Hinomaru (hinomaru.frag) 2.5 punts

Write a *fragment shader* that generates a procedural texture with Japan's flag. You may assume that we will test your shader with the "Plane" object only. "Plane" has texture coordinates ranging from (0,0) to (1,1). All points and vectors in this problem refer to **texture space coordinates.**

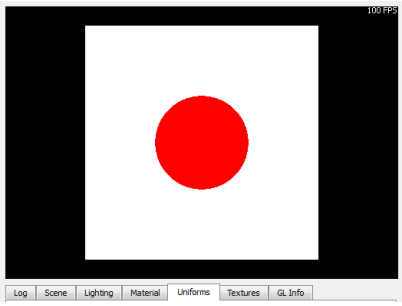

Figure 1: Expected result.

Notice that the background is white and the central circle is red. The texture coordinates of the center are C=(0.5, 0.5). The radius of the circle is 0.2, and it is centered at C. That is to say: if the fragment's (s,t) texture coordinates (gl_TexCoord[0].st) are closer than 0.2 to the point (0.5, 0.5), the fragment will be red. Otherwise, it will be white.

Identifiers (mandatory)

```
hinomaru.frag
```

Clipping plane (retall.vert, retall.frag) 2.5 punts

Write a *vertex shader* and a *fragment shader* that together display only that part of the model that lies on the negative halfspace of a plane defined by the user.

The plane is defined in **model space**, with a normal vector in the direction of the vector that goes from the origin to GL_LIGHT0, and the independent term given by a user-supplied *uniform*.

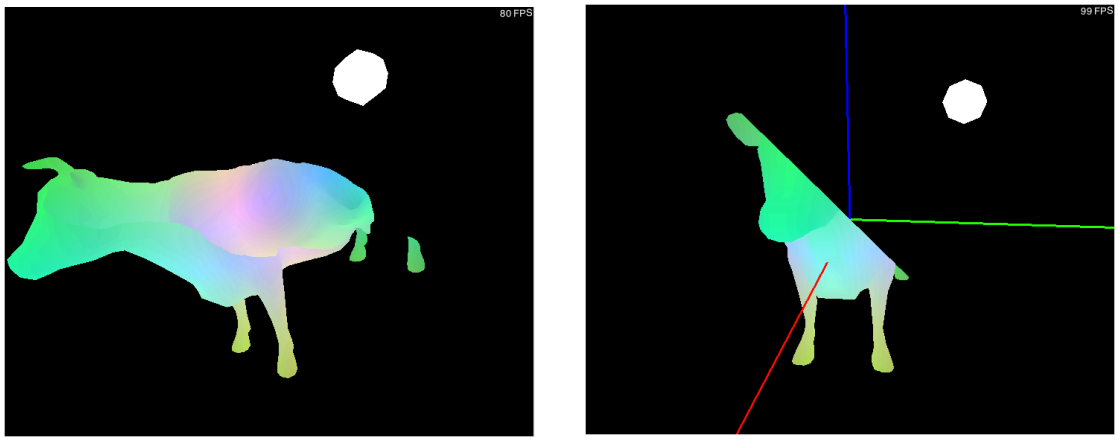

Figura 2: "cow" with offset=0.06 and the light at (-0.25, 0.5, 0.5). The figure on the right has the light in that same position, but offset set to zero, and the clipping plane containing the camera position.

You may test it activating the light animation and with, for instance, offset=0, so that the clipping plane always goes through the origin, and the orientation keeps changing.

Identifiers (mandatory)

```
retall.vert
retall.frag
uniform float offset;
```

Parallax hallucination (hallu.frag) 2.5 punts

Solving *Animate Texture* you experimented on the effect of modifying the texture coordinates (s, t). There we perturbed the coordinates with a time dependent **vec2 offset**, so that the coordinates became **(s, t)+ offset.** Here we ask for something similar, but changing the way in which **offset** is defined.

Let (s, t) be the texture coordinates received by the shader (gl_TexCoord[0].st). The fragment shader will start by retrieving the color **c** of the texture at that point (s, t). Now let **m** be the largest component of **c**'s rgb color. This **m** (in [0, 1]) is an approximation of the texel's color brightness. We will then define a **vec2 u** with components (**m, m**). Notice that this vectors length will be larger for bright colors and shorter for dark colors. Next rotate **u** by $\theta=2\pi t$ radians, were t measures time in seconds. Remember that a 2D rotation matrix is:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Now we are ready to compute the offset vector as **(a/100.0)u**, were **a** is a **uniform float**, and **u** is the rotated vector we just computed. The final color of the fragment is then retrieved from the texture (**uniform sampler2D map**) at **(s,t)+offset**.

The expected result (which is a bit subtle) for "Plane" and **a=0.5** can be seen in **hallu-gray-stones.mp4** and **hallu-color-stones.mp4**, using the textures with those same (Figure 3). Notice how lighter colors describe a larger circumference than the darker ones, creating a parallax effect that our visual system interprets as different depths.

Figura 3: Textures hallu-gray-stones.png and hallu-color-stones.png

Identifiers (mandatory)

```
hallu.frag
uniform sampler2D map;
uniform float time;
uniform float a;
```

Tangential spin (rot.vert, rot.frag) 2.5 punts

Write a *vertex shader* and a *fragment shader* that jointly display the objects with texture, but distorted by moving their vertices in the surface's tangent space. A parameter **d** will be taken to be the radius of a circumference around each vertex, contained in that vertex's tangent plane. The vertex will move at constant speed along that circumference. To achieve this, we will use two standard vertex attributes defined by ShaderMaker, declaring them like this:

```
attribute vec3 attrTangent;
attribute vec3 attrBitangent;
```

Together with the vertex normal, these define a standard basis for the tangent space, which you can see by checking ShaderMaker's option "show tangent space" (in "Scene").

Each vertex should move along its associated circumference at constant speed and frequency `0.1*time`, where `time` is the `uniform float` defined by ShaderMaker (in seconds from last program compilation). At `time==0` (ignoring phase, see below) we should use the position pointed by the tangent vector (`attrTangent`) from the vertex. However, each vertex will move (with respect to this basic movement) with a phase equal to $2\pi(s+t)$ where s, t are the texture coordinates. We provide two movies showing the expected effect on a textured torus, with d=0.2 and d=0.002. Notice that the model breaks down during the deformation, because the tangent spaces defined by Shadermaker on the same vertex differ by some amount on the different polygons using the vertex.

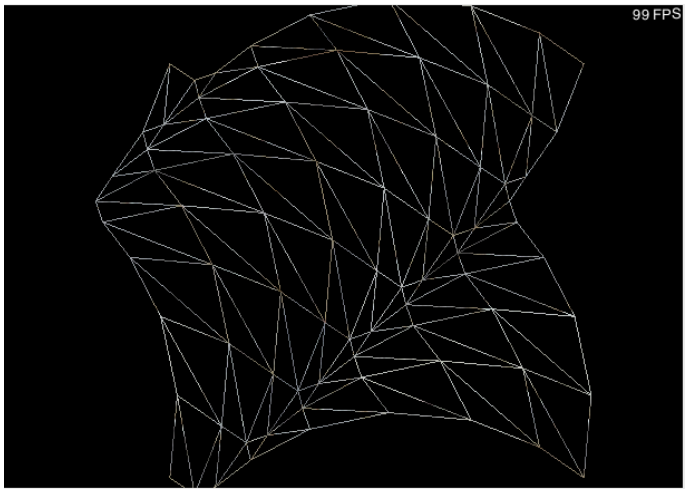

Figura 4: rot.* deforming "Plane", with d=0.2 and time=1s

Hint 1: The matrix whose rows are the unit tangent, bitangent and normal vectors changes vectors in tangent space to the basis in which the tangent basis is, and that the transpose of that matrix is its inverse.

Hint 2: You don't need to use a rotation matrix. Remember the trigonometric circle…

Identifiers (mandatory)

```
rot.vert
rot.frag
uniform float d;
attribute vec3 attrTangent;
attribute vec3 attrBitangent;
uniform sampler2D ex;
```