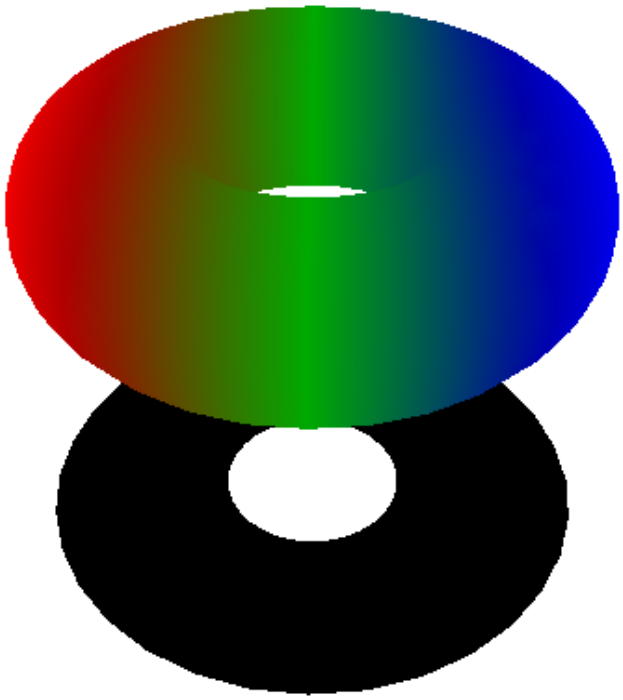
Shadow

Escriu un **vertex shader** i un **geometry shader** per ShaderMaker que simulin l'ombra que projecta l'objecte sobre el terra, que suposarem situat al pla **Y = -2.0,** 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 geometry shader haurà d'emetre dos triangles (en *clipping 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 = -2.0.

Aquí teniu el resultat que s'espera amb el torus. Observeu que no hi ha cap tipus d'il·luminació.



En aquest problema no cal cap variable varying definida per l'usuari. El geometry shader només ha d'escriure `gl_FrontColor` i `gl_Position`

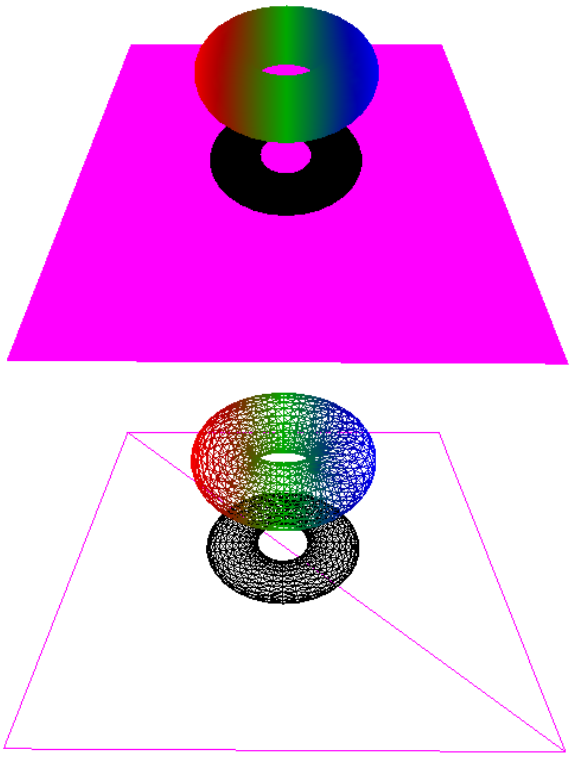
Shadow (versió 2) **(2on control laboratori, curs 2012-13, Q1)**

Escriu un **vertex shader** i un **geometry shader** per ShaderMaker que simulin l'ombra que projecta l'objecte sobre una taula, que suposarem situada al pla **Y = -2.0,** 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 geometry shader haurà d'emetre dos triangles (en *clipping space*): un corresponent al triangle original (amb el color tal qual, sense il·luminació), i un altre (de color negre) corresponent a la projecció del triangle al pla Y = -2.0.

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 magenta alineat amb els eixos de l'aplicació, amb costat 8 i centrat al punt (0, -2.1, 0).

Aquí teniu el resultat que s'espera amb el torus. Observeu que no hi ha cap tipus d'il·luminació.



En aquest problema no cal (ni es permet) cap variable varying definida per l'usuari. El geometry shader només ha d'escriure `gl_FrontColor` i `gl_Position`.

Explode (1)

Escriu un **vertex shader** i un **geometry shader** 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

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

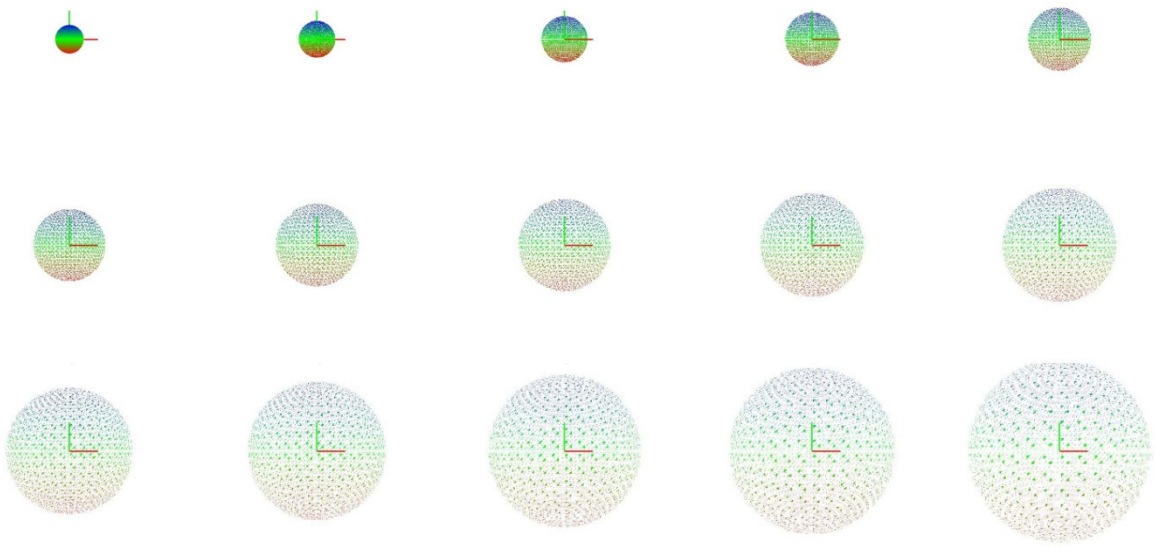
on speed és la velocitat desitjada (en unitats del model per segon), time és el temps (en segons), i **n** és el promig de les normals del tres vèrtexs del triangle (en *object space*).

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;
```

Aquí teniu el resultat amb l'esfera:



Explode (2)

Escriu un **vertex shader** i un **geometry shader** 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:

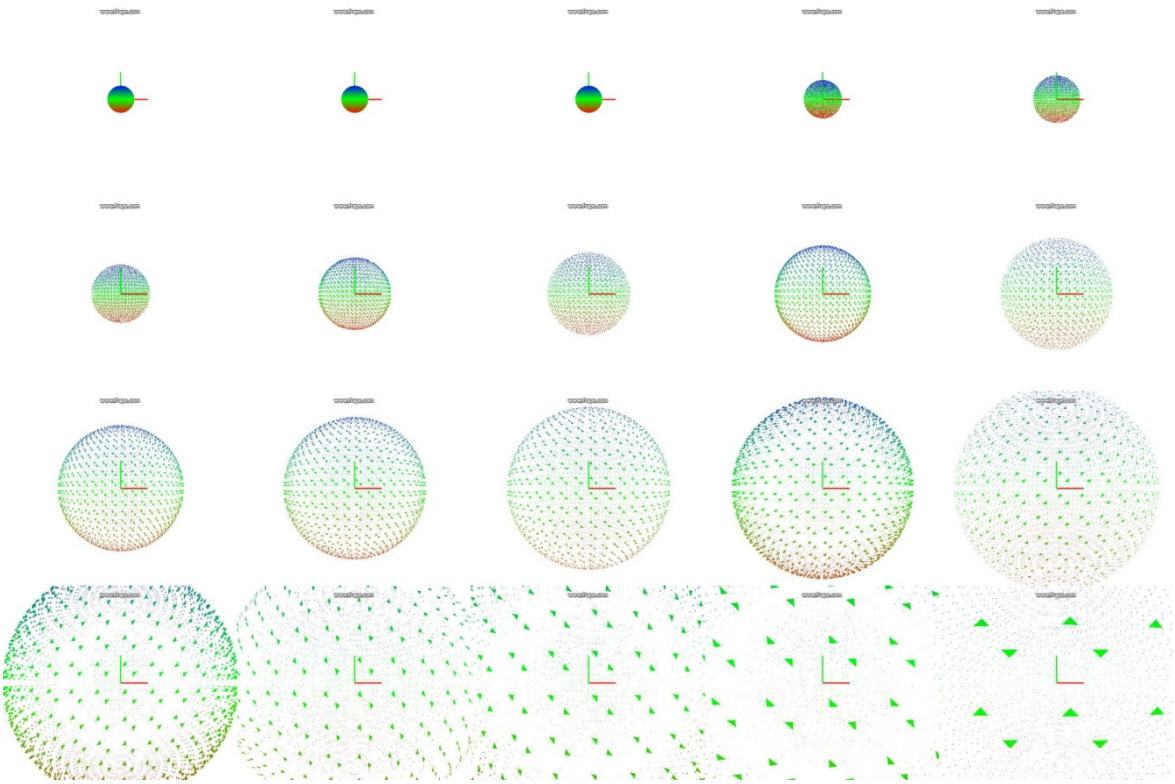
$$T(\text{speed}\cdot\text{time}\cdot\mathbf{n})\cdot R_z(\text{angSpeed}\cdot\text{time})$$

on speed és la velocitat desitjada (en unitats del model per segon), time és el temps (en segons), **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:

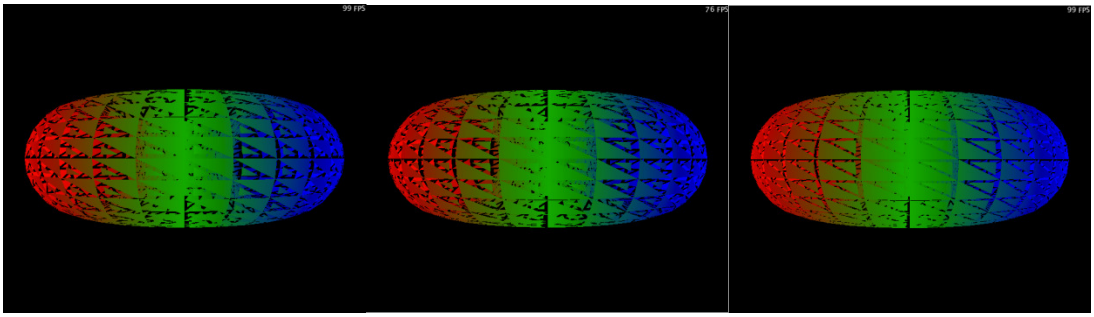
```
const float speed = 1.2;
const float angSpeed = 8.0;
```

Aquí teniu el resultat amb l'esfera:



Oscillating Shrink (2on control laboratori, curs 2012-13, Q2)

Escriu un **geometry shader** que copiï en sortida la meitat dels triangles, i encongeixi 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<=time<1.0/speed, es copiaran en sortida els que tinguin un identificador (gl_PrimitiveIDIn) parell, i entre 1.0/speed <= time < 2.0/speed, els que el tinguin senar, i així continuarà alternant cada 1.0/speed segons. Pels demés triangles, farem servir la part fraccionaria de time*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 time*speed és enter, i positiu altrament, atenyent un màxim (1) quan time*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

A la figura podeu veure el torus vist amb la càmera per defecte del ShaderMaker, amb diferents valors de speed i time.

Progressive (2on control laboratori, curs 2012-13, Q1)

Escriu un **vertex shader** i un **geometry shader** per ShaderMaker que vagin dibuixant els triangles de l'objecte a mesura que passi el temps, amb una velocitat de 100 primitives per segon.

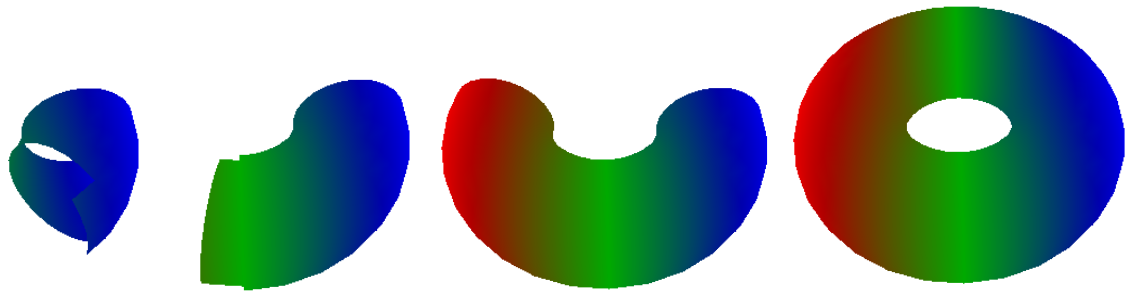
El VS simplement haurà d'escriure gl_Position (amb la posició en *object space*) i gl_FrontColor (amb el color que li arriba pel vèrtex).

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 geometry shader 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 el torus, després de 2, 4, 8 i 16 segons (fins a 200, 400, 800 i 1600 triangles, respectivament).



En aquest problema no cal (ni es permet) cap variable varying definida per l'usuari. El geometry shader només ha d'escriure `gl_FrontColor` i `gl_Position`

Voxelize (1)

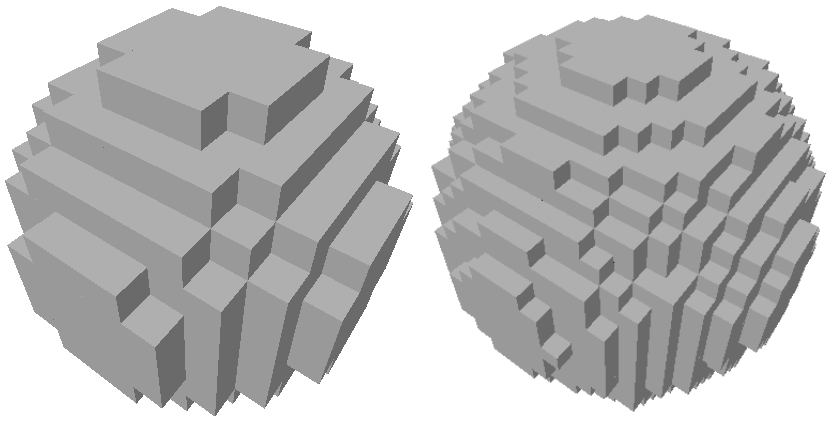
Escriu un **vertex shader** i un **geometry shader** per dibuixar una aproximació de la voxelització del model.

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 step·(i,j,k), amb i, j, k enters.

Aquí teniu els resultats amb l'esfera, per diferents valors de step (0.2 i 0.1):

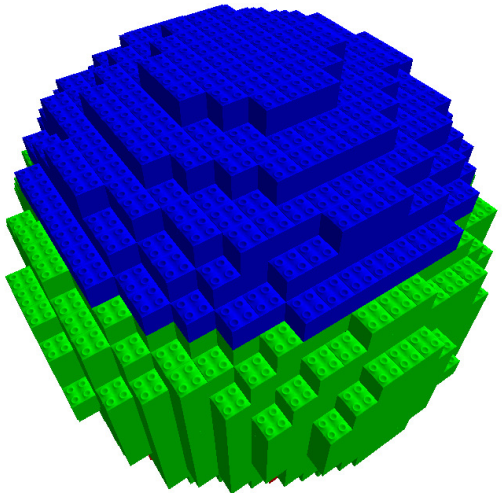


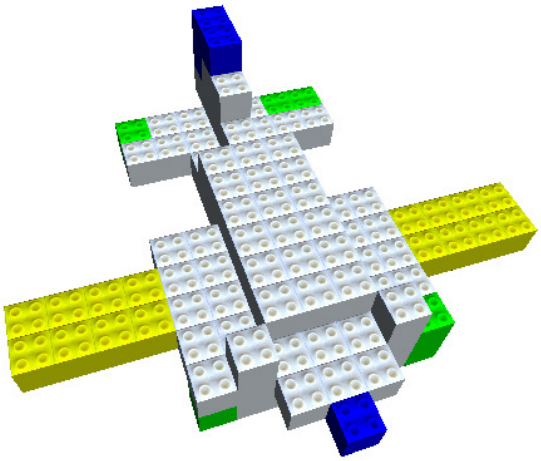
Lego

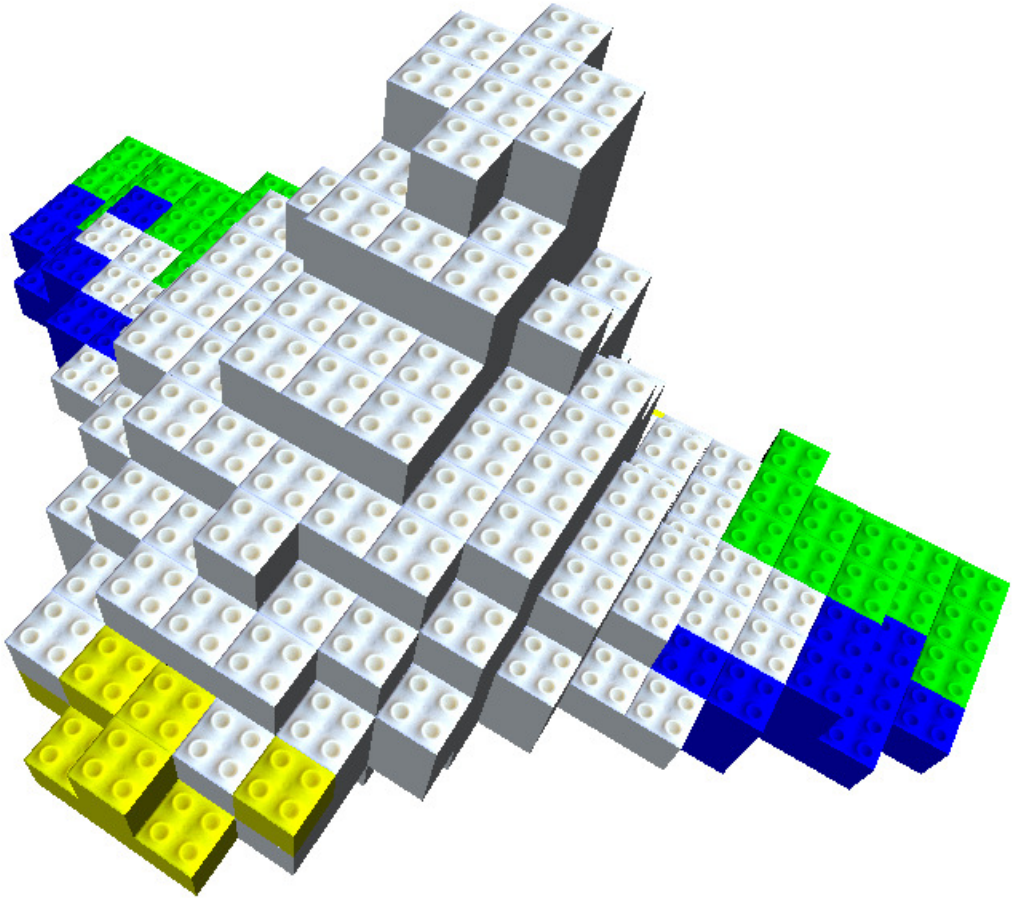
Escriu un **vertex shader** i un **geometry shader** per dibuixar una aproximació del model amb fitxes de Lego, tal i com indica la figura.

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,Y,W) de les peces de Lego.

Aquí teniu els resultats amb l'esfera, el cessna i el boid.obj:

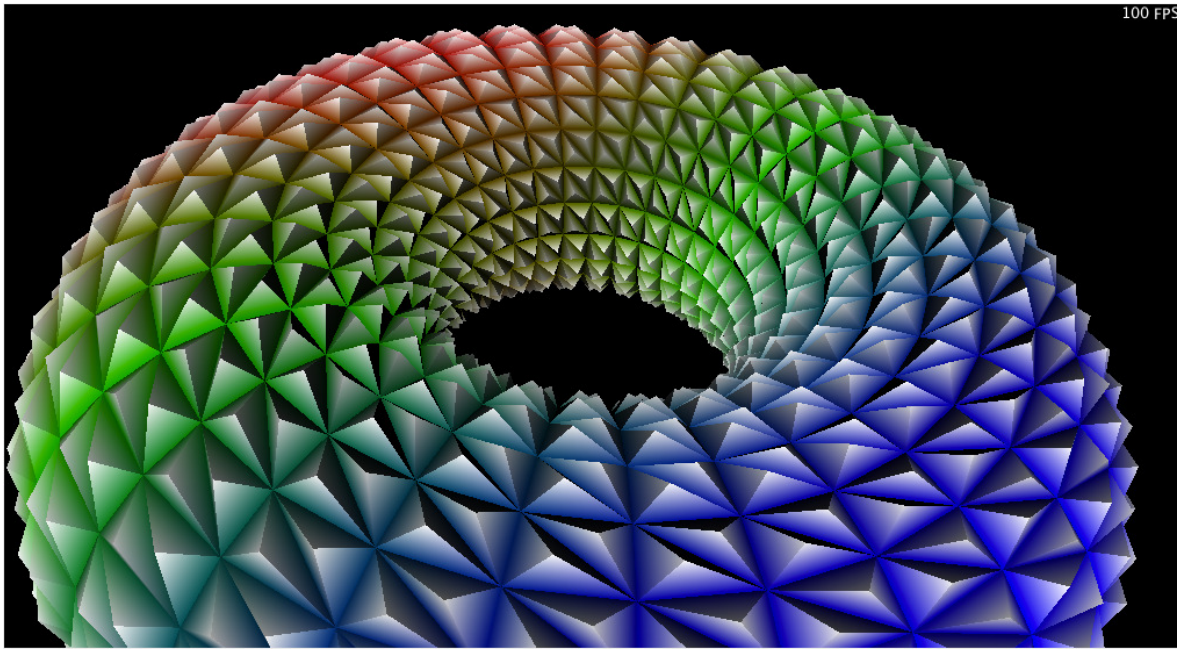






Spikes **(2on control laboratori, curs 2012-13, Q2)**

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. El **geometry shader** fa els seus càlculs en coordenades d'ull, i no rep cap més dada que el propi T (amb els atributs estàndards associats), i el uniform citat més amunt.



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