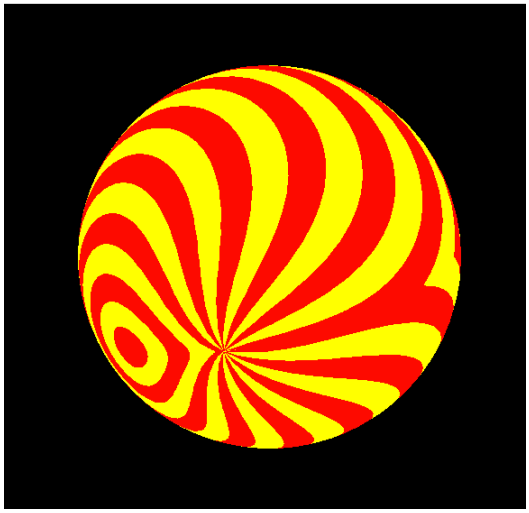
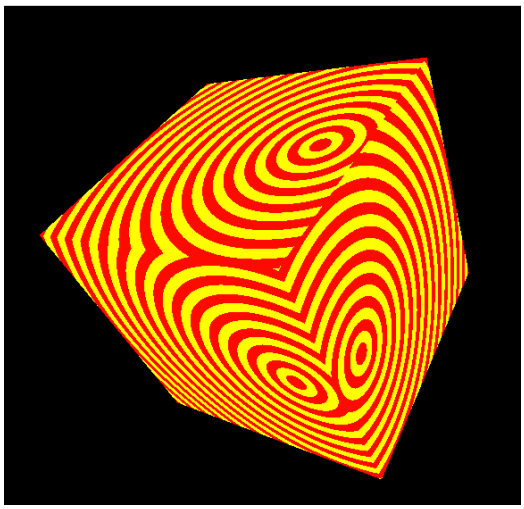
CircularStripes 2.5 punts

Escriu un **fragment shader** que "texturi" el model amb el color d'una textura procedural, tal com mostra la figura. El color del fragment dependrà de les coordenades (s,t) del fragment, i del valor d'un **uniform int nstripes** i d'un altre **uniform vec2 origin**. Més concretament, dependrà de la longitud del vector $(s,t) - origin$, de forma que si aquesta està a l'interval $[0, \frac{1}{nstripes})$, el fragment serà vermell; si està en $[\frac{1}{nstripes}, \frac{2}{nstripes})$ serà groc, i així successivament.





Ambdues figures estan fetes amb $origin = (0.5, 0.15)$ i $nstripes = 29$, amb càmeres arbitràriament rotades.

Identificadors (ús obligatori)

```
stripes.frag
uniform int nstripes;
uniform vec2 origin;
```

Disco-sphere 2.5 punts

Volem simular de forma senzilla una esfera platejada similar a les que es poden veure en algunes discoteques:

Podeu veure el resultat esperat amb l'objecte Sphere i la textura **gold.png** al vídeo **disco-sphere.mp4.**

Observeu que l'esfera gira al voltant de l'eix Y a una velocitat de 0.1 rad/s. Per tant, el **vertex shader** aplicarà al vèrtex una rotació de 0.1**time* radians al voltant de l'eix Y (tot això en object space), i després escriurà gl_Position amb el nou vèrtex (en clip space). També caldrà que el **vertex shader** escrigui la posició del vèrtex en eye space en una variable varying que farà servir el fragment shader.

El **fragment shader** calcularà el color del fragment utilitzant una textura

uniform sampler2D sampler;

de la següent manera. Primer calcularà una aproximació de la normal **n** (en eye space) fent servir les funcions dFdx, dFdy, de forma anàloga a l'exercici *calculant la normal al fragment shader*. Això és necessari perquè shadermaker només proporciona normals per vèrtex, però en aquest shader ens interessa una normal que sigui aproximadament igual per tots els fragments d'una mateixa cara.

Després farà servir les components ($\mathbf{n}_x$, $\mathbf{n}_y$) de la normal per calcular el color del fragment com el producte

$$T * \mathbf{n}_z$$

on T és el color del texel al punt (s,t) = ($\mathbf{n}_x$, $\mathbf{n}_y$) de la textura.

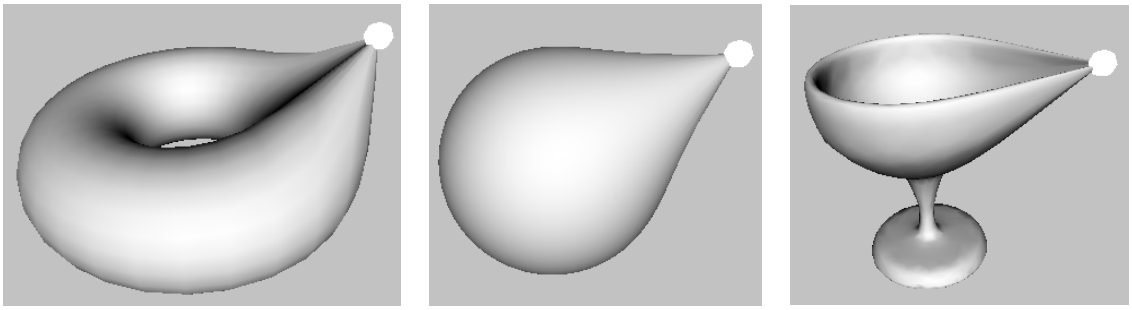
Identificadors (ús obligatori)

```
disco-sphere.vert
disco-sphere.frag
uniform float time;
uniform sampler2D sampler;
```

Magnet 2.5 punts

Volem deformar la malla com si la font de llum GL_LIGHT0 fos una mena d'imant que exerceix una força sobre els vèrtexs propers de la malla.

Aquí teniu el resultat esperat amb diferents objectes, amb GL_LIGHT0 a la posició (1.0, 1.0, 1.0, 1.0):



Escriu un **vertex shader** que calculi la nova posició del vèrtex V' (en object space) com a interpolació lineal entre la posició original del vèrtex V i la posició de la llum F,

$$V' = (1.0-w)V + wF$$

on el paràmetre d'interpolació lineal w el calcularem com

$$w = \text{clamp}(1/d^n, 0, 1)$$

on d és la distància entre V i F, i el paràmetre n és un **uniform float n** que controla la velocitat amb la que decreix la influència de la llum sobre els vèrtexs (en tots els exemples hem fet servir n=8).

Nota: recordeu que GLSL us donarà la posició del font de llum en eye space, però heu de fer els càlculs anteriors en object space.

Un cop calculat el nou vèrtex, el VS escriurà gl_Position en clip space com és usual, i calcularà el color del vèrtex utilitzant directament la component z del vector normal en eye space (similar a l'exercici de basic lighting per vèrtex).

Els vídeos **magnet*.mp4** mostren el resultat esperat amb una llum a la posició (1.0, 1.0, 1.0, 1.0), activant les opcions "Show lights" i "Auto-rotate" del ShaderMaker.

Identificadors (ús obligatori)

```
magnet.vert
uniform float n;
```

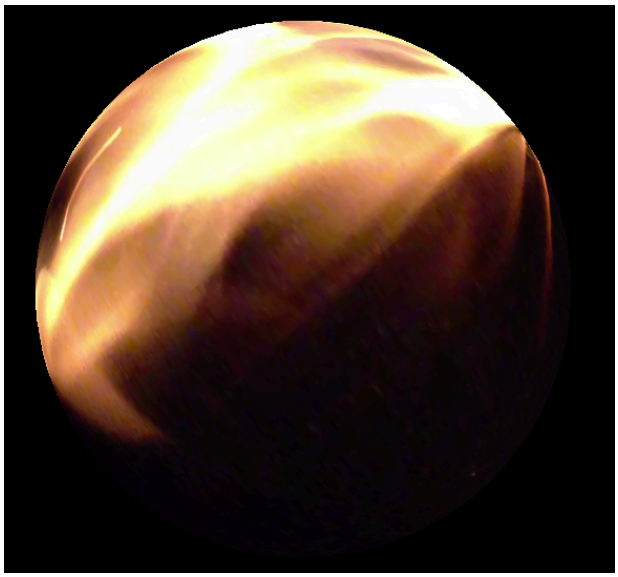
Fire 2.5 punts

Quan volem mostrar quelcom cremant, podem recórrer a diverses tècniques per a simular foc. Una de molt senzilla consisteix en mostrar en seqüència diverses textures capturades d'un foc real, animant-les en el temps.

Programa un **fragment shader** que mostri seqüencialment quatre textures foc-1.png, foc-2.png, foc-3.png i foc-4.png. Per a controlar la velocitat de l'animació, hauràs de disposar un **uniform float slice**, de forma que segons el valor de time es mostri una textura diferent:

- $0 \leq time < slice$: mostra foc-1
- $slice \leq time < 2 * slice$: mostra foc-2
- $2 * slice \leq time < 3 * slice$: mostra foc-3
- $3 * slice \leq time < 4 * slice$: mostra foc-4
- i així successivament, de forma cíclica.

Aquí veieu un frame de l'esfera cremant:



Identificadors (ús obligatori)

```
fire.frag
uniform float slice;
uniform sampler2D sampler0;
uniform sampler2D sampler1;
uniform sampler2D sampler2;
uniform sampler2D sampler3;
```

recordeu que cal associar cada unitat de textura amb el corresponent sampler, i que si compileu un codi que no fa servir algun dels samplers haureu de tornar a definir la seva associació per a que funcioni correctament.