, caldrà que escrigui la normal de la seva cara dins el prisma (en *object space*), així com la seva gl_Position (òbviament en *clip space*).

Tasques del **FS**:

- Simplement calcular el color del fragment com el gris que resulta de fer servir la component Z de la normal en *eye space* (similar a l'exercici simple lighting).

Cal lliurar:

```
extrude.vert   extrude.geom    extrude.frag
```

Identificadors:

```
uniform float d;
```

Grass 2.5 punts

Aquest exercici és una continuació de l'anterior. No l'intenteu fer abans de completar Extrude.

Escriu un **VS**, **GS** i **FS** per obtenir un efecte similar al de la figura (plane amb d=0.1):

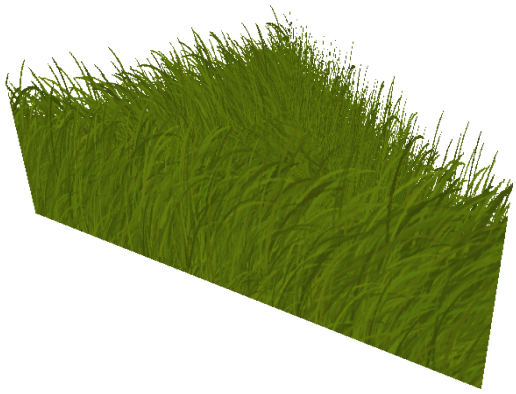
Tasques del **VS**: idèntiques a l'exercici extrude.

Tasques bàsiques del **GS**: també haurà de crear un prisma de base triangular, però sense incloure la cara superior. Al igual que abans, *d* és un uniform float definit per l'usuari.

Tasques del **FS**:

- Determinarà si es tracta del fragment d'una cara vertical (N_z==0, on N_z és la normal en *object space*) o horitzontal (altrament).
- Rebrà també la posició **gPos** del fragment en *object space* (caldrà que el GS li passi).
- Si la cara és vertical, usarà la textura grass_side. Com a coordenades de textura, feu servir `vec2(4*(gPos.x - gPos.y), 1.0 - gPos.z/d)`. Haureu de descartar el fragment si el texel té alpha < 0.1.
- Si la cara és horitzontal, usarà la textura grass_top. Com a coord. de textura, useu `4*gPos.xy`.

Aquí teniu un exemple processant només el primer triangle de plane (d=0.1):



Cal lliurar:

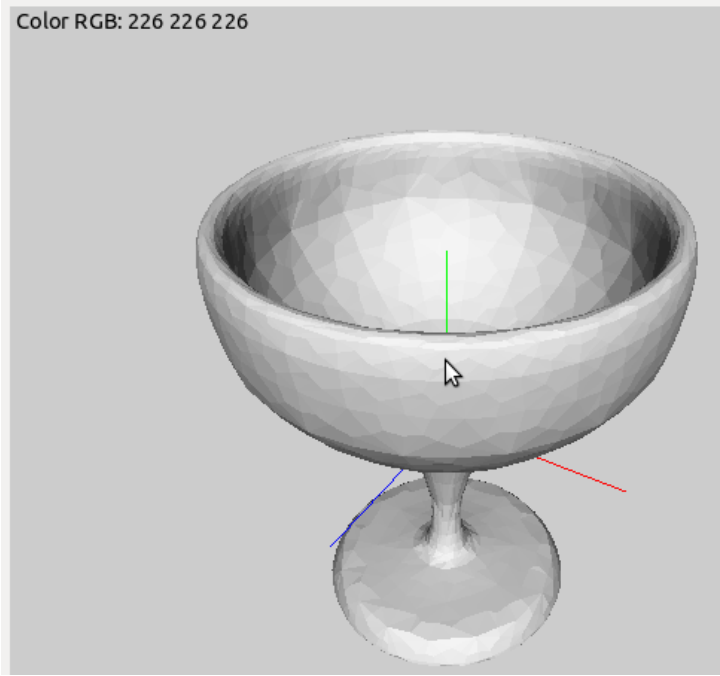
`grass.vert` `grass.geom` `grass.frag`

Identificadors:

```
uniform float d;
uniform sampler2D grass_top, grass_side;
```

Show-color 2.5 punts

Completeu el **effect plugin** que us proporcionem per obtenir un efecte com el de la figura (glass.obj):



El **plugin** haurà de dur a terme aquestes tasques:

1. Configurar un timer per tal que la finestra es repinti contínuament, encara que no hi hagi altra mena d'esdeveniments (similar al que fa el plugin d'auto-update).

2. Mostrar a la finestra OpenGL el color RGB del píxel on està el cursor (valors entre 0 i 255).

   Les coordenades del cursor relatives a la finestra OpenGL les podeu obtenir amb

   ```
   int mousex = pglwidget->mapFromGlobal(QCursor::pos()).x();
   int mousey = pglwidget->mapFromGlobal(QCursor::pos()).y();
   ```

   Per llegir el color del buffer de color, haureu d'usar la crida **glReadPixels.**

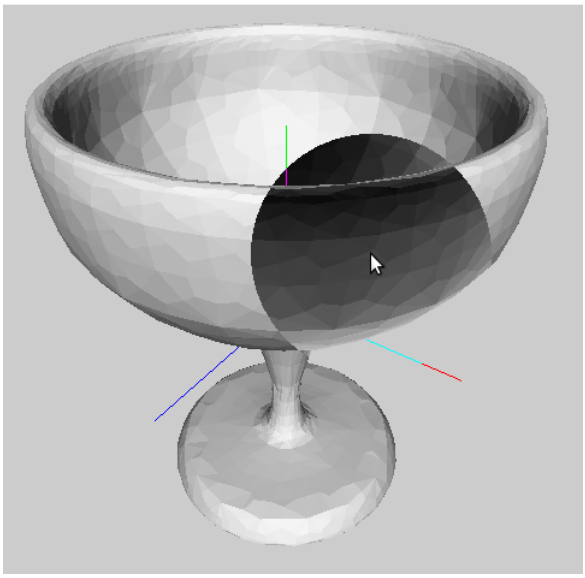
   Per escriure text a la finestra crideu el mètode **renderText.**

Cal lliurar:

`show_color.h`
`show_color.cpp`

Inverter 2.5 punts

Completeu el **effect plugin** que us proporcionem per obtenir un efecte com el de la figura (glass.obj):



El **plugin** haurà de dur a terme aquestes tasques:

1. Configurar un timer per tal que la finestra es repinti contínuament, encara que no hi hagi altra mena d'esdeveniments (similar al que fa el plugin d'auto-update).

2. Definir un **fragment shader** amb la funcionalitat que es descriu a sota.

3. Obtenir les coordenades (x,y) del cursor i passar-li aquesta informació actualitzada al fragment shader (**enters**). Les coordenades del cursor relatives a la finestra OpenGL les pots obtenir amb

```
int mousex = pglwidget->mapFromGlobal(QCursor::pos()).x();
int mousey = pglwidget->mapFromGlobal(QCursor::pos()).y();
```

El **fragment shader** (el codi del qual estarà com QString al mateix plugin) haurà de dur a terme aquestes tasques:

1. Calcular la distància d, en píxels, entre el fragment i la posició del cursor.

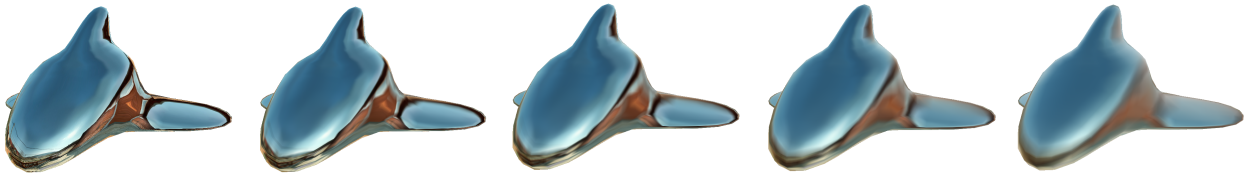
2. Si d > 100, el color del fragment serà directament gl_Color; altrament el color del fragment es calcularà invertint gl_Color, és a dir, un color (r,g,b,a) passarà a ser (1-r, 1-g, 1-b, 1-a).

Cal lliurar:

`inverter.h`
`inverter.cpp`

Glossy 2.5 punts

Al fitxer adjunt us proporcionem una implementació (VS+FS) de *sphere mapping*. Volem aconseguir l'efecte de la figura, fent servir un paràmetre enter *r* que determini la quantitat de mostres del sphere map que es tenen en compte per calcular el color de cada fragment (boid.obj, *r*=0, 5, 10, 20, 40):



El **FS** que us proporcionem té una funció `sampleTexture` que pren una mostra de la textura (feu servir **glossy.png**) al punt de coordenades de textura (s,t):

```
vec4 sampleTexture(sampler2D sampler, vec2 st, int r)
{
   return texture2D(sampler, st);
}
```

El que us demanem és que completeu aquesta funció per tal que retorni el promig dels colors d'una regió quadrada de la textura centrada en el punt *st*. En concret, si C(s,t) el que retorna la funció texture2D pel punt (s,t), i (W,H)=(512,512) és la mida de la imatge, la funció `sampleTexture` ha de retornar

$$\frac{1}{(2r+1)^2}\sum_{i=-r}^{r}\sum_{j=-r}^{r} C\left(s+\frac{i}{W}, t+\frac{j}{H}\right)$$

Observeu que el nombre de crides a texture2D serà $(2r+1)^2$, i per tant és d'esperar que el frame rate baixi considerablement per valors grans de *r*. **Hi ha un cert risc de penjar el PC amb valors grans de r.** Hem afegit un min(40,r) al codi per tal de limitar aquest risc, però mireu de fer les proves amb valors petits.

Aquí teniu un altre exemple del resultat esperat (glass.obj, r=40):

Cal lliurar:

`glossy.vert  glossy.frag`

Identificadors:

```
uniform int r;
uniform sampler2D glossy;
```