Spring (spring.*) 2.5 punts

Escriu **VS+FS** que simulin l'expansió i compressió cícliques del model 3D com si fos una molla (vegeu el vídeo **spring.mp4** al zip de l'enunciat.

El VS s'encarregarà de l'animació, que tindrà dues fases **que es repetiran cada 3.5 segons.**

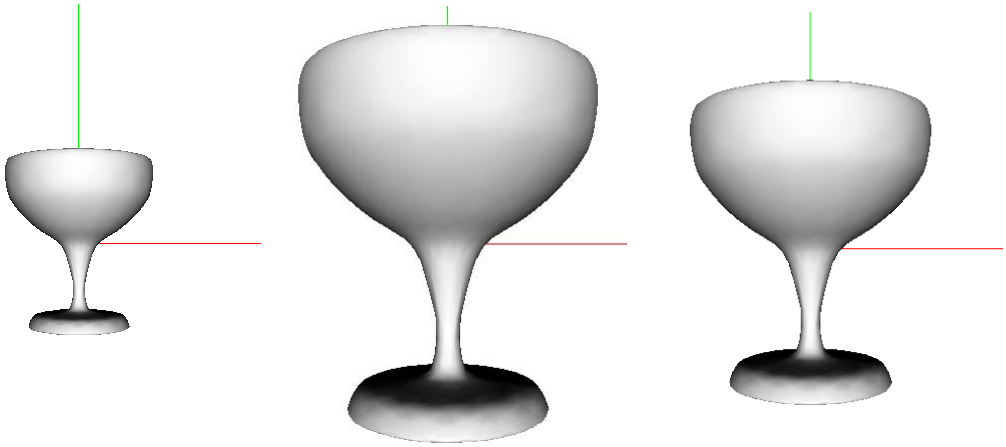
La primera fase (expansió) tindrà una durada de 0.5 segons i mourà els vèrtexs des de l'origen de coordenades fins a la seva posició original en *model space*. Per a calcular la interpolació linial entre aquestes dues posicions, feu que el paràmetre d'interpolació linial sigui $(t/0.5)^3$, on t és el temps en segons *des de l'inici del període* (per exemple, quan time = 4, t = 0.5).

La segona fase (compressió) tindrà una durada de **3 segons**, i mourà els vèrtexs des de la seva posició inicial en *model space* cap a l'origen. Ara però volem que els vèrtexs es moguin a velocitat uniforme. Penseu com heu de calcular el paràmetre d'interpolació lineal, a partir d'un valor t que dins de cada període estarà dins l'interval [0.5, 3.5).

Un cop calculada la posició del vèrtex en *model space*, caldrà transformar-lo a *clip space* com feu usualment. El color del vèrtex serà el gris que té per components la Z de la normal en *eye space*.

El FS farà les tasques per defecte.

Aquí teniu el resultat als instants de temps 0.4, 0.5, 1:



Fitxers i identificadors (ús obligatori):

`spring.vert, spring.frag`

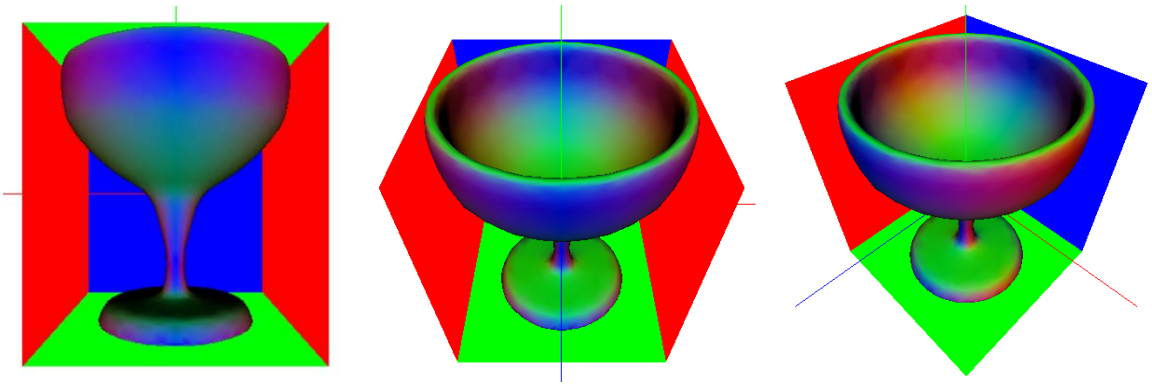
Walls (walls.*) 2.5 punts

Escriu **VS+GS+FS** que dibuixin cada triangle del model de la manera habitual, i a més a més les cares de la capsa englobant de l'escena (boundingBoxMin, boundingBoxMax).

El VS farà les tasques per defecte.

El GS escriurà cada triangle de la manera habitual. A més a més, quan **gl_PrimitiveIDIn** sigui 0, el GS haurà de dibuixar les cares de la capsa englobant de l'escena. El color d'aquestes cares dependrà de l'orientació: vermell per cares perpendiculars a l'eix X, verd per les perpendiculars a l'eix Y, i blau per a les altres dues cares.

Aquestes cares només s'han de dibuixar si són *backface* respecte la càmera, de forma que les cares de la capsa mai no taparan el model:



Per esbrinar si una cara és *backface*, podeu usar el signe de P·N (on P és un vèrtex qualsevol de la cara i N la normal de cara, tots dos en eye space). Alternativament, podeu usar el signe de la distància entre l'observador i el pla (a,b,c,d) de la cara.

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

Fitxers i identificadors (ús obligatori):

`walls.vert, walls.geom, walls.frag`

Equi (equi.*) 2.5 punts

Escriu un **VS** i un **FS** per tal de visualitzar panorames representats amb projecció equirectangular. Al directori **/assig/grau-g/Textures/** trobareu la textura **verandah.png** (Fotografia de Greg 'Groggy' Lehey):

En aquesta textura, un texel (s,t) representa el color en la direcció del vector unitari (x,y,z) determinat per aquestes equacions:

$\theta = 2\pi s$
$\psi = \pi(t-0.5)$

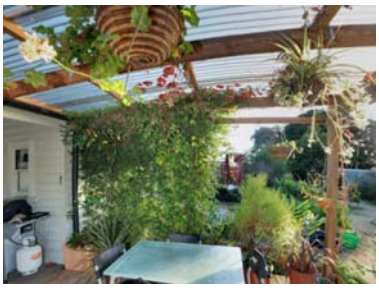
$x = \cos(\psi)\cos(\theta)$
$y = \sin(\psi)$
$z = \cos(\psi)\sin(\theta)$

Aquest exercici està pensat pel model de l'esfera (sphere.obj), amb la càmera situada dins l'esfera.

El VS, a banda d'escriure gl_Position, enviarà al FS les coordenades del punt en **object space**.

El FS agafarà aquest punt (x,y,z), i calcularà les coordenades (s,t) on està representat el que es veu en direcció (x,y,z). Per fer això, heu de calcular primer els angles θ, ψ a partir de (x,y,z) (aïllant-los a les equacions anteriors; caldrà que feu servir les funcions **asin** i **atan** de GLSL), per després calcular les coordenades de textura (s,t) a partir de θ, ψ. El color del fragment serà el color de la textura a (s,t). La funció **atan**(a, b) de GLSL calcula l'arctangent de a/b, retornant un valor dins l'interval $[-\pi, \pi]$.

Aquí teniu el resultat esperat (sphere.obj, càmera situada aproximadament a l'origen):


Fitxers i identificadors (ús obligatori):

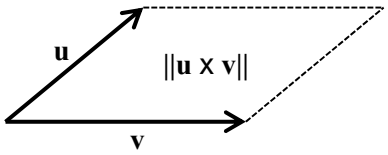
```
equi.vert, equi.frag
uniform sampler2D panorama;
const float PI = 3.141592;
```

Base (base.*) 2.5 punts

Escriu un **plugin** que escrigui pel canal de sortida estàndard (amb cout), l'àrea de la superfície de la meitat inferior del primer objecte de l'escena.

Només heu d'implementar **onPluginLoad**(). Aquest mètode recorrerà les cares del primer objecte de l'escena, i calcularà l'àrea de la seva superfície com la suma de l'àrea de les seves cares. Només cal sumar l'àrea de les cares on la seva y mínima sigui més petita que la Y del centre de la capsa englobant de l'objecte (feu servir el mètode **boundingBox()** de la classe Object).

Podeu assumir que els models seran malles de triangles. L'àrea d'un triangle es pot calcular fàcilment tenint en compte que el **mòdul del producte vectorial** de dos vectors **u** i **v** dóna l'àrea del paral·lelogram definit per **u** i **v**,



El mètode haurà d'escriure pel terminal l'àrea del primer objecte, precedida pel literal "Area:".

Aquí teniu un exemple de resultat, si només està carregat l'objecte per defecte:

Area: 18.6827

Fitxers i identificadors (ús obligatori):

`base.pro, base.h, base.cpp`