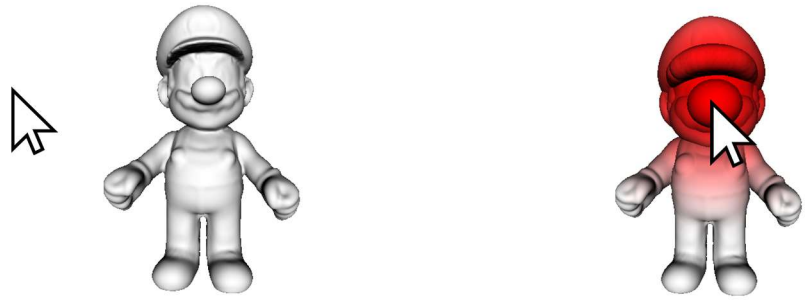
Mouse Bump (mousebump.*)

Escriu VS+FS per a inflar el model carregat allà on està col·locat el ratolí:



El VS farà les següents tasques. Sigui P el vèrtex original en *eye space*. El VS calcularà un nou punt P' com el resultat de desplaçar P en la direcció de la normal (en *eye space*) un 3% de la longitud de la diagonal de la caixa contenidora de l'objecte. La posició final del vèrtex, en *eye space*, serà el resultat de calcular la interpolació lineal entre P i P' en funció d'un paràmetre d'interpolació *t*.

El paràmetre *t* ha de variar suaument (smoothstep) entre 0 i 1, tenint en compte un uniform anomenat *radius* i la distància *d* que hi ha entre la posició del ratolí i la de P en *window space*, de forma que

- *t* = 0 quan *d* >= 80% de *radius*
- *t* = 1 quan *d* <= 5% de *radius*
- i hi hagi una interpolació suau per a la resta de valors de *d*.

El color del vèrtex serà una interpolació lineal entre blanc i vermell en funció del valor de *t*, amb una il·luminació bàsica en funció de la component z de la normal en *eye space*.

Per a calcular la posició del ratolí en *window space* cal que feu servir la següent funció:

```
vec2 getMousePositionWindowSpace() {
    if(test == 0) return mousePosition;
    if(test == 1) return vec2(400,520);
    if(test == 2) return vec2(600,225);
    if(test == 3) return vec2(200,375);
    return vec2(400,300);
}
```

que teniu disponible al fitxer mousePosition.vert. A més a més, per a calcular la posició de P en *window space* us pot ser útil el **uniform vec2 viewport** que conté la mida del viewport, en pixels (*width*, *height*). El FS farà les tasques per defecte.

Identificadors obligatoris:

mousebump.vert, mousebump.frag (en minúscules!)

```
uniform int test = 0;
uniform vec3 boundingBoxMin;
uniform vec3 boundingBoxMax;
uniform vec2 mousePosition;
uniform float radius = 300;
uniform vec2 viewport;
```

La resta d'uniforms necessaris segons l'enunciat.

Damage (damage.*)

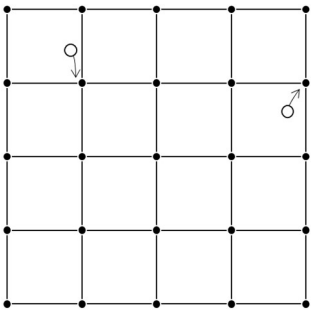
Una cerca a internet per una funció GLSL que torni un real aleatori retorna funcions similars a aquesta:

```
float rand(vec2 co){
   return cos(sin(dot(co, vec2(12.9898, 78.233))) * 437.585453) / 2 + 0.5;
}
```

que farem servir per acolorir aleatòriament un model. El paràmetre *co* juga el paper de llavor, i fent servir porcions de les coordenades de punts en espai objecte, podrem variar el color del model com a la figura:

Per aconseguir-ho, subdividirem la capsa englobant del model en **`uniform int num=4`** parts iguals en cada direcció coordenada. Diguem V_i als vèrtexs d'aquestes subcaixes. Programa un VS que per cada vèrtex trobi el V_i més proper, diguem que sigui **`vrnd`**, i assigni al vèrtex el color amb component vermella **`rand(vrnd.xy)`**, component verda **`rand(vrnd.yz)`** i component blava **`rand(vrnd.zx)`**, modulat per la component z de la normal en *eye space* com fem habitualment.

Aquí teniu un example (en 2D) d'una capsa subdividida amb num=4, juntament amb el vèrtex V_i més proper a dos punts d'exemple:



No has de modificar el FS.

Nota: es valorarà el càlcul de **`vrnd`** de forma concisa i aprofitant les expressions vectorials del GLSL.

Per a comprovar visualment si estàs calculant vrnd correctament, pots provar (temporalment) d'escriure gl_Position substituint el vèrtex per vrnd.

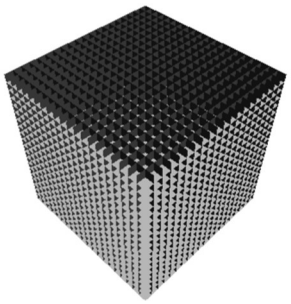
Identificadors obligatoris:

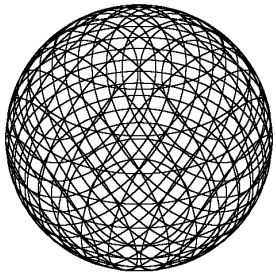
`damage.vert, damage,frag (en minúscules!)`
`uniform vec3 boundingBoxMin;`
`uniform vec3 boundingBoxMax;`
Els uniforms de l'enunciat.

Implicit (implicit.*)

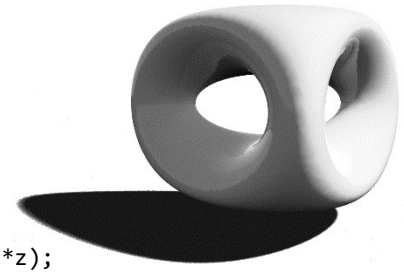
Escriu un VS+FS per obtenir una visualització "en filferros" d'una superfície implícita. Una superfície implícita està definida per una igualtat de la forma f(x,y,z) = 0, on f és qualsevol funció real definida a l'espai. Per exemple, si definim $f(x,y,z) = x^2 + y^2 + z^2 – r^2$, la superfície implícita és una esfera de radi r.

Una forma de visualitzar una superfície implícita és dibuixar una col·lecció densa de plans, i usar un FS que només dibuixi els fragments pels quals f(P) ≈ 0, descartant la resta (P és la posició del fragment en *object space)*. En aquest exercici usarem el model **ortho_planes.obj**, el qual conté tres col·leccions de plans perpendiculars als eixos; considerant els fragments pels quals f(P) ≈ 0, obtenim el resultat de la dreta, que correspon a la intersecció dels plans amb l'esfera:





En el nostre cas, us proporcionem una funció polinomial f que representa la superfície de la dreta:

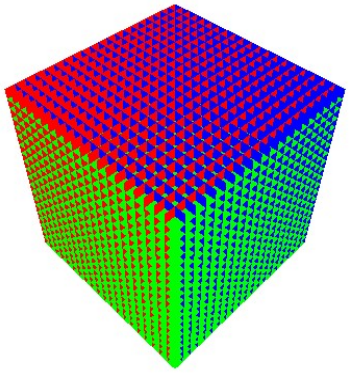


```
// disponible al fitxer f.txt
float f(vec3 Q)
{
	float x = Q.x*1.65;
	float y = Q.y*1.9;
	float z = Q.z*1.01;
	return 2*y*(y*y-3*x*x)*(1-z*z)+pow(x*x+y*y,2)-(9*z*z-1)*(1-z*z);
}
```

El VS farà les tasques imprescindibles, enviant al FS les coordenades del vèrtex en *object space,* i el color original del vèrtex (sense cap il·luminació).

Respecte al FS, el seu comportament dependrà d'un **uniform int mode = 0.**

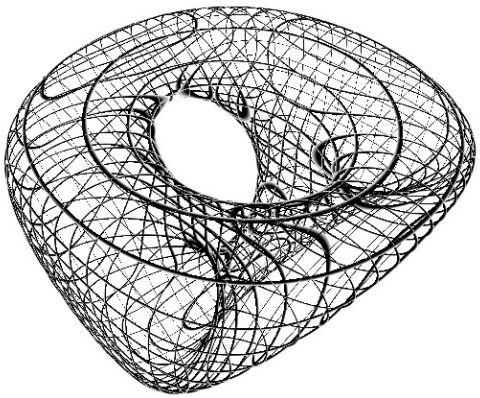
Si mode és 0 (1 punt), el FS simplement escriurà com a color del fragment el color original que rep del VS, sense cap il·luminació. Això ens permet visualitzar el domini en el qual avaluarem la funció f:



Si mode és 1 (5 punts), visualitzarem només els fragments pels quals f(P) **≈** 0, on P és la posició del fragment en *object space*. Sigui v el valor de f(P). Usarem un llindar definit per

uniform float eps = 0.05;

Si **|v| < eps**, considerem que el fragment és prou proper a la superfície implícita f(P)=0, i escriurem com a color del fragment el gris que té per components el valor **20*v**. Altrament, descartarem el fragment:

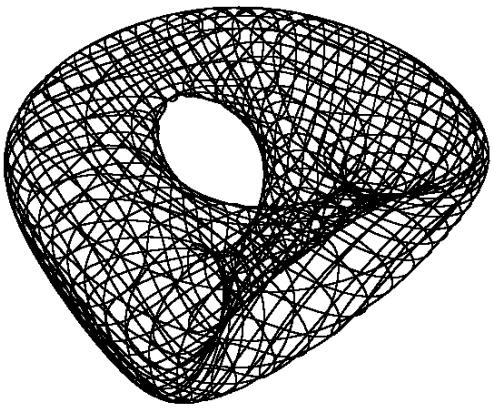


Com que el model original conté plans perpendiculars als eixos, la imatge resultant mostra la intersecció de la superfície implícita amb aquests plans, fet que recorda una representació en filferros. Observeu que el gruix de les línies és variable.

Si mode és 2 (4 punts), mirarem d'aconseguir línies amb un gruix uniforme. Per fer-ho, la decisió de descartar o no el fragment no usarà un llindar constant, sinó que el llindar es calcularà com la variació de f(P) entre el fragment i els seus veïns. Concretament, si abans dibuixàvem els fragments si |v| < eps, ara dibuixarem els fragments si |v| < *width*, on *width* el podeu calcular usant les funcions dFdx, dFdy:

$$width = \left|\frac{\partial v}{\partial x}\right| + \left|\frac{\partial v}{\partial y}\right|$$

En aquest mode, si |v| < *width,* el color del fragment serà **negre**; altrament es descartarà:



Degut a com estan implementades habitualment dFdx, dFdy, el resultat del test serà orientatiu.

Identificadors (ús obligatori):
`implicit.vert, implicit.frag (minúscules!)`
Els uniforms de l'enunciat.