window.vert, window.frag 2.5 punts

Escriu un **VS** i un **FS** per tal de simular una finestra a través de la qual es pot veure l'interior d'una habitació. Al ZIP de l'enunciat trobareu les textures **interior.png** i **window.png**:


El VS, a banda de les tasques habituals, li passarà al FS **la normal N en eye space.**

El FS accedirà a la textura **window.png** (amb les coordenades de textura habituals) per obtenir un color que li direm **C**.

Si la component alfa de C és 1.0, el color del fragment serà simplement C.

Si la component alfa de C és inferior a 1.0, el color del fragment es calcularà com a interpolació lineal entre dos colors: d'una banda C, i d'altra banda, el color de la textura **interior.png** al punt de coordenades **vtexCoord + vec2(0.5*N.x, -0.5*N.y)**, on com hem dit N és la normal en eye space. Observeu que estem usant un offset que depèn de les components de la normal. El paràmetre de la interpolació lineal serà el **valor absolut de N.z**.

Aquí teniu el resultat esperat (plane.obj), des de diferents punts de vista:

Observeu que, degut a l'offset en les coordenades (s,t), la part visible de l'interior depèn de l'orientació del model.

Fitxers i identificadors (ús obligatori):

```
window.vert, window.frag
uniform sampler2D interior0;  // observeu el digit 0 al final
uniform sampler2D window1;    // observeu el digit 1 al final
```

gradient3.vert, gradient3.frag 2.5 punts

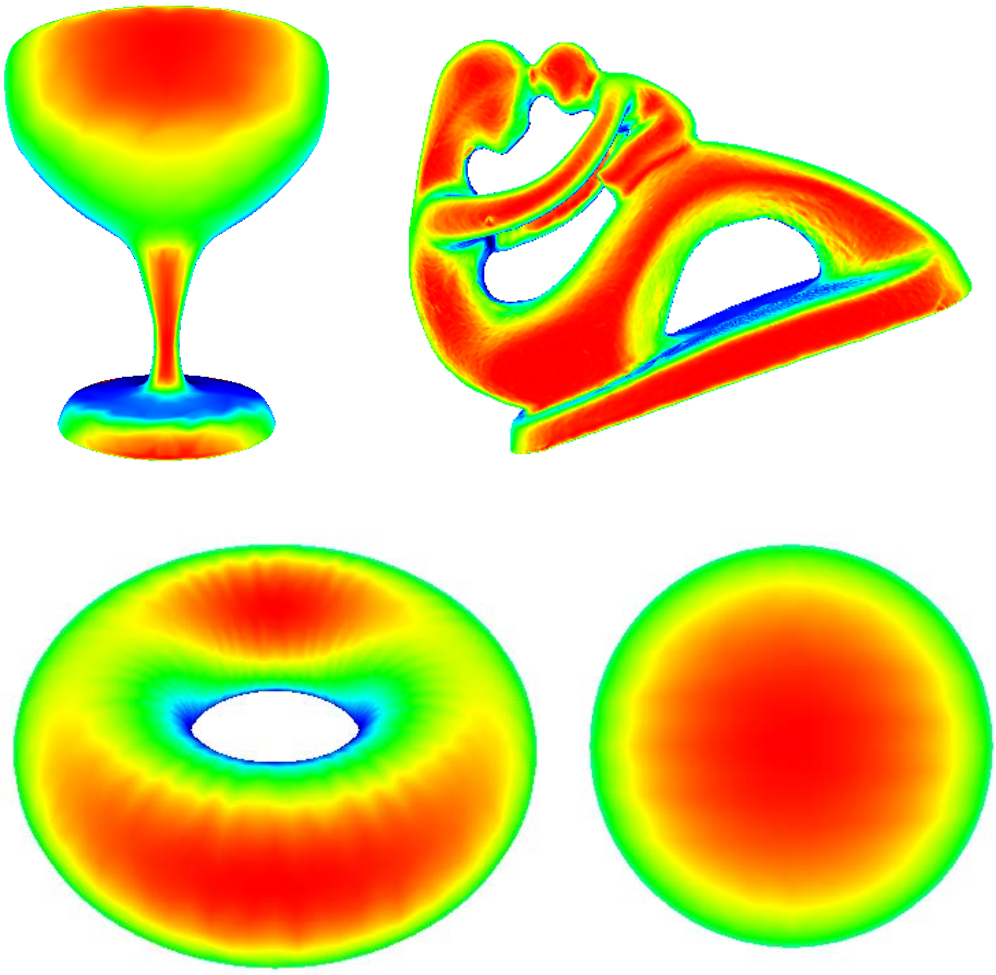
Escriu un VS i un FS que colorin el model en funció de la component Z de la normal unitària en eye space.

El VS haurà de passar la normal al FS, en eye space.

El FS calcularà el color en funció de la component Z de la normal unitària. El gradient de color estarà format per la interpolació d'aquests cinc colors: **red, yellow, green, cian, blue**. L'assignació s'haurà de fer de forma que els vèrtexs amb **N.z = 1 es pintin de vermell**, i els vèrtexs amb **N.z<=0 es pintin de blau**.

Per a la interpolació lineal entre colors consecutius del gradient, feu servir la funció **mix**. Una altra funció que us pot ser útil és **fract**, la qual retorna la part fraccionaria de l'argument.

Aquí teniu el resultat esperat amb diferents models:



Fitxers i identificadors (ús obligatori):
`gradient3.vert, gradient3.frag`

cartoon.vert, cartoon.frag 2.5 punts

Escriu un VS i un FS que colorin el model de forma anàloga a alguns còmics.

El VS farà les tasques habituals, passant al FS les dades que caldran per calcular la il·luminació per fragment.

El FS primer calcularà un valor real **f** afegint els termes difós i especular (Phong), però sense tenir en compte ni el color del material ni el color de la font de llum:

$$f = (N \cdot L) + (R \cdot V)^s$$

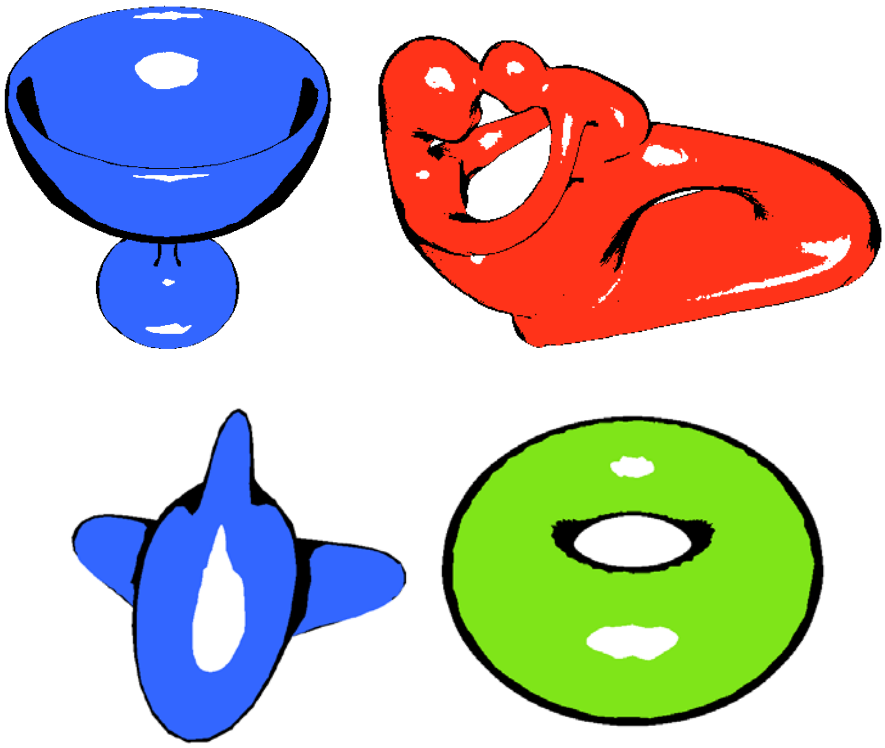
N = normal unitària
L = vector unitari cap a la font de llum
R = reflexió del vector L respecte N. Es pot calcular com R=2(N·L)N-L
V = vector unitari del vèrtex cap a la càmera
s = shininess del material (`matShininess`)

Observació: recordeu evitar "restar" quan els productes escalars N·L o R·V són negatius.

Un cop heu calculat f, el color final del fragment serà:

- Negre, si f < 0.4;
- El color difós del material (matDiffuse), si f>=0.4 i f<0.98;
- El color especular del material (matSpecular) si f>=0.98

Aquí teniu el resultat esperat amb diferents models:



Fitxers i identificadors (ús obligatori):
`cartoon.vert, cartoon.frag`

sun.vert, sun.frag — 2.5 punts

Escriu un VS i un FS que colorin els fragments com si haguéssim aplicat una textura amb un Sol groc sobre un fons vermell.

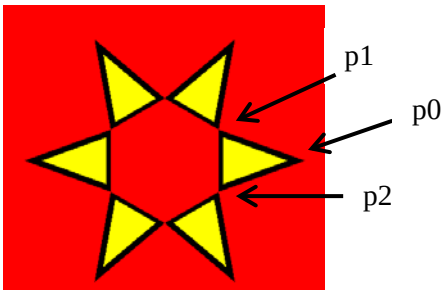
El VS farà les tasques habituals, passant al FS les coordenades de textura **multiplicades per 10**.

El FS farà servir **la part fraccionària** de les coordenades de textura, que anomenarem (s,t).

Si la distància del punt (s,t) al punt (0.5, 0.5) és inferior a 0.2, el color final del fragment serà groc (cada Sol té un cercle de color groc de radi 0.2 en espai de textura).

Altrament, el color es decidirà com groc o vermell comprovant si el punt (s,t) és a dins d'algun dels n triangles (**uniform int n=12**) que defineixen els "rajos" del Sol, definits pels tres vèrtexs p0, p1, p2:

- p0 = (0.5, 0.5) + 0.45(cos α, sin α)
- p1 = (0.5, 0.5) + 0.2 (cos(α+β), sin(α+β))
- p2 = (0.5, 0.5) + 0.2 (cos(α-β), sin(α-β))



on

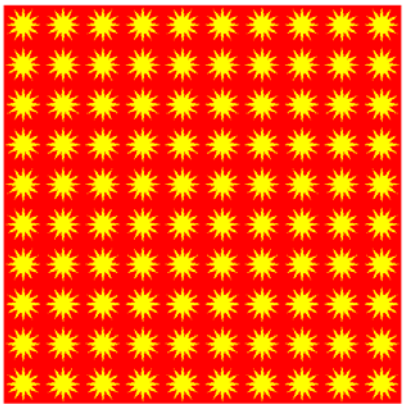
- α=2πi/n (amb i variant entre 0 i n-1)
- β= π/n

La figura mostra aquests tres punts pel triangle amb i=0 quan n=6.

Si (s,t) és a dins d'algun del n triangles, el color final serà groc. Altrament serà vermell.

Podeu usar aquesta funció per determinar si un punt p és dins del triangle definit per p0, p1, p2:

```
bool pointInsideTriangle(vec2 p, vec2 p0, vec2 p1, vec2 p2)
{
 float a = (-p1.y*p2.x + p0.y*(-p1.x + p2.x) + p0.x*(p1.y - p2.y) + p1.x*p2.y);
 float s = 1./a*(p0.y*p2.x - p0.x*p2.y + (p2.y - p0.y)*p.x + (p0.x - p2.x)*p.y);
 float t = 1./a*(p0.x*p1.y - p0.y*p1.x + (p0.y - p1.y)*p.x + (p1.x - p0.x)*p.y);
 return (s>0) && (t>0) && (1-s-t>0);
}
```



Fitxers i identificadors (ús obligatori):

```
sun.vert, sun.frag
uniform int n = 12;
```