Twist 2.5 punts

Escriu un **vertex shader** que apliqui a cada vèrtex una **transformació de modelat** consistent en una rotació de θ_y radians respecte l'eix Y del model.

L'angle de rotació θ_y l'heu de calcular com

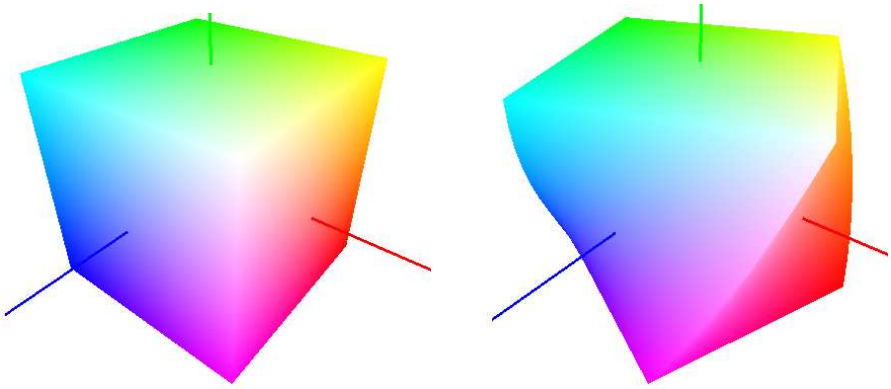
$$\theta_y = 0.4\ y\ \sin(t),$$

on y és la coordenada y del vèrtex en object space, i t és el temps en segons. Recordeu que la rotació d'un punt respecte l'eix Y es pot calcular multiplicant aquesta matriu pel punt:

$$\begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{pmatrix}$$

El VS també haurà de fer les tasques habituals (pas a clip space i propagació del color que li arriba, **sense il·luminació**).

Aquí teniu el resultat esperat amb el cub, en els instants de temps t=0, t=0.5:



Aquests són alguns fotogrames del vídeo **twist.mp4:**

Identificadors (ús obligatori)

```
twist.vert
uniform float time;
```

Marble 2.5 punts

Escriu un **vèrtex shader** i un **fragment shader** que simulin l'aparença del marbre. El **VS** haurà de proporcionar al FS la informació que necessita (normal i posició en object space). El **FS** calcularà el color de cada fragment de la següent manera. Primer, caldrà generar unes coordenades de textura (s,t) utilitzant la tècnica basada en dos plans S i T:

$$s = a_s x + b_s y + c_s z + d_s w \qquad t = a_t x + b_t y + c_t z + d_t w$$

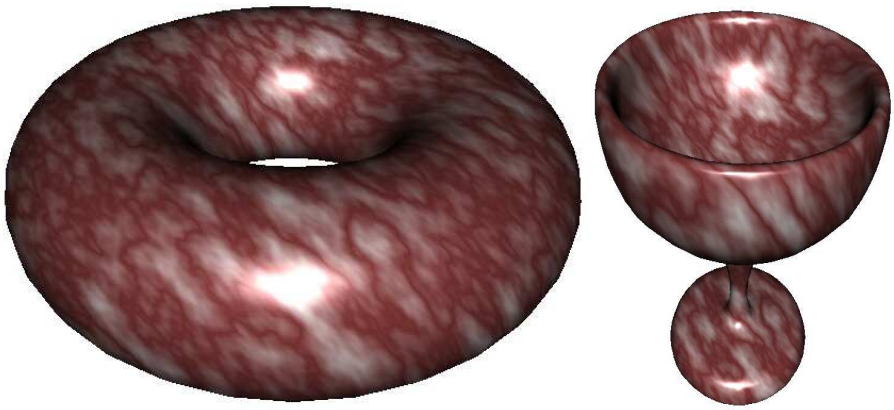
on [x y z w] són les coordenades del punt en **object space**, i els plans S i T els heu definir com:

$$(a_s\, b_s\, c_s\, d_s) = 0.3*(0, 1, -1, 0) \qquad (a_t\, b_t\, c_t\, d_t) = 0.3*(-2, -1, 1, 0)$$

Un cop calculades les coordenades de textura (s,t) al FS, calcularu un valor *v* en [0,1] prenent una mostra del canal vermell (r) de la textura **noise.png** al punt (s,t). El color difós de l'objecte el calcularu com un gradient de color que estarà format per la interpolació d'aquests tres colors: *white*, *redish* = (0.5, 0.2, 0.2, 1.0) i *white*, utilitzant el valor *v*. El càlcul s'ha de fer de forma que *v*=0 doni *white*, *v*=0.5 doni *redish*, i *v*=1.0 doni un altre cop *white*. Després de calcular aquest color difós, el color final del fragment serà el que retorni la funció **shading**, que haureu de cridar amb N i V essent vectors unitaris en **eye space**, i *diffuse* essent el color difós calculat prèviament.

```
vec4 shading(vec3 N, vec3 V, vec4 diffuse) {
       const vec3 lightPos = vec3(0.0,0.0,2.0);
       vec3 L = normalize( lightPos - V );
       vec3 R = reflect(-L,N);
       float NdotL = max( 0.0, dot( N,L ) );
       float RdotV = max( 0.0, dot( R,V ) );
       float Ispec = pow( RdotV, 20.0 );
       return diffuse * NdotL + Ispec;
}
```

Aquí teniu exemples dels resultats esperats amb la textura **noise.png** i el torus i el glass.obj:



Identificadors (ús obligatori)

```
marble.vert   marble.frag
uniform sampler2D noise;
```

Ironed 2.5 punts

Escriu un **vertex shader** que simuli una mena de *planxat* del model.

Amb el vèrtex en object space, la coordenada Y s'haurà de calcular com:

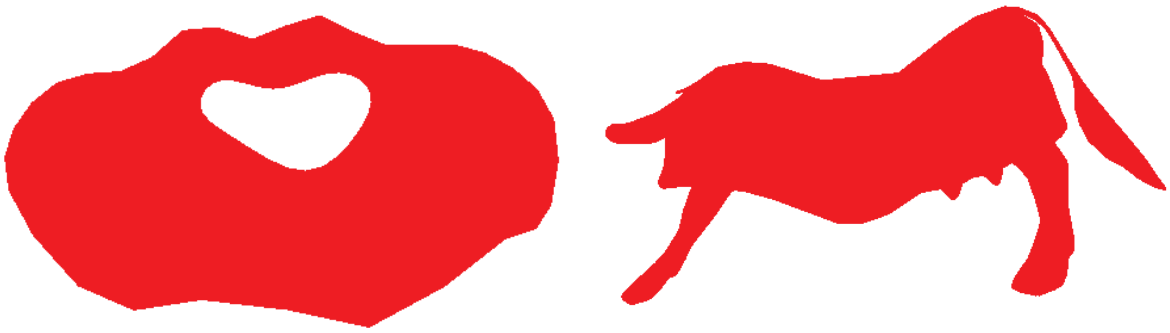
$$A \sin(2\pi x + 3t)$$

on A és l'amplitud (uniform float **amplitude**), x és la coordenada X del vèrtex en object space, i t és el temps. La transformació de modelat anterior només afecta a la coordenada Y.

El VS haurà de transformar el vèrtex a clip space, com és usual.

Feu que el color final del vèrtex sigui vermell, (1.0, 0.0, 0.0).

Aquí teniu el resultat esperat amb el torus i la vaca, amb **amplitude** = 0.1:



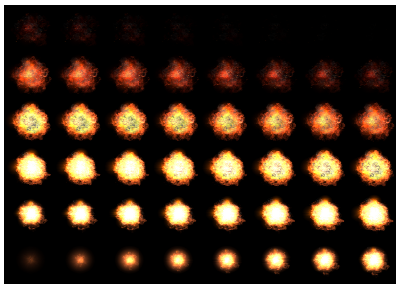
També podeu veure el resultat desitjat, des de diferents punts de vista, a **ironed.mp4.**

Identificadors (ús obligatori)

```
ironed.vert
const float PI = 3.14159;
uniform float amplitude;
```

Explosion 2.5 punts

Una forma molt senzilla de simular una explosió consisteix en mostrar en seqüència diverses imatges capturades d'una explosió real (o simulada), animant-les en el temps. Per comoditat, podem ajuntar totes les imatges (*frames* de l'animació) en una única textura. Per aquest exercici usarem la textura RGBA **explosion.png** (cortesia d'April Young) que conté 8x6=48 frames, organitzats per files:



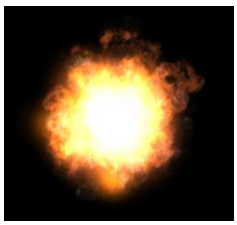
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
08	09	10	11	12	13	14	15
00	01	02	03	04	05	06	07

Programa un **fragment shader** que mostri seqüencialment cadascun d'aquests frames, modificant convenientment les coordenades de textura. Per simplificar l'exercici, la velocitat de l'animació serà de 30 frames per segon, i per tant cada frame es mostrarà durant un slice de 1/30 segons. D'acord amb el valor de time, el frame que s'haurà de mostrar és:

- $0 \leq time < slice$: mostra frame 0
- $slice \leq time < 2 * slice$: mostra frame 1
- $2 * slice \leq time < 3 * slice$: mostra frame 2
- ...
- $47 * slice \leq time < 48 * slice$: mostra frame 47
- i així successivament, de forma cíclica.

Observeu que els frames estàn organitzats per files; el frame 0 ocupa en espai de textura el rectangle (0,0)→(1/8, 1/6), el frame 1 ocupa (1/8, 0)→(2/8,1/6), el frame 8 ocupa (0,1/6)→(1/8, 2/6), i així successivament. Les coordenades (s,t) que haureu d'utilitzar per accedir a la textura seran la suma de dos termes: un offset que depèn del frame (per exemple, (1/8, 1/6) pel frame 1), i un altre terme que depèn de gl_TexCoord[0].st escalat per (1/8, 1/6).

El color final del fragment l'heu de calcular com *color*.a * *color*, és a dir, el color de la mostra de la textura multiplicat per la seva component alpha. Aquí veieu un frame del model del pla (vegeu també **explosion.mp4**):



Identificadors (ús obligatori)

```
fire.frag
uniform float time;
uniform sampler2D explosion;
```