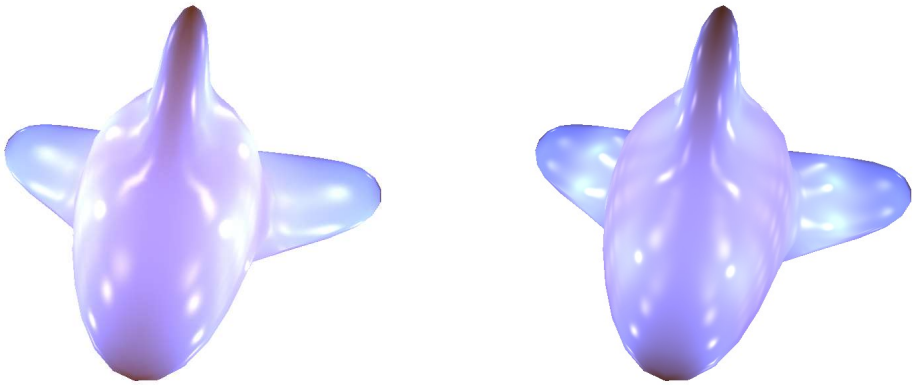
ManyLights (manylights.*)

useu `~/assig/grau-g/Viewer/GlarenaSL` per resoldre aquest exercici.

Escriu VS+FS per il·luminar el model amb un nombre arbitrari de fonts de llum distribuïdes de manera uniforme dins la capsa englobant del model:



Concretament, volem aplicar **el model d'il·luminació de Phong per fragment** (com vau fer a l'exercici lighting 4):

$$K_aI_a + K_dI_d(N\cdot L) + K_sI_s(R\cdot V)^s$$

on

K_a, K_d, K_s = reflectivitat ambient, difosa i especular del material
s = shininess del material
I_a, I_d, I_s = propietats ambient, difosa i especular de la llum
N = vector normal unitari
L = vector unitari cap a la font de llum
V = vector unitari del vèrtex cap a la càmera
R = reflexió del vector L respecte N. Es pot calcular com $R=2(N\cdot L)N-L$

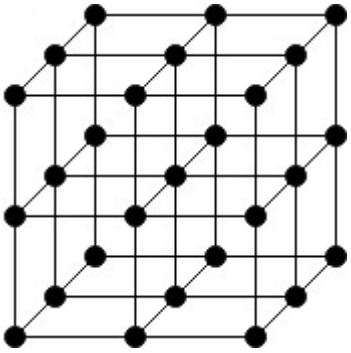
Pel que fa al material, feu servir els uniforms usuals (matAmbient, matDiffuse, matSpecular, matShininess). Pel que fa al color de la llum, useu també els uniforms habituals (lightAmbient, lightDiffuse, lightSpecular).

En comptes d'una única font de llum a lightPosition, n'hi haurà (NUM+1)* (NUM+1)* (NUM+1) fonts de llum, on

uniform int NUM = 5; // no proveu amb valors més grans que 5

Les fonts de llum (en object space) les heu de situar en una graella regular, cobrint la capsa englobant del model (donada per boundingBoxMin, boundingBoxMax).

Per exemple, si NUM = 2, les fonts de llum estaran en les posicions de la figura (una d'elles coincidirà amb boundingBoxMin i una altra amb boundingBoxMax):

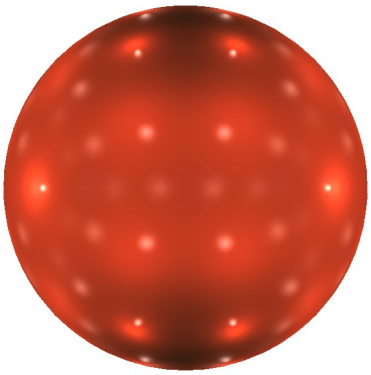


Òbviament, pels càlculs d'il·luminació, les llums hauran d'estar en l'espai adient.

El color final del fragment serà la suma de les contribucions de totes les fonts de llum. Com que n'hi ha moltes, per tal de no cremar el model, a la contribució de cada llum caldrà multiplicar-li un decreixement exponencial en funció de la distància d (en eye space) entre el punt i la llum,

exp(-decay*d);

amb **uniform float decay = 6.0,** de forma que cada llum només il·luminarà la geometria més propera.

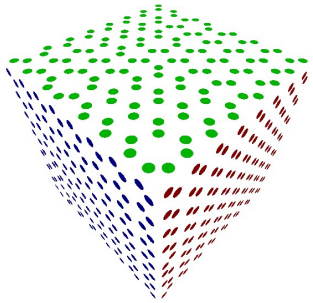


Identificadors obligatoris:
`manylights.vert, manylights.frag`
Els uniforms de l'enunciat.

Dots (dots.*)

useu `~/assig/grau-g/Viewer/GlarenaSL` per resoldre aquest exercici.

Escriu VS+GS+FS per tal de dibuixar només cercles centrats a cada triangle del model:



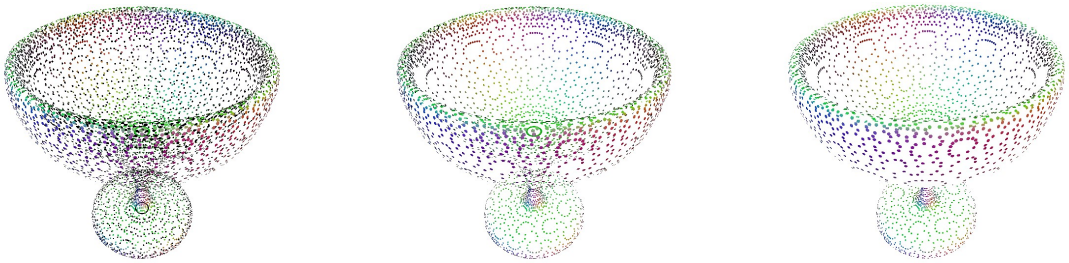
El VS farà les tasques per defecte, però escriurà gl_Position en *object space* i enviarà la normal en *eye space*.

El GS escriurà, per cada vèrtex, quatre valors: gfrontColor (amb el color que rep del VS), gl_Position (en l'espai adient), les coordenades P del vèrtex (en *object space*) i el centre C del triangle (també en *object space*). Feu servir un **uniform bool culling = true** per eliminar les cares backface: si culling és cert i tots els vèrtexs del triangle tenen la normal amb Z negativa, el GS no emetrà cap primitiva.

El FS serà l'encarregat de dibuixar els cercles. Per esbrinar si el fragment és dins el cercle, farà servir la distància (en object space) entre el punt P i el centre C. Si aquesta distància és més gran que **uniform float size = 0.02**, el fragment és fora del cercle. El tractament d'aquests fragments dependrà de **uniform bool opaque = true.** Si opaque és cert, els fragments fora del cercle seran de color blanc; altrament, els fragments fora del cercle es descartaran.

Els fragments dins els cercles tindran el color que reben del GS.

Aquí teniu el resultat amb opaque i culling false (esquerra), culling true (centre) i tots dos true (dreta).



Puntuació orientativa: 7 punts per dibuixar correctament els cercles; 2 punts pel culling; 1 punt pel mode opaque.

Identificadors obligatoris:
`dots.vert, dots.geom, dots.frag`
Els uniforms de l'enunciat.

Hole (hole.*)

Useu GLarenaPL de la vostra instal·lació local del visualitzador

Escriu un **VS+FS** que texturi l'objecte **plane.obj** amb la textura **hole.jpg**:

El VS farà les tasques imprescindibles.

El FS aplicarà la textura usant les coordenades de textura que rebrà del VS. Recordeu que l'objecte plane.obj té coordenades de textura en l'interval [0, 1], i que el centre és al (0.5, 0.5).

Volem, però, que hi hagi una certa repetició de la part central de la textura (la part dins un radi R=0.2, en espai de textura, respecte al centre).

Per a controlar el nombre de repeticions, usarem un **uniform int N = 3**.

Si N=0, dibuixarà la textura tal qual (com la figura de dalt).

Si N=1, dibuixarà la textura, excepte pels fragments dins el cercle central, pels quals s'escalaran les coordenades de textura (respecte al centre) de manera adient (en funció del radi R) per aconseguir repetir la textura (penseu quin factor d'escala cal aplicar en aquest cas).

Si N>1, cal aplicar repetidament la transformació anterior a les coordenades de textura prou properes al centre, per a aconseguir N còpies. Aquí teniu el resultat amb N=1, N=2 i N=3.

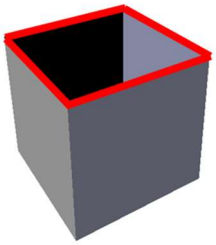
Identificadors obligatoris:
`hole.vert, hole.frag`
Els uniforms de l'enunciat.

Boundary (boundary.*)

Useu GLarenaPL de la vostra instal·lació local del visualitzador

Escriu un **plugin** que escrigui el nombre total d'arestes dels objectes de l'escena, i també el nombre total d'arestes que formen part de la frontera de la superfície, és a dir, arestes que només pertanyen a una cara.

Per exemple, si a un cub amb 6 cares (12 arestes) li traiem una de les seves cares, la seva superfície continua tenint 12 arestes, 4 de les quals seran de la frontera:



Els mètodes **onPluginLoad** i **onObjectAdd** cridaran un mètode privat, que s'encarregà de calcular i escriure pel canal de sortida estàndard (amb cout) les següents magnituds, per a cadascun dels objectes de l'escena:

- Nombre total d'arestes (E) de l'objecte.
- Nombre d'arestes que només pertanyen a una cara.

Per al càlcul del nombre d'arestes, és important que no compteu arestes duplicades. Per exemple, si una cara té una aresta amb vèrtexs (3, 6), i una altra té una aresta amb vèrtexs (3, 6) o (6,3), en realitat són la mateixa aresta (compartida per més d'una cara i, per tant, que no pertany a la frontera).

Aquí teniu un exemple del resultat esperat (**observeu el format de la sortida**), per una escena amb monkey.obj, default.obj i man.obj (els comentaris no formen part de la sortida):

```
// monkey
E=1657
Border=410

// default
E=9630
Border=0

// man
E=44597
Border=2806
```

Identificadors obligatoris:
boundary.cpp, boundary.h, boundary.pro
Entregueu sols aquests tres arxius dins una carpeta que es digui boundary, en un .zip, .tar o .rar