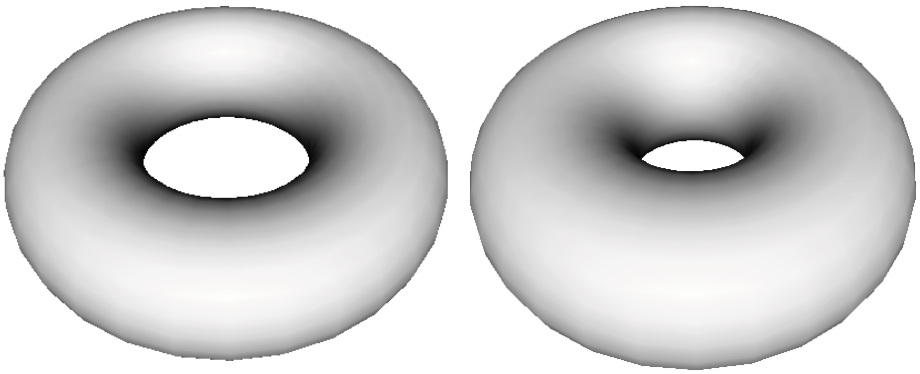
Animate vertices (1)

Escriu un **vertex shader** que, abans de transformar el vèrtex a clip space, el mogui una certa distància *d(t)* en la direcció de la seva normal. Calculeu el valor de *d(t)* com una sinusoïdal amb una certa amplitud *A* i freqüència F (tots dos uniform float).

Feu servir el uniform time que us proporciona el Shader Maker.

Aquí teniu dos frames de l'animació, amb amplitud 0.1



Teniu un vídeo d'exemple amb amplitud = 0.1 i freqüència = 1Hz a animate-vertices-1.mp4

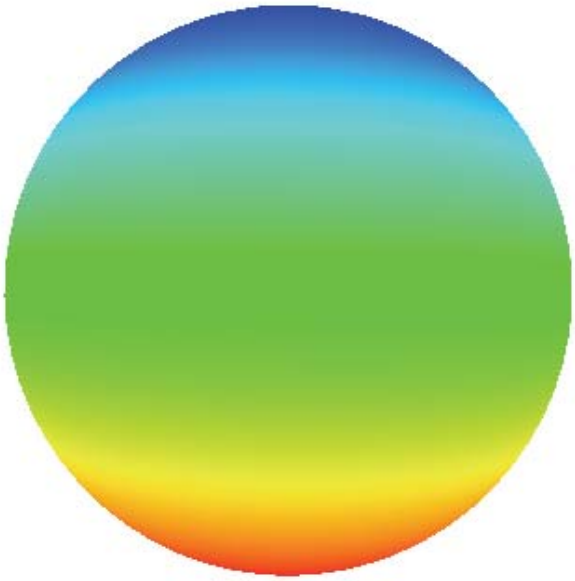
Gradient de color (1)

Escriu un **vertex shader** que apliqui un gradient de color al model segons la seva coordenada Y en object space. L'aplicació rebrà dos uniforms minY, maxY amb els valors extrems de la coordenada Y del model.

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 y=minY es pintin de vermell, i els vèrtexs amb y=maxY es pintin de blau.

Per 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 la imatge que s'espera amb el model de l'esfera:



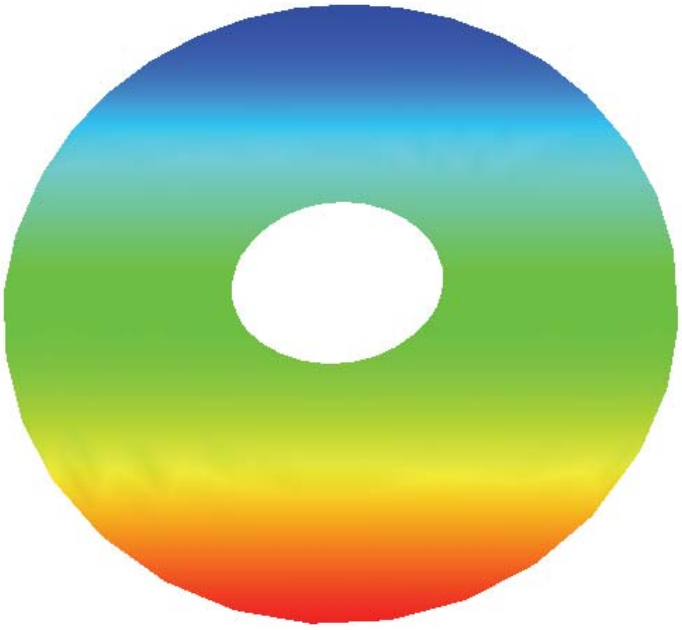
Gradient de color (2)

Escriu un **vertex shader** que apliqui un gradient de color al model segons la seva coordenada Y en coordenades normalitzades de dispositiu (és a dir, després de la divisió de perspectiva).

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 y=-1.0 es pintin de vermell, i els vèrtexs amb y=1.0 es pintin de blau.

Per 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 la imatge que s'espera amb el model del torus:

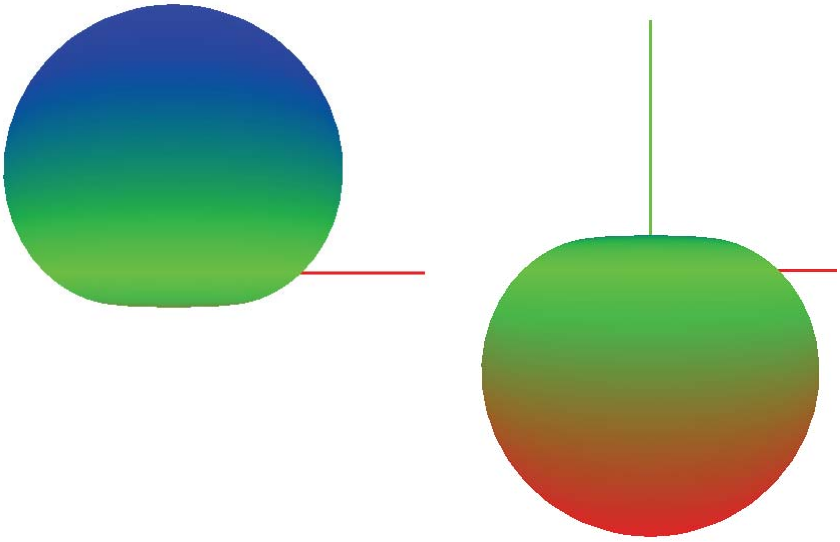


Oscillate

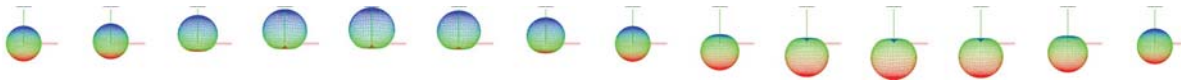
Escriu un vertex shader que pertorbi cada vèrtex del model en la direcció de la seva normal, desplaçant-lo una distància *d* que variï sinusoidalment amb amplitud igual a la component *y* del vèrtex i període 2π segons. Per calcular la amplitud s'agafarà la component *y* en eye space si el **uniform bool eyespace** és cert; altrament s'agafarà la component *y* en object space.

El VS haurà d'assignar com a color del vèrtex directament **gl_Color**.

Aquí teniu els resultats esperats amb la esfera, després de π/2 i 3 π/2 segons (per aquesta vista concreta el resultat no depèn del uniform eyespace) .



Un cicle complet:

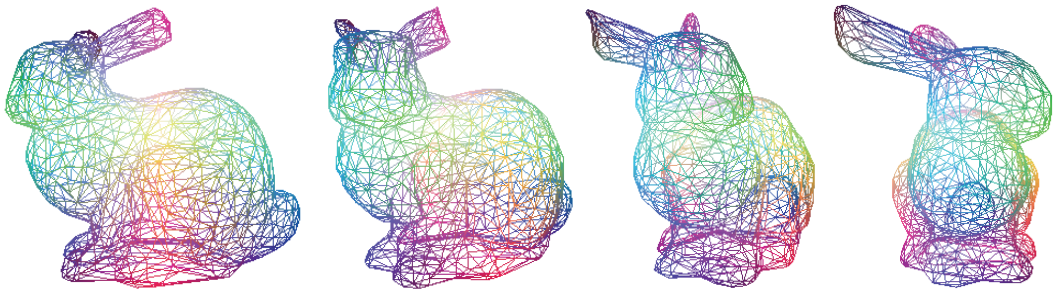


Auto-rotate

El shader maker té una opció per auto-rotar la llum, però no per auto-rotar el model.

Escriu un **vertex shader** que, abans de transformar cada vèrtex, li apliqui una rotació al voltant de l'eix Y. El shader rebrà un **uniform float speed** amb la velocitat de rotació angular (en rad/s). Feu servir la variable **uniform float time** per l'animació.

Aquí teniu els resultats (en wireframe) amb el bunny, amb speed = 0.5 rad/s:



time=0 time=1 time=2 time=3

Recordeu que la rotació d'un punt respecte l'eix Y es pot calcular multiplicant aquesta matriu pel punt:

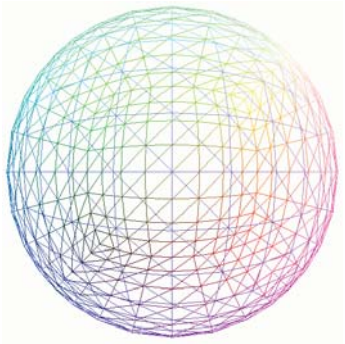
$$\begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{pmatrix}$$

Spherize

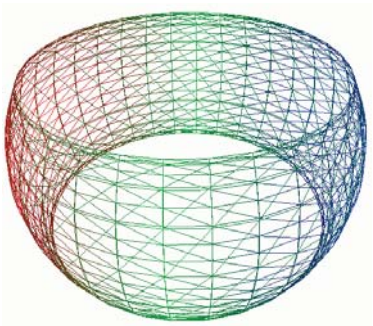
Escriu un **vertex shader** que, abans de transformar cada vèrtex, el projecti sobre una esfera de radi unitat centrada a l'origen del sistema de coordenades del model.

La projecció es farà en la direcció del vector que uneix l'origen del sistema de coordenades del model amb el vèrtex (òbviament en model space). Aquesta projecció és similar a l'O-mapping del centroide estudiat a classe, amb el centroide a l'origen.

Aquí teniu els resultats esperats (en wireframe) amb diferents models:



Cub



Torus

Cow

CRT display

Escriu un **fragment shader** que simuli l'aparença de les imatges dels antics tubs CRT. Per aconseguir aquest efecte, caldrà eliminar (*discard*) tots els fragments d'algunes línies del viewport. En concret, caldrà que només sobrevisquin els fragments d'una de cada *n* línies, on *n* és un **uniform int** proporcionat per l'usuari.

Observació: quan feu servir gl_FragCoord, tingueu en compte que per defecte les coordenades (x,y) en window space fan referència al centre del píxel. Per exemple, un fragment a la cantonada inferior esquerra de la finestra té coordenades (0.5, 0.5).

Aquí teniu dos exemples amb el torus, amb n={2, 4, 6}.