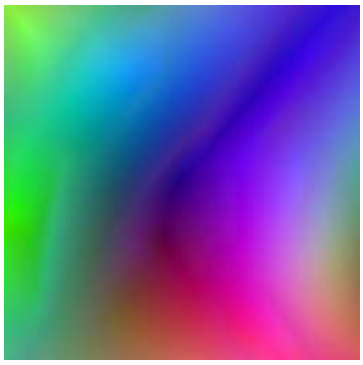
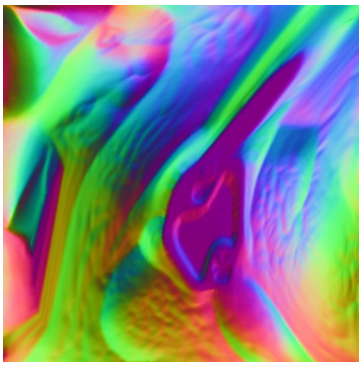
Geometry Image (geometry-image.*)

Useu: `~/assig/grau-g/Viewer/GlarenaSL` El test és només orientatiu

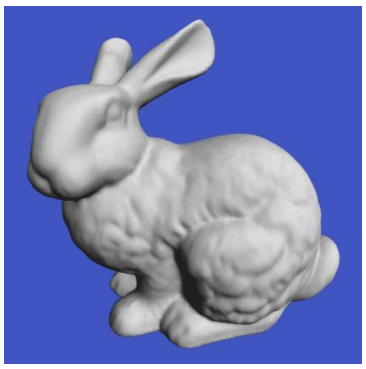
Si tenim una superfície parametritzada, es pot construir una textura on cada texel amb coordenades (s,t) representi la posició (assumirem en *object space*) del punt de la superfície que té aquelles coordenades de textura. La imatge resultant és una *geometry image*, la qual sovint s'utilitza amb una altre imatge que codifica la normal (també en *object space*). Aquí teniu un exemple per a una determinada parametrització del Stanford bunny:



Geometry image



Normal image



Bunny

Escriu un VS+FS que, a partir d'un pla força subdividit (objecte **plane256.obj**), dibuixi la superfície representada per una *geometry image* (**bunny-geo.png**) amb les normals d'una *normal image* (**bunny-norm.png**).

Concretament, el **VS** farà les següents tasques. Primer, calcularà unes coordenades de textura (s,t) a partir de les coordenades (x,y) del vèrtex. A l'objecte **plane256.obj**, (x,y) varien en [-1,1]; escaleu i traslladeu aquestes coordenades per generar coordenades de textura en [0.004, 0.996].

Useu aquestes coordenades (s,t) per prendre una mostra de les dues textures:

```
uniform sampler2D positionMap;
uniform sampler2D normalMap1;   // observa el dígit 1 al final
```

Els valors (r,g,b) de la primera textura els interpretarem directament com la nova posició P del vèrtex, en object space. Per tant reconstruirem un objecte amb coordenades en [0,1].

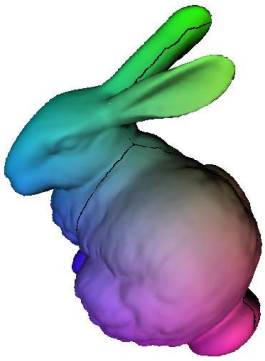
Els valors (r,g,b) de la segona textura codifiquen la normal N del vèrtex, també en object space. Però atès que la textura us retorna valors en [0,1] caldrà fer la transformació adient per tenir les components de la normal en [-1, 1].

El color del vèrtex dependrà d'un **`uniform int mode = 0`**. Si **mode és 0 [4 punts]**, el color serà directament la posició P del vèrtex en object space:

Ignoreu la discontinuïtat al voltant del cap i les orelles; la superfície no tancarà perfectament.

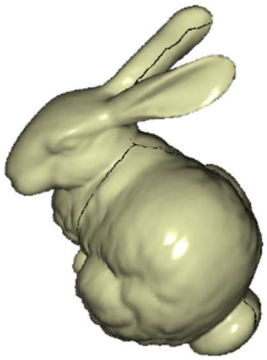
Per la resta de modes, usarem la normal N que heu obtingut de la segona textura, però en eye space i normalitzada.

Si **mode és 1 [2 punts]**, el color serà el mateix que en mode=0, però multiplicat per la N.z.

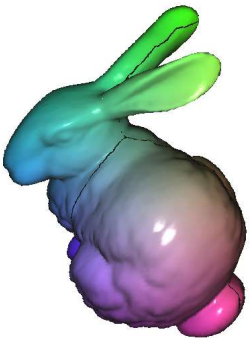


Si **mode és 2 [3 punts]**, el color es calcularà (per vèrtex) amb el model d'il·luminació de Phong,

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



Finalment, si **mode és 3 [1 punt]**, també calcularеu el color amb el model de Phong, però aquest cop usareu la posició P en substitució de matDiffuse.



El **FS** farà les tasques per defecte.

Identificadors obligatoris:

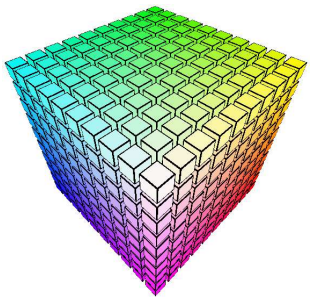
`geometry-image.vert, geometry-image.frag` (Has escrit `geometry-image` correctament? En minúscules?)
Tots els uniforms de l'enunciat, i els requerits per aplicar Phong (matAmbient, matDiffuse...)

Gràfics
Curs 2023/24 Q1

RGB color space (rgb-color-space.*)

Useu: `~/assig/grau-g/Viewer/GLarenaSL`

Escriu VS+GS+FS per representar, amb cubs de colors, l'espai de color RGB:



Considerarem que els valors (r,g,b) d'aquest espai tenen components en [0,1]. Treballarem amb l'objecte **filled-cube.obj** el qual té una col·lecció de triangles distribuïts per l'interior d'un cub amb extrems als punts (-1, -1, -1) i (1,1,1).

El **VS** farà les tasques imprescindibles, escrivint gl_Position en *object space*.

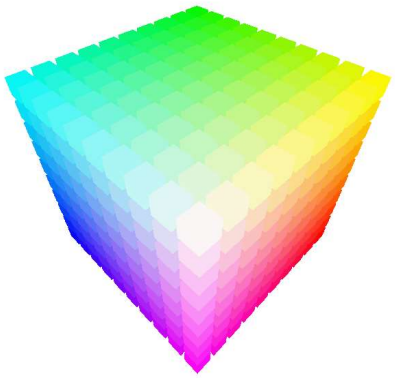
El **GS** crearà un cub per cada triangle, centrat al centre del triangle, i amb una mida de 0.16. El color de les cares del cub dependrà del `uniform int mode = 3.`

Si **mode = 1 [5 punts]**, el color de les cares del cub l'haureu de calcular a partir de les coordenades del centre C del triangle; aquest centre tindrà coordenades (x,y,z) en [-1, 1], que haureu de transformar a coordenades (r,g,b) en [0,1].

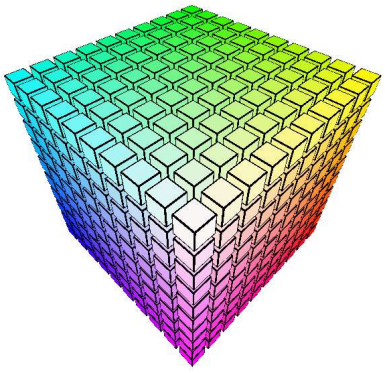
Si **mode = 2 [3 punts]**, el FS assignarà al fragment el color que li arribi del GS, tret d'un petit marge al voltant de cada cara, que serà de color negre. Per aquest marge, podeu fer que el GS generi coordenades de textura (s,t) per cada vèrtex, entre 0 i 1, i que el FS utilitzi el color de la cara si s, t estan dins [0.05, 0.95]; i que altrament assigni color negre.

Si **mode = 3 [2 punts]**, a diferència del mode 2, el GS només generarà un cub si el centre C té alguna coordenada més petita que `uniform float cut = -0.25,` de forma que seran visibles cubs de l'interior de l'espai RGB.

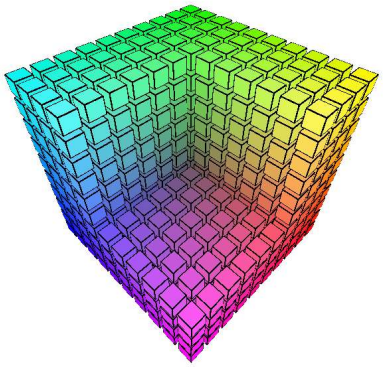
El FS farà les tasques imprescindibles, assignant el color que li arribi del GS, excepte pel petit marge en els modes 2 i 3.



Mode = 1



Mode = 2



Mode = 3

Identificadors obligatoris:

`rgb-color-space.vert, rgb-color-space.geom, rgb-color-space.frag` **(en minúscules!)**

La resta d'uniforms estàndard necessaris segons l'enunciat.

Depthnormal (depthnormal.*)

Useu `GLarenaPL` de la vostra instal·lació local del visualitzador

Escriu un **plugin** que, utilitzant dues passades de rendering, permeti visualitzar, de manera independent, el depth map i el normal map de l'escena, des de la càmera actual.

El mètode **onPluginLoad** carregarà dos parells de shaders, que haureu d'escriure vosaltres:

- depth.vert, depth.frag
- normal.vert, normal.frag

El primer parell tenen el comportament per defecte, però assignant, com a color del fragment, el gris que té per components la Z del fragment, en window space.

El segon parell és similar, però el VS li passarà al FS la normal en object space. El FS convertirà la normal en un color, transformant les seves components (en object space) de valors en [-1, 1] a valors en [0,1]. Aquesta nova normal serà el color del fragment.

El mètode paintGL dibuixarà l'escena en dos passos. Abans però, caldrà redefinir la relació d'aspecte de la càmera, amb camera()->setAspectRatio(*float* ar), tenint en compte que la relació d'aspecte per la finestra OpenGL és width() / height(), però quan pintem només en la meitat esquerra/dreta del viewport, serà la meitat d'aquest valor.

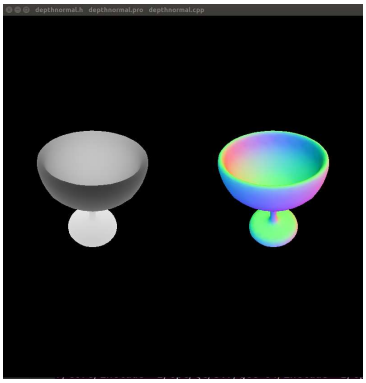
Al primer pas, caldrà activar la primera parella de shaders (depth.*), usar glViewport amb els paràmetres adients per la meitat esquerra, enviar als shaders les matrius necessàries, i finalment dibuixar l'escena.

Al segon pas, caldrà activar la segona parella de shaders (normal.*), usar glViewport amb els paràmetres adients per la meitat dreta, enviar als shaders les matrius necessàries, i finalment dibuixar l'escena.

Recordeu que els mètodes width() i height() de QOpenGLWidget permeten conèixer la mida de la finestra OpenGL. La signatura de la funció glViewport és

```
void glViewport(GLint xorigin, GLint yorigin,  GLsizei width, GLsizei height);
```

A continuació teniu un exemple del resultat esperat:



Identificadors obligatoris:

```
depthnormal.cpp, depthnormal.h, depthnormal.pro
normal.vert, normal.frag
depth.vert, depth.frag
```