Sistemes de Coordenades i Transformacions Geomètriques

1. Animate Vertices 1 (animate-vertices1.*)

Escriu un **vertex shader** que mogui el vèrtex una certa distància *d(t)* en la direcció de la seva normal (en *object space*). Un cop desplaçat, caldrà escriure gl_Position en *clip space*, com habitualment.

Calculeu el valor de d(t) com una sinusoïdal amb una certa amplitud i freqüència:

```
uniform float amplitude = 0.1;
uniform float freq = 1; // expressada en Hz
```

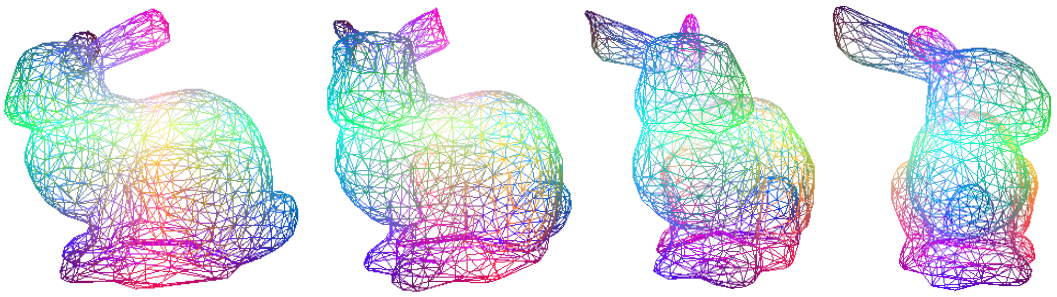
Feu servir el **uniform time** que us proporciona el viewer.

Calculeu el color del vèrtex com el gris que té per components la Z de la normal en eye space.

2. Auto-rotate (auto-rotate.*) (1er control laboratori, curs 2011-12, Q2)

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:

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

El VS escriurà el color original del vèrtex, sense cap il·luminació.

3. Twist (twist.*) (1er control de laboratori, 2013-14 Q1)

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.

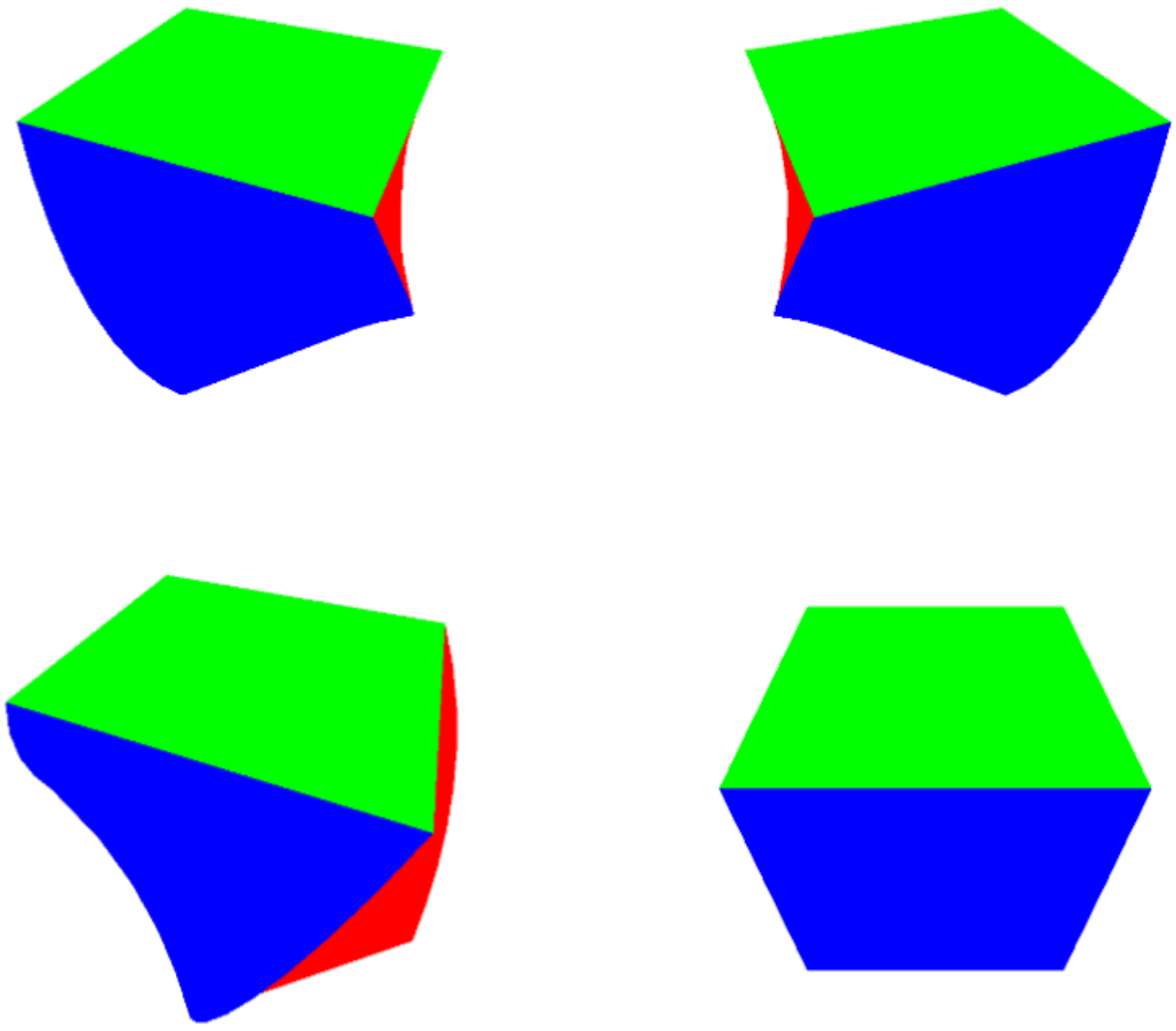
L'angle de rotació θ_y l'heu de calcular com

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

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:

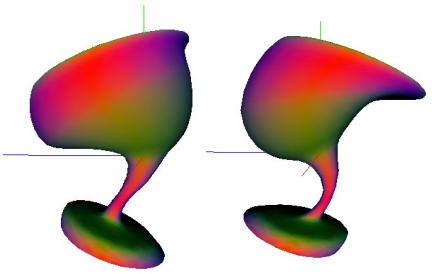
$$\begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{pmatrix}$$

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



4. Wave (wave.*) Control 18/19Q2

Escriu **VS+FS** per deformar l'objecte com si fes una ona:



El VS simplement ha d'aplicar al vèrtex (en *object space*) una **rotació** respecte l'eix X d'un cert angle ϕ. Aquest angle variarà amb el temps segons una sinusoïdal amb aquests paràmetres (assumint radiants):

- Amplitud: la donada per un **uniform float amp = 0.5**
- Freqüència (en Hz): la donada per un **uniform float freq = 0.25**
- Fase: la **coordenada Y del vèrtex en object space**; per time=0, l'angle ϕ serà amp*sin(vertex.y).

Un cop aplicada la rotació al vèrtex, cal escriure gl_Position en clip space com habitualment. El color de sortida serà el que calcula el VS per defecte (color multiplicat per N.z en eye space).

El FS farà les tasques per defecte.

Matriu de rotació al voltant de l'eix X:

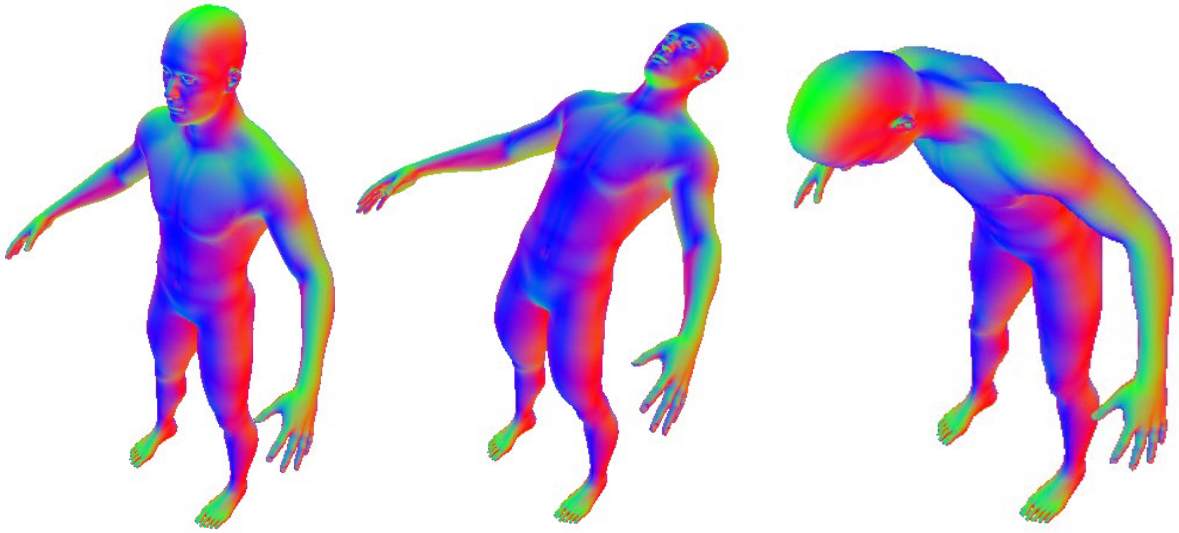
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

Identificadors obligatoris:

```
const float pi = 3.141592;
uniform float amp = 0.5;
uniform float freq = 0.25;
```

5. Contortion (contortion.*)

Escriu **VS+FS** que dibuixin el model humà (**man.obj**) amb una rotació que variï en el temps:



El **VS** haurà d'aplicar a cada vèrtex (en *object space*) una rotació respecte un eix paral·lel a l'eix X. L'angle de rotació *A* vindrà determinat per **(y-0.5)·sin(t)**, on y és la coordenada Y del vèrtex en *object space*. Només volem rotar els vèrtexs de la part superior del cos, per tant **l'angle haurà de ser 0 pels vèrtexs amb y per sota de 0.5** (l'alçada aproximada del genoll).

Concretament volem que la rotació es faci respecte d'un eix paral·lel a l'eix X que passi pel punt **(0, 1, 0)** el qual està prop del centre del cos. Per tant, **abans i després** d'aplicar la rotació al vèrtex, haureu d'aplicar la translació corresponent. Recordeu que la matriu de rotació respecte l'eix X és:

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

El VS haurà d'escriure gl_Position com fem habitualment. El color calculat pel VS serà directament el color d'entrada, **sense il·luminació**.

El **FS** simplement escriurà el color que li arriba del VS.

Recordeu no fer cap càlcul innecessari.

Fitxers i identificadors (ús obligatori):

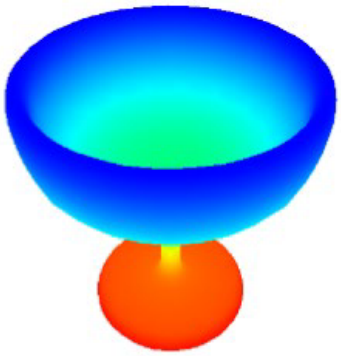
```
contortion.vert, contortion.frag
uniform float time;
```

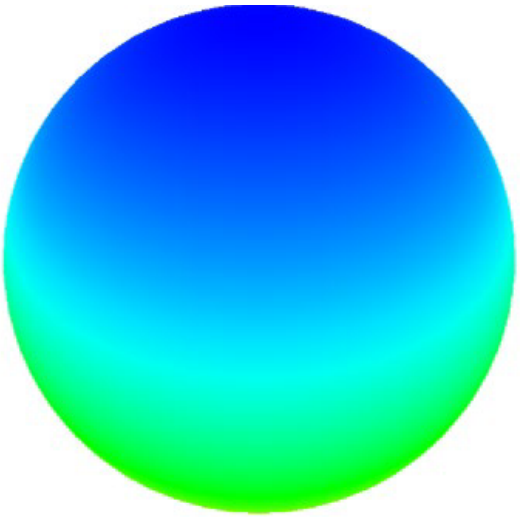
6. Color Gradient 1 (gradient-1.*)

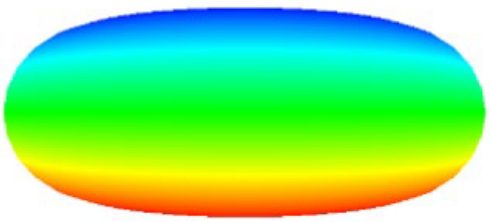
Escriu un vertex shader que apliqui un gradient de color al model segons la seva coordenada Y en *object space*. Feu servir els uniforms amb la capsa englobant de l'escena per obtenir els valors extrems de la coordenada Y del model, minY, maxY.

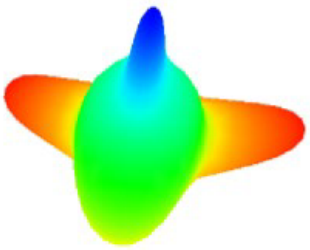
El gradient de color estarà format per la interpolació d'aquests cinc colors: red, yellow, green, cian, blue. L'assignació s'haurà de fer de forma que els vèrtexs amb y==boundingBoxMin.y es pintin de vermell, i els vèrtexs amb y==boundingBoxMax.y es pintin de blau. Per a la interpolació lineal entre 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.

Al color resultant no cal afegir-hi cap il·luminació.









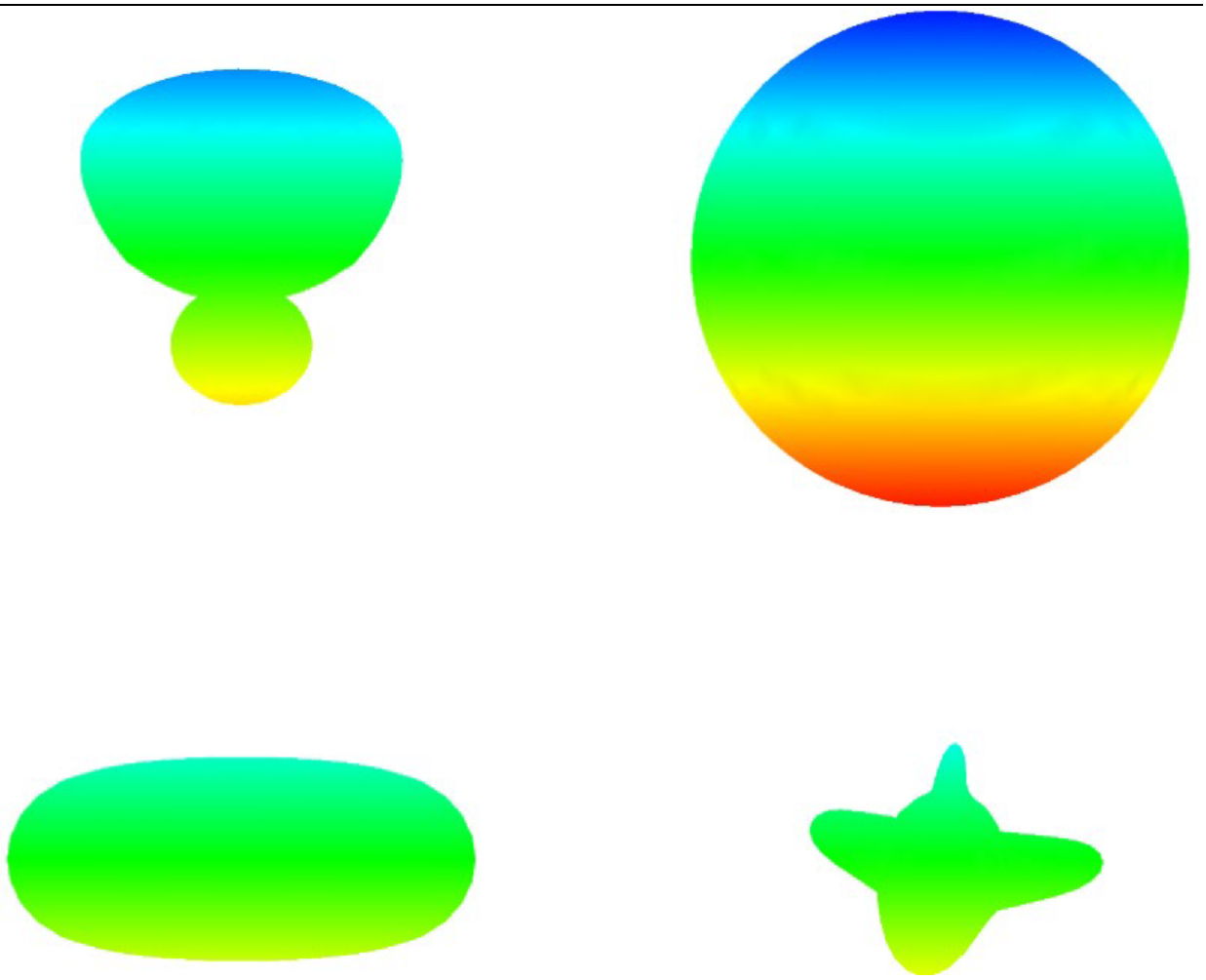
7. Color gradient 2 (gradient-2.*)

Escriu un vertex shader que apliqui un gradient de color al model segons la seva coordenada Y en NDC (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 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.

El resultat dependrà òbviament de la càmera.

Al color resultant no cal afegir-hi cap il·luminació.



8. Reverse Z 1 (reverse-z1.*)

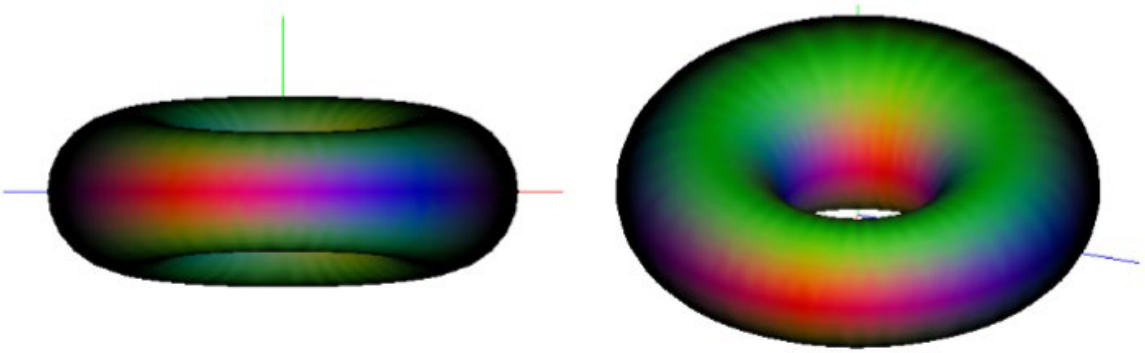
La funció d'OpenGL glDepthFunc permet triar el tipus de depth test a aplicar per OpenGL. Per defecte, el test és GL_LESS, és a dir, un fragment passa el test si la seva Z (en window space) és més petita que la Z emmagatzemada al depth buffer.

Escriu un **vertex shader** que modifiqui la Z dels vèrtexs per tal d'aconseguir el mateix efecte que tindria cridar glDepthFunc amb GL_GREATER des de l'aplicació. El resultat és que s'invertirà la visibilitat de les cares, sent visibles les cares més llunyanes a l'observador.

Cal modular el color per la component Z de la normal en coordenades d'ull.

El color de sortida serà similar al que calcula el VS per defecte: color multiplicat per |N.z| en eye space. El valor absolut és necessari per il·luminar correctament cares backface, que ara seran visibles.

Aquí teniu un exemple amb el model del torus, sense invertir i invertint la Z:



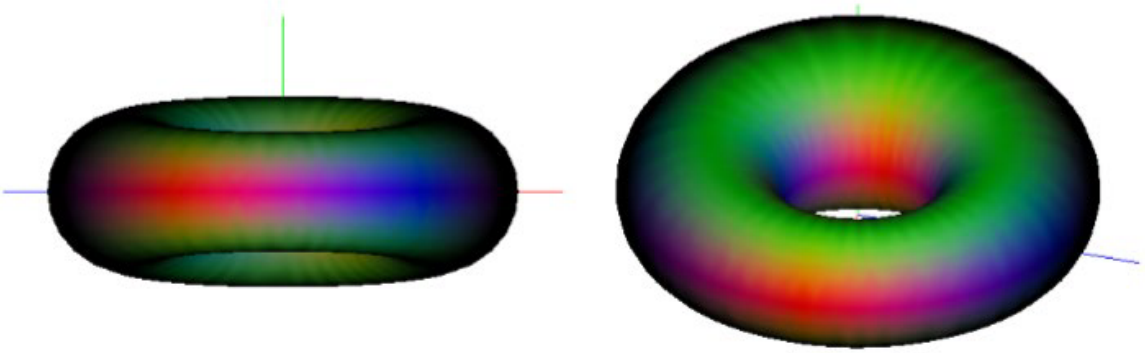
9. Reverse Z 2 (reverse-z2.*)

La funció d'OpenGL glDepthFunc permet triar el tipus de depth test a aplicar per OpenGL. Per defecte, el test és GL_LESS, és a dir, un fragment passa el test si la seva Z (en window space) és més petita que la Z emmagatzemada al depth buffer.

Escriu un fragment shader que modifiqui la Z dels fragments per tal d'aconseguir el mateix efecte que tindria cridar glDepthFunc amb GL_GREATER des de l'aplicació. El vertex shader farà les tasques per defecte. El resultat és que s'invertirà la visibilitat de les cares, sent visibles les cares més llunyanes a l'observador. Cal modular el color per la component Z de la normal en coordenades d'ull.

El color de sortida serà el que calcula el VS per defecte (color multiplicat per N.z en eye space).

Aquí teniu un exemple amb el model del torus (amb shader per defecte, i amb el shader que es demana):



10. Zoom (zoom.*)

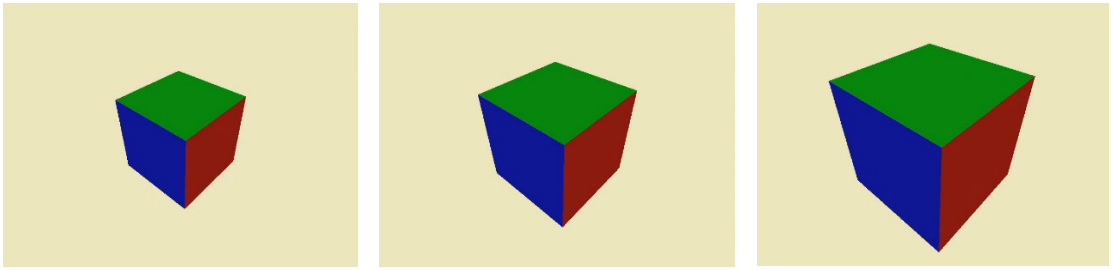
Escriu VS+FS per simular un zoom automàtic que ampliï i redueixi l'objecte a mesura que passi el temps.

El VS serà l'encarregat d'aplicar aquest zoom, que implementarem **escalant les coordenades x,y del vèrtex en NDC**. El factor d'escala serà **0.5+|sin(time)|.**

El color del vèrtex serà simplement el color original multiplicat per la N.z en eye space.

Recordeu que us demanem que apliqueu l'escalat en NDC.

El FS farà les tasques per defecte (que siguin imprescindibles per al resultat demanat).



Identificadors (ús obligatori):

```
zoom.vert, zoom.frag
uniform float time;
```

11. Oscillate (oscillate.*) (1er control laboratori, curs 2012-13, Q1)

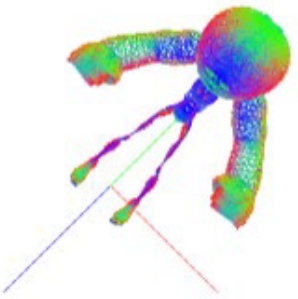
Escriu un vertex shader que pertorbi cada vèrtex del model (en *object space*) en la direcció de la seva normal, desplaçant-lo una distància d que variï sinusoïdalment amb una amplitud d i període 2π segons.

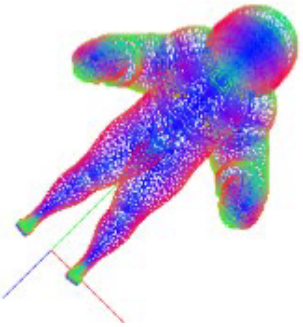
Calculeu la amplitud d com

$$d=(r/10)y$$

on r és la meitat de la mida de la diagonal de la capsa contenidora de l'escena (feu servir boundingBoxMin i boundingBoxMax), y és la coordenada Y del vèrtex, agafada 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 el color inicial.





12. Bouncing (bouncing.*) (1er control de laboratori, 2014-15 Q1)

Escriu un **vèrtex shader** que faci que l'objecte reboti per la pantalla dintre dels límits definits per un **uniform float scale = 8.0** .

Caldrà que apliquis a l'objecte la translació **T** resultant de calcular:

$$\mathbf{T} = \text{scale} * (\mathbf{T0} + \mathbf{V} * \mathbf{t})$$

on la translació inicial és **T0 = (-1, -1, 0)** i la velocitat és **V = (2, 2,0)**.

Si a la fórmula anterior féssim servir t=time, després d'un temps l'objecte sortiria de la pantalla i no tornaria. Per a fer que reboti podem fer que el temps que apliquem a les equacions de moviment no surti d'un cert rang. Hauràs de fer servir la funció **triangleWave** que per un valor de x=0 torna 1, disminueix uniformement fins a 0 a x=1, torna a augmentar uniformement fins a 1 a x=2, i així successivament:

```
float triangleWave(float x) {
    return abs(mod(x, 2) - 1.0);
}
```

A més a més, volem usar diferents valors de temps per cada coordenada. Per tant, en el càlcul de la translació T, en compte de multiplicar directament la velocitat per time, fareu servir un producte (component a component) pel vector 3D

$$\mathbf{t} = (\text{triangleWave}(\text{time} / 1.618),\ \text{triangleWave}(\text{time}),\ 0)$$

Per a que tota la trajectòria de l'objecte sigui visible, un cop aplicada la translació caldrà que apliquis un escalat uniforme als vèrtexs, equivalent a dividir les tres coordenades de cada vèrtex pel **uniform float scale**.

També hauràs d'il·luminar cada vèrtex fent servir el color **vec4(0.3, 0.3, 0.9, 1.0)** i la component Z de la normal en **eye space**.

Identificadors (ús obligatori):

```
bouncing.*
uniform float time;
uniform float scale;
```