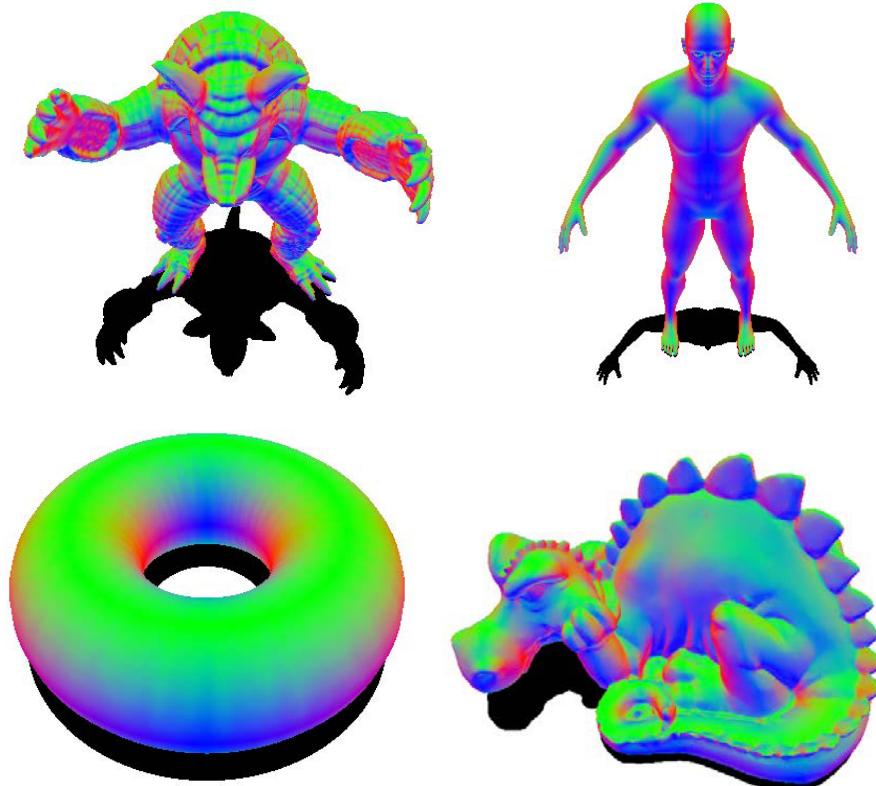

1. Shadow (shadow.*) (2^{on} control laboratori, curs 2011-12, Q2)

Escriu VS+GS+FS que simulin l'ombra que projecta l'objecte sobre el terra, que suposarem situat al pla **Y = boundingBoxMin.y**, respecte una font de llum direccional en la direcció vertical (eix Y).

Per cada triangle (que haurà de rebre amb coordenades en *object space*), el GS haurà d'emetre dos triangles (en *clip space*): un corresponent al triangle original (amb el color sense il·luminació), i un altre (de color negre) corresponent a la projecció del triangle al pla Y anterior.

El GS només ha d'escriure `gfrontColor` i `gl_Position`.

Pel FS us serveix el codi per defecte.



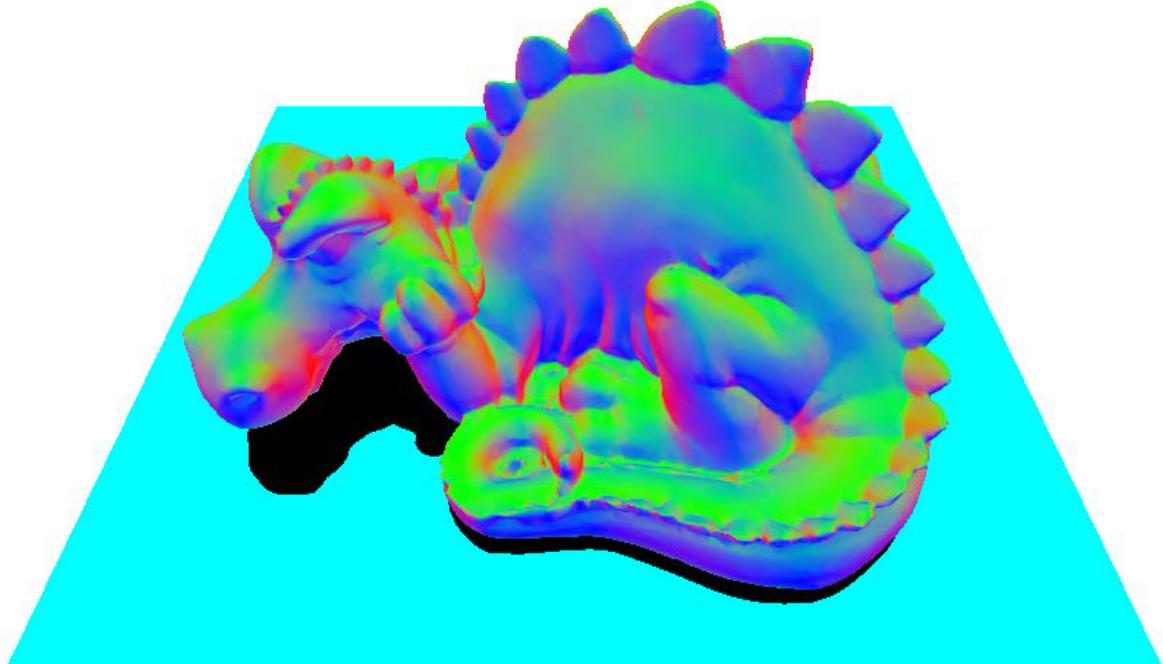
2. Shadow 2 (shadow-2.*)

(2^{on} control laboratori, curs 2012-13, Q1)

Escriu VS+GS+FS que simulin l'ombra que projecta l'objecte sobre el terra, que suposarem situat al pla **Y = boundingBoxMin.y**, respecte una font de llum direccional en la direcció vertical (eix Y).

Per cada triangle (que haurà de rebre amb coordenades en *object space*), el GS haurà d'emetre dos triangles (en *clip space*): un corresponent al triangle original (amb el color sense il·luminació), i un altre (de color negre) corresponent a la projecció del triangle al pla Y anterior.

A més a més, si el GS detecta que està processant la primera primitiva de l'objecte (que podeu detectar amb `gl_PrimitiveIDIn == 0`), haurà d'emetre dos triangles formant un rectangle cian alineat amb els eixos de l'aplicació, i situat una mica (0.01) més avall del pla **Y = boundingBoxMin.y**. Sigui R la meitat de la diagonal de la capsula englobant de l'escena, i sigui C el centre de la capsula. El rectangle tindrà costat $2R$, i estarà centrat al $(C.x, boundingBoxMin.y, C.z)$.



3. Explode (explode.*)

Escriu VS+GS+FS per simular una explosió de l'objecte com la del vídeo **explode-1.mp4**

El VS haurà de passar al GS la posició i la normal de cada vèrtex (tots dos en *object space*).

El GS haurà d'aplicar a cada vèrtex del triangle la translació donada pel vector

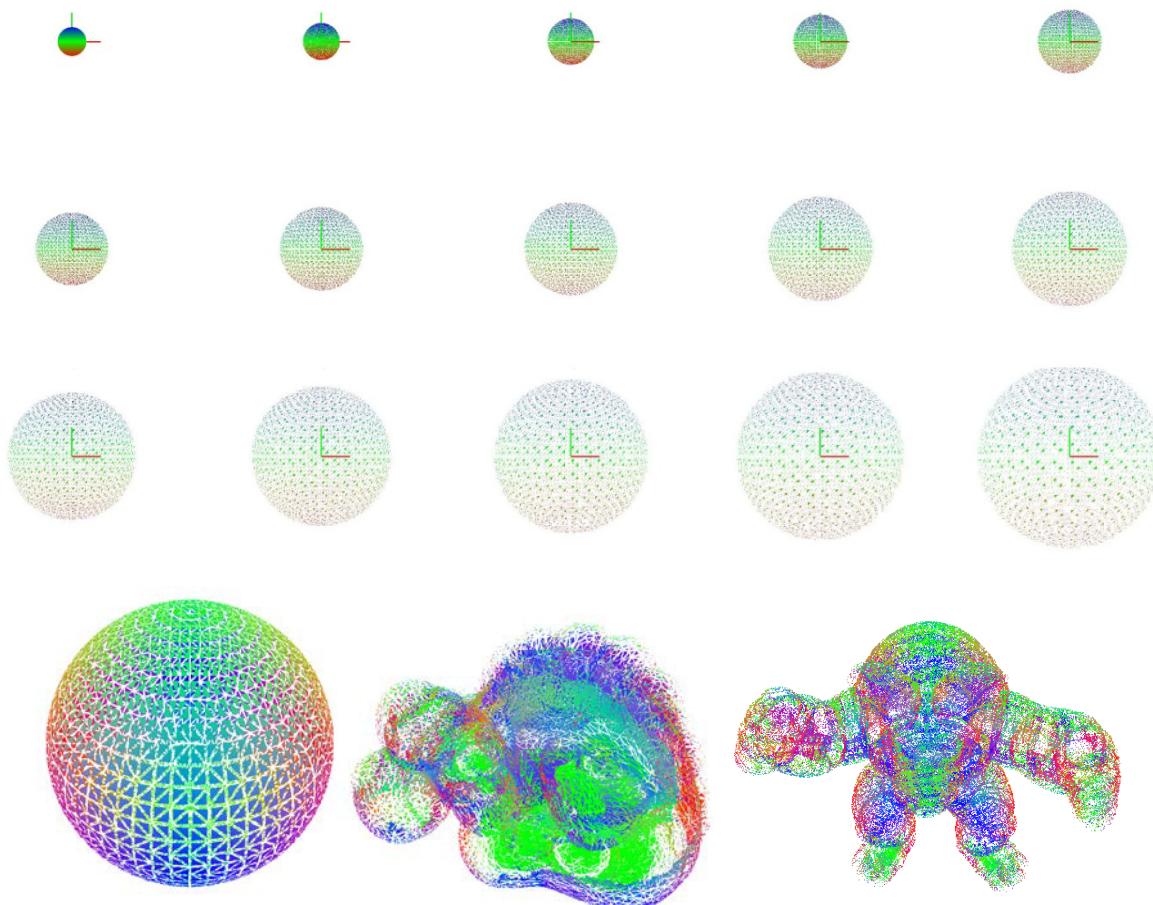
$$\text{speed} \cdot \text{time} \cdot \mathbf{n},$$

on speed és la velocitat desitjada (en unitats del model per segon), time és el temps (en segons), i \mathbf{n} és el promig de les normals del tres vèrtexs del triangle (en *object space*).

Després d'aplicar la translació anterior (en *object space*), el GS haurà de treure els vèrtexs en *clip space*.

Pel vídeo es va fer servir aquest valor:

```
const float speed = 1.2;
```



4. Explode (2)

Escriu VS+GS+FS per simular una explosió de l'objecte com la del vídeo **explode-2.mp4**

El VS haurà de passar al GS la posició i la normal de cada vèrtex (tots dos en *object space*).

El GS haurà d'aplicar a cada triangle la següent transformació geomètrica:

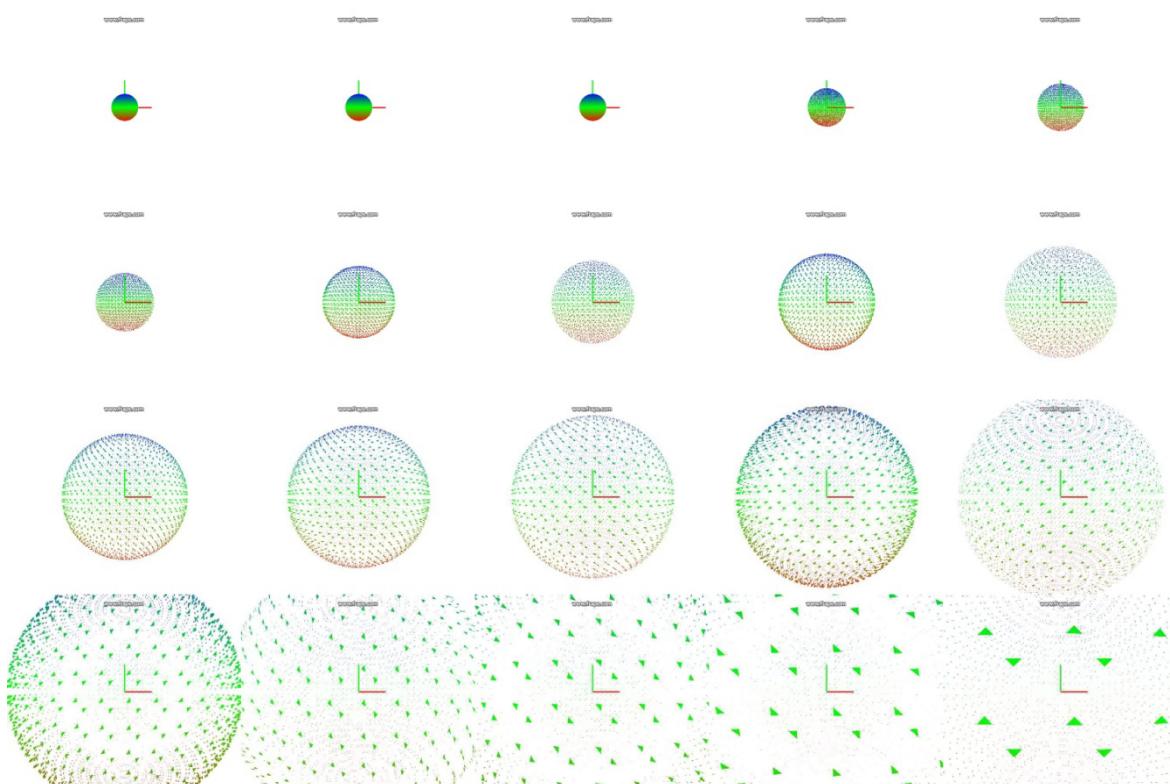
$$T(\text{speed} \cdot \text{time} \cdot \mathbf{n}) \cdot R_z(\text{angSpeed} \cdot \text{time})$$

on speed és la velocitat desitjada (en unitats del model per segon), time és el temps (en segons), \mathbf{n} és el promig de les normals del tres vèrtexs del triangle (en *object space*), angSpeed és la velocitat angular (rad/s) i R_z és una rotació respecte l'eix paral·lel a l'eix Z del model que passa pel baricentre del triangle.

Pel vídeo es van fer servir aquests valors:

```
const float speed = 1.2;  
const float angSpeed = 8.0;
```

Aquí teniu el resultat amb l'esfera:



5. Progressive

(2^{on} control laboratori, curs 2012-13, Q1)

Escriu VS+GS+FS que vagin dibuixant els triangles de l'objecte a mesura que passi el temps, amb una velocitat de 100 primitives per segon.

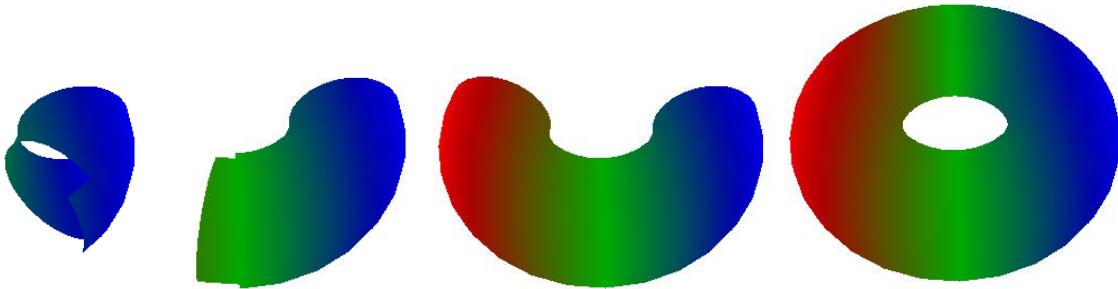
El VS simplement haurà d'escriure `gl_Position` (amb la posició en *object space*) i `vfrontColor` (amb el color que li arriba pel vèrtex).

El GS emetrà un triangle per cada triangle que rebi (és a dir, el comportament habitual d'un GS), però només ho farà pels primers n triangles de l'objecte, on n l'heu de calcular com $n = \lfloor 100t \rfloor$ amb t sent el temps transcorregut en segons (variable *time*).

El FS haurà d'emetre cada triangle amb el color tal qual, sense il·luminació.

Recordeu que podeu saber l'identificador de primitiva amb `gl_PrimitiveIDIn`. La primera primitiva té identificador 0.

Aquí teniu el resultat que s'espera amb un torus.



6. Voxelize (1)

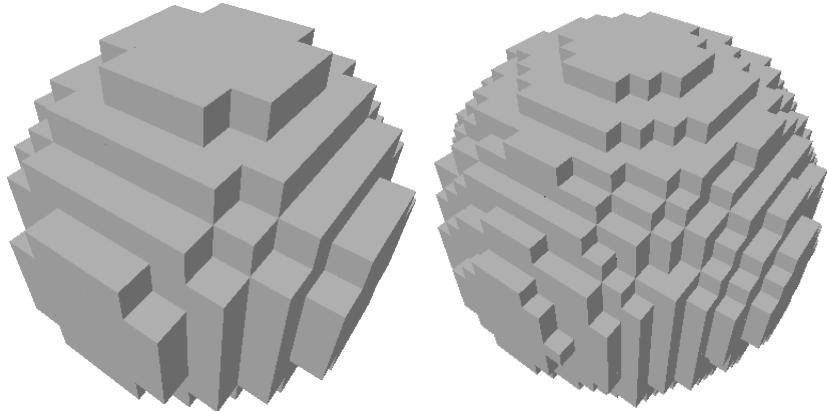
Escriu VS+GS+FS dibuixar una aproximació de la voxelització del model.

La mida de cada voxel vindrà determinada per la variable **uniform float step**.

El VS haurà de passar al GS la posició de cada vèrtex (en *object space*).

El GS haurà d'emetre, per cada triangle d'entrada, les sis cares d'un cub de mida *step* centrat al punt més proper al baricentre del triangle que sigui de la forma $\text{step} \cdot (i, j, k)$, amb i, j, k enters.

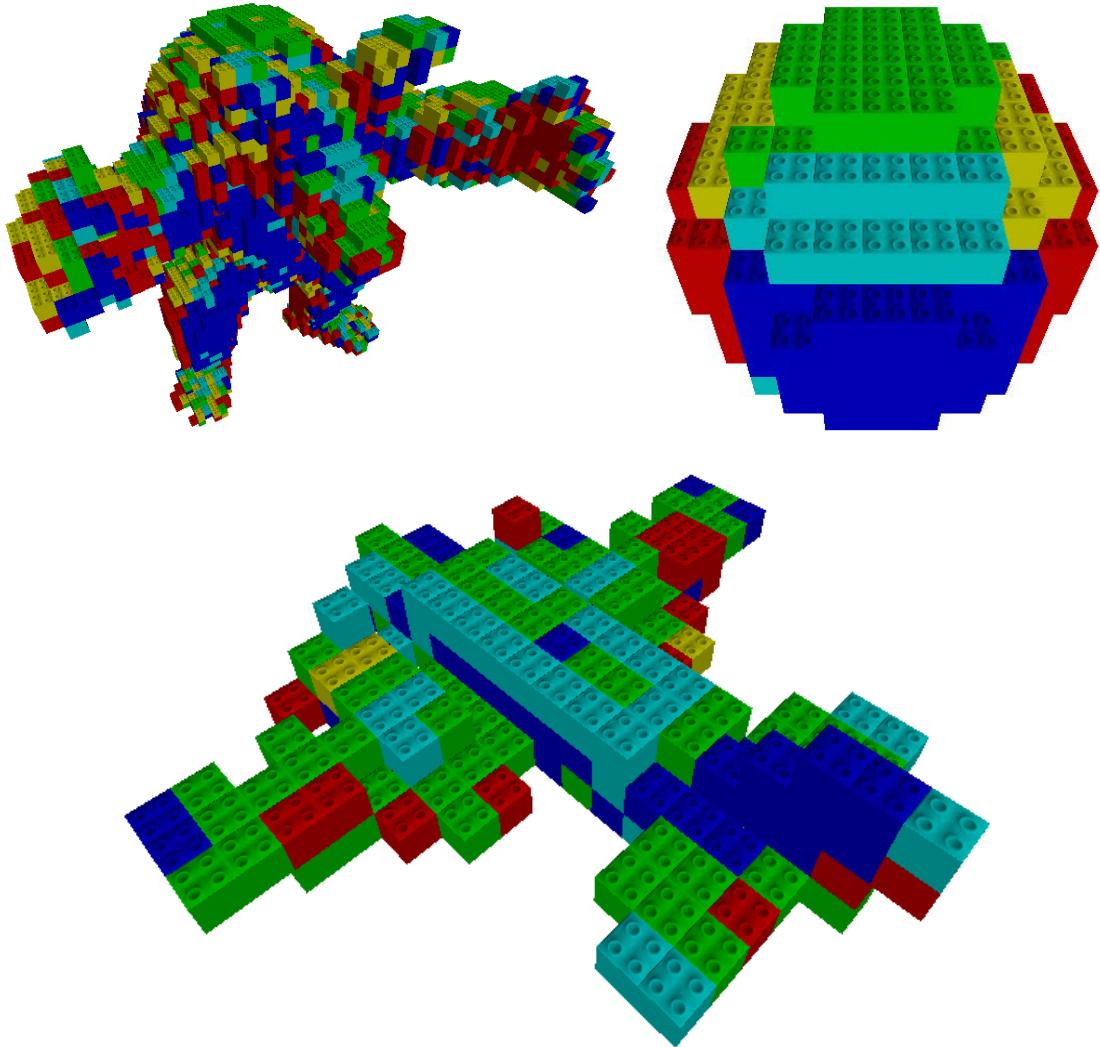
Aquí teniu els resultats amb una esfera amb diferents valors de step:



7. Lego (lego.*)

Escriu VS+GS+FS per dibuixar una aproximació del model amb fitxes de Lego, tal i com indica la figura.

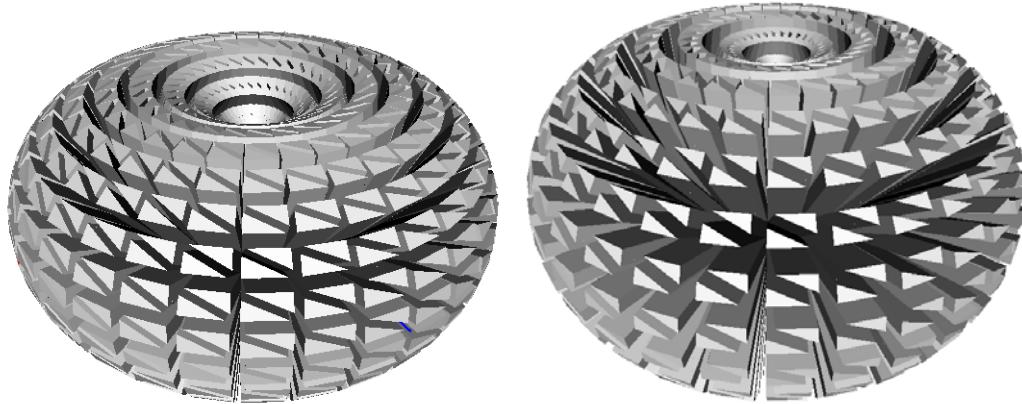
Observa que aquest exercici és similar a l'anterior, però amb cara superior de cada cub texturada i colorejada amb un dels colors bàsics (R,G,B,C,Y) de les peces de Lego (trieu el color més proper, en distància Euclídea, al color del vèrtex). Feu servir la textura lego.png per modular aquest color al FS.



8. Extrude

(2on control de laboratori, 2013-14 Q1)

Escriu un **VS**, **GS** i **FS** per obtenir l'efecte de la figura (torus amb $d=0.5$, $d=1.0$):

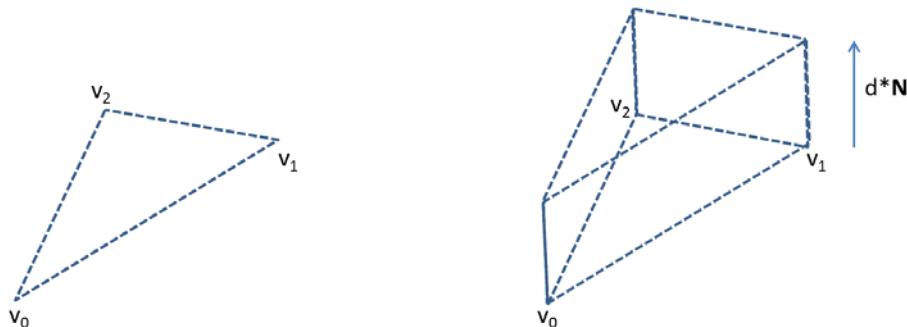


Tasques del **VS**:

- Escriure a `gl_Position` la posició del vèrtex en *object space*.
- Passar al GS la normal del vèrtex, també en *object space*.

Tasques del **GS**:

- El GS haurà de crear, per cada triangle, un prisma de base triangular (vegeu figura). Sigui V_0 , V_1 i V_2 els vèrtexs del triangle original, i N_0 , N_1 , N_2 les seves normals. Sigui N el promig normalitzat de N_0 , N_1 , N_2 . Sigui d un uniform float definit per l'usuari. Els vèrtexs del prisma seran V_0 , V_1 , V_2 , i (V_0+d*N) , (V_1+d*N) i (V_2+d*N) .



- Per cadascun dels vèrtexs del prisma que creï el GS, caldrà que escrigui la normal de la seva cara dins el prisma (en *object space*), així com la seva `gl_Position` (òbviament en *clip space*).

Tasques del **FS**:

- Simplement calcular el color del fragment com el gris que resulta de fer servir la component Z de la normal en *eye space* (similar a l'exercici simple lighting).

9. Grass

(2on control de laboratori, 2013-14 Q1)

Aquest exercici és una continuació de l'anterior. No l'intenteu fer abans de completar Extrude.

Escriu un **VS**, **GS** i **FS** per obtenir un efecte similar al de la figura (plane amb $d=0.1$):



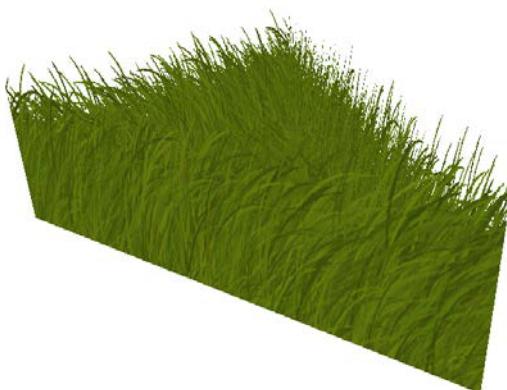
Tasques del **VS**: idèntiques a l'exercici extrude.

Tasques bàsiques del **GS**: també haurà de crear un prisma de base triangular, però sense incloure la cara superior. Al igual que abans, d és un uniform float definit per l'usuari.

Tasques del **FS**:

- Determinarà si es tracta del fragment d'una cara vertical ($N_z==0$, on N_z és la normal en *object space*) o horitzontal (altrament).
- Rebrà també la posició **gPos** del fragment en *object space* (caldrà que el GS li passi).
- Si la cara és vertical, usarà la textura `grass_side`. Com a coordenades de textura, feu servir $\text{vec2}(4 * (\text{gPos.x} - \text{gPos.y}), 1.0 - \text{gPos.z}/d)$. Haureu de descartar el fragment si el texel té alpha < 0.1.
- Si la cara és horitzontal, usarà la textura `grass_top`. Com a coord. de textura, useu $4 * \text{gPos.xy}$.

Aquí teniu un exemple processant només el primer triangle de plane ($d=0.1$):



Cal Iliurar:

`grass.vert` `grass.geom` `grass.frag`

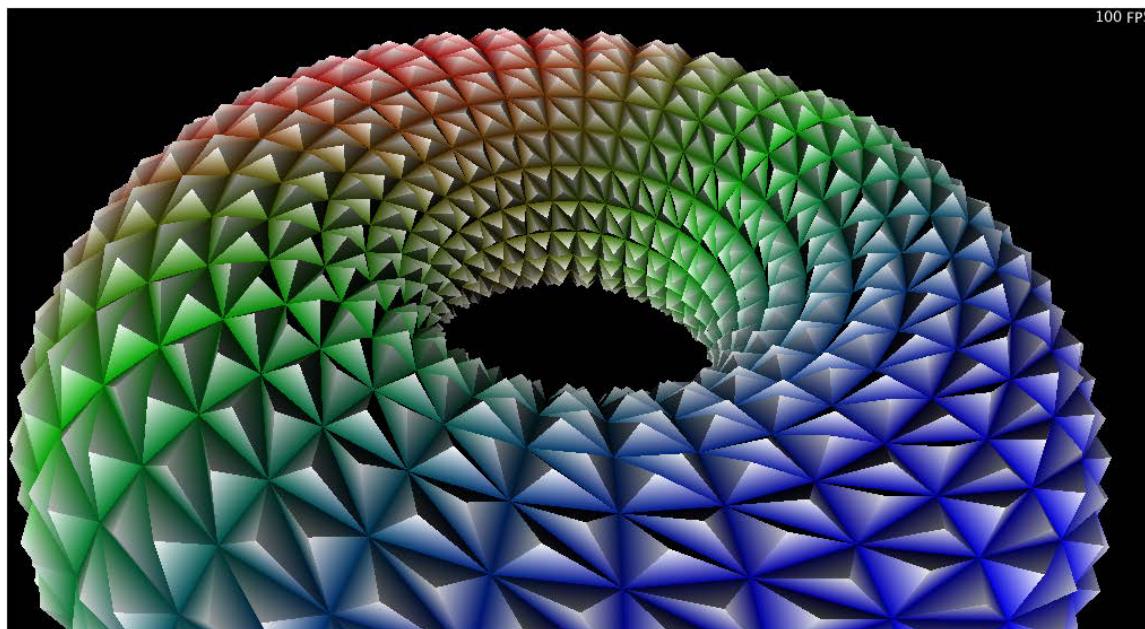
Identificadors:

```
uniform float d;  
uniform sampler2D grass_top, grass_side;
```

10. Spikes

(2on Control de laboratori, 2012-13 Q2)

Escriu un **geometry shader** que permeti generar punxes o cavitats com a la figura. Per a fer-ho, el shader subdivideix cada triangle T que rep en tres, unint els vèrtexs amb el baricentre del triangle T, però desplaçant aquest baricentre en la direcció de la normal de T una distància especificada pel **uniform float disp**. El color associat amb el vèrtex que correspon al baricentre de T serà el blanc. Els altres, conservaran el que tenien, modulat per la z de la normal en coordenades d'ull, com al “Basic Lighting (1)”. El **geometry shader** fa els seus càlculs en coordenades d'ull, i no rep cap més dada que el propi T (amb el color associat a cadascun), i el uniform citat més amunt.

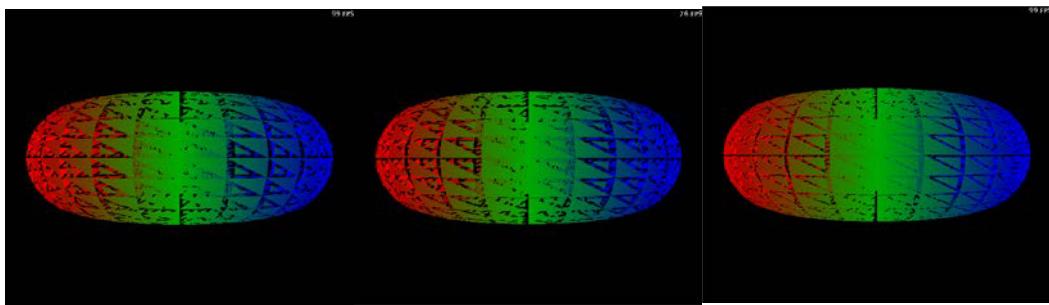


A la figura veiem el torus amb **disp=0.05**

11. Oscillating Shrink

(2on Control de laboratori, 2012-13 Q2)

Escriu un **geometry shader** que copii en sortida la meitat dels triangles, i enconeixi progressivament l'altra meitat. Quins triangles reben quin tractament anirà canviant amb el temps. El shader rebrà un **uniform float speed**, i el **uniform float time**, de forma que quan $0 \leq \text{time} < 1.0/\text{speed}$, es copiaran en sortida els que tinguin un identificador (`gl_PrimitiveIDIn`) parell, i entre $1.0/\text{speed} \leq \text{time} < 2.0/\text{speed}$, els que el tinguin senar, i així continuará alternant cada $1.0/\text{speed}$ segons. Pels demés triangles, farem servir la part fraccionaria de $\text{time} * \text{speed}$ per a obtenir una nova posició dels vèrtexs, interpolada entre la posició original i la del baricentre. El pes de la interpolació variarà sinusoidalment, de manera que serà zero quan $\text{time} * \text{speed}$ és enter, i positiu altrament, atenyent un màxim (1) quan $\text{time} * \text{speed}$ és un múltiple d' $1/2$, moment en el qual tots tres vèrtexs coincidiran amb el baricentre.



speed = 1.0, time= 0.2 speed= 1.0, time = 1.2 speed = 0.2, time = 9.4

El torus vist amb la càmera per defecte del ShaderMaker, amb diferents valors de speed i time.

Podeu veure l'animació resultant a shrink.mp4.

12. Spherical Points

(2on control de laboratori, 2013-14 Q2)

Volem veure els vèrtexs dels models, en comptes de les seves cares. A més, volem veure'ls com petites esferes. Per a aconseguir-ho, programa un *vertex shader*, un *geometry shader* i un *fragment shader*, d'acord amb les següents indicacions.

El geometry shader ignorarà tots els vèrtexs rebut, llevat del primer de cada primitiva. Per cada un d'aquests primers vèrtexs, emetrà un quad en sortida (els corresponents triangles, evidentment), centrat al vèrtex, i que en el sistema de coordenades d'ull tingui arestes de longitud igual al valor d'un `uniform float side;` paral·lels als eixos x i y en coordenades d'ull. A més heu d'assignar als vèrtexs les coordenades de textura correctes per a que el *fragment shader* pugui determinar el color de cada fragment a partir d'una de les imatges d'esferes que us proporcionem.

Per tal que es vegin esferes, cal que el fragment shader dibuixi sols els fragments de l'esfera. Per aquest motiu, tot fragment que un cop texturat rebi un valor d'alfa diferent d'1.0 ha de ser descartat.

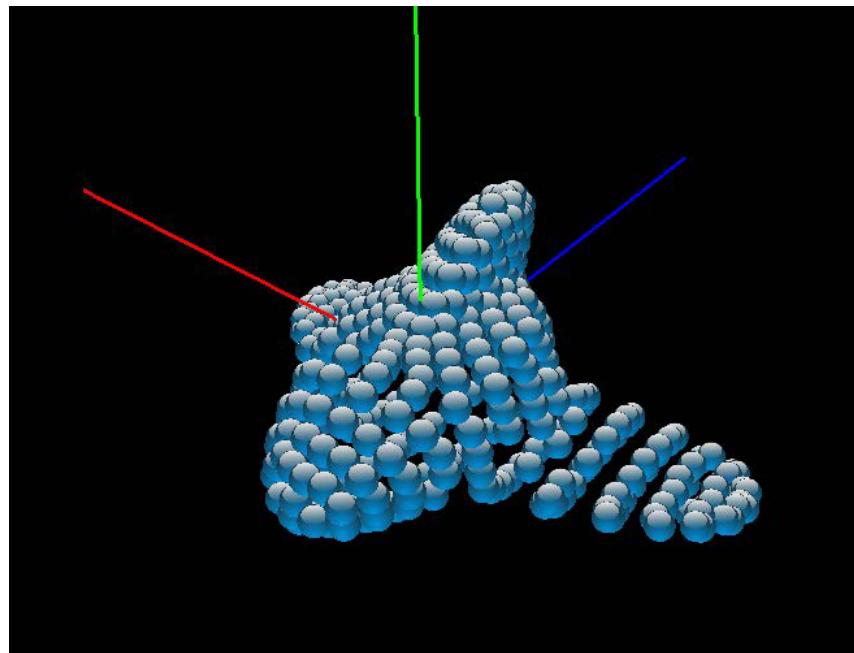


Figura 4: ‘boid’ amb `side=0.1`, usant la textura `sphere1.png`

13. area.vert, area.geom

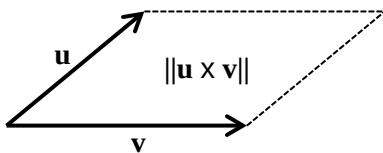
(2on control de laboratori, 2014-15 Q1)

Escrit un VS i un GS que colorin els triangles en funció de la seva àrea.

El VS escriurà **gl_Position** en un espai de coordenades apropriat.

El GS farà les següents tasques:

1. Calcularà **l'àrea del triangle en eye space**. Recordeu que el **mòdul del producte vectorial** de dos vectors **u** i **v** dóna l'àrea del paral·lelogram definit per **u** i **v**,

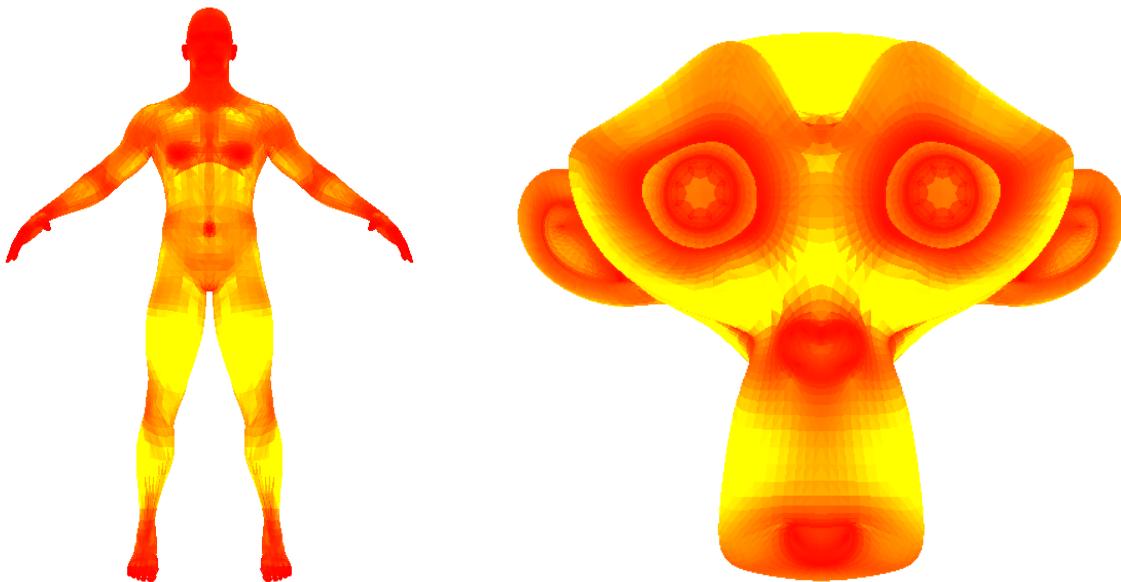


2. Normalitzarà aquest valor dividint per un **const float areamax = 0.0005**
3. El color dels vèrtexs es calcularà amb un gradient de color que vagi del vermell (àrea normalitzada nul·la) al groc (àrea normalitzada 1):



4. Emetrà els vèrtexs del triangle en clip space, com és habitual.

Aquí teniu el resultat esperat amb els models **man.obj** i **monkey1.obj** del ZIP de l'enunciat:



Identificadors (ús obligatori):

`area.vert, area.geom`