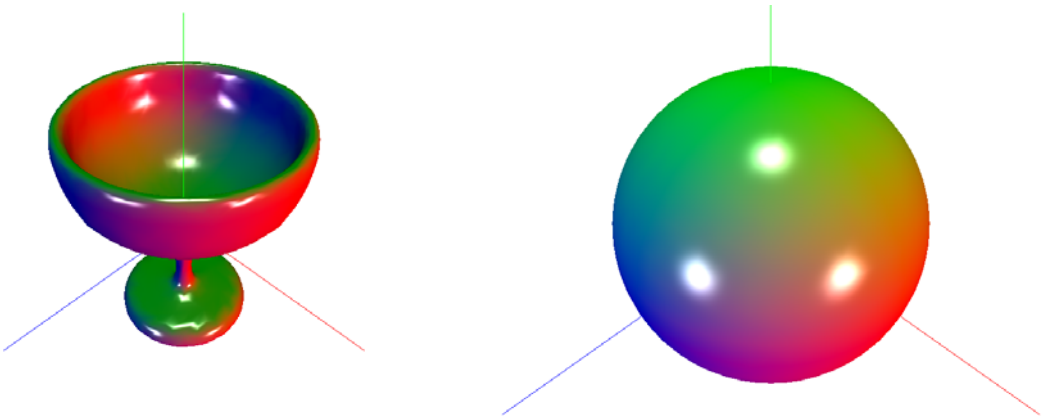
3lights (3lights.*)

Write **VS+FS** to compute per-fragment Phong lighting with 3 directional lights:



The VS must carry out typical tasks, and must provide the FS with the necessary data for lighting.

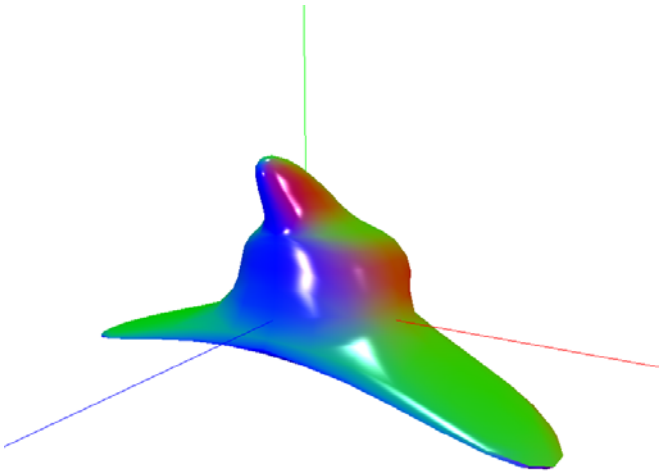
The FS must compute per-fragment color by accummulating the contribution of three directional lights (therefore ignore *lightPosition*):

- A light inifinitely far way along the X+ (model space), with *lightDiffuse* **red**.
- A light inifinitely far way along the Y+ (model space), with *lightDiffuse* **green**.
- A light inifinitely far way along the Z+ (model space), with *lightDiffuse* **blue**.

When computing the contribution of each light, ignore *matAmbient* and *lightAmbient*, and use the formula:

$$K_d I_d (N \cdot L) + K_s I_s (R \cdot V)^S$$

where I_d (*lightDiffuse*) is different for each light. For the rest of parameters (K_d, K_s, I_s, s) use the usual uniforms.

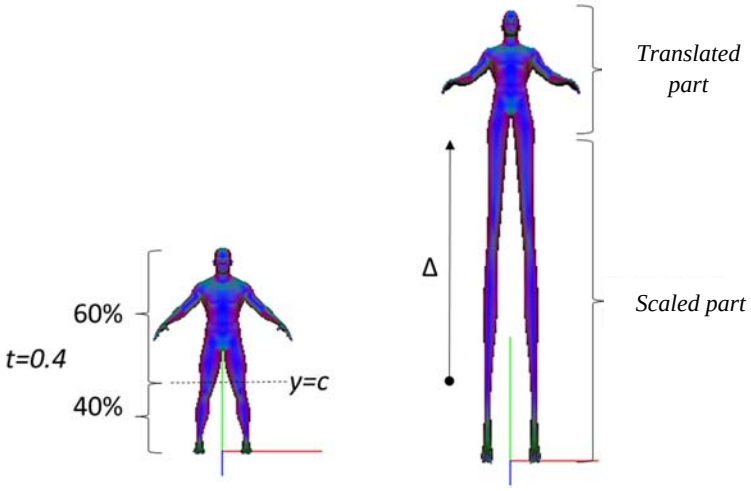


Dalify (dalify.*)

Write **VS+FS** to deform the 3D model in the vertical direction (Y *model space* axis), to achieve an appearance similar to that of some animals in Salvador Dalí pictures:

Les temptacions de Sant Antoni (Salvador Dalí, 1946)

The VS should deform the model modifying only the Y *model space* coordinate:

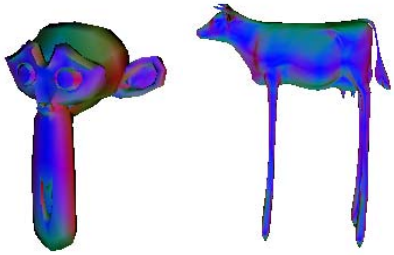


Let c be the result of linearly interpolating boundingBoxMin.y and boundingBoxMax.y, according to an interpolation paremeter **uniform float t = 0.4**.

If Y<c, the VS should apply a scale along Y axis given by **uniform float scale = 4.0** to make the model legs longer. Otherwise, the VS will not apply any scale, but a translation Δ to Y. To compute Δ, note that continuity at y=c involves that **c*scale = c + Δ** (thus isolate Δ).

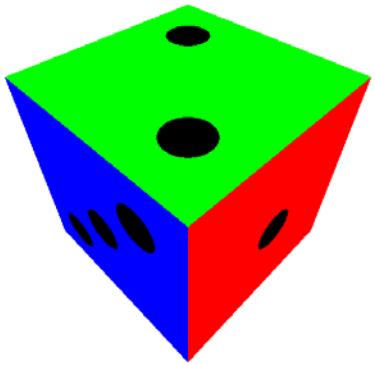
Since we are not updating the clipping planes, the model might appear clipped.

The FS should carry out the default tasks.



Dice (dice.*)

Write **VS+FS** to texture procedurally **cube.obj** so that it appears as a dice with one, two or three dots:



VS: usual tasks, providing the FS with necessary data. No lighting is required on frontColor.

FS: should not use any texture image, but compute the final fragment color depending on its XYZ coordinates and its normal, that it should get from the VS in *object space*.

The number of dots per face should depend on the *object space* normal vector:

- Faces with normal vector parallel to **X** axis, should show one dot.
- Faces with normal vector parallel to **Y** axis, should show two dots.
- Faces with normal vector parallel to **Z** axis, should show three dots.

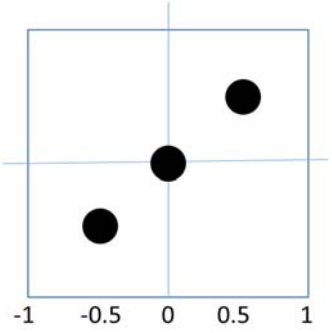
Use a robust test to determine which face are you treating (avoid equality comparisons between floats).

You should not use texture coordinates, but the XYZ point coordinates that the VS will provide to FS in *object space*. For cube.obj, coordinates are in [-1, 1].

Example for a face with normal parallel to X axis:

- Let Q be the 2D point defined by YZ coords of the point (ignore X coordinate).
- If the distance from Q to the central point of the face (0, 0) is smaller than 0.2, the fragment color should be black (the fragment corresponds to a dot); otherwise the color should be the original color, without lighting.

The treatment for the other faces is similar, taking XZ or XY to define point Q, and computing the fragment color as black if the distance from Q to any involved center (0,0), (0.5, 0.5), (-0.5, -0.5) is below the radius 0.2:



Magic window (magic.*) 2.5 punts

Write **VS**+**FS** to simulate a window across which we could see the room interior and again the outside. In /assig/grau-g/Textures you will find the textures **window.png**, **interior.png** i **exterior.png**:



VS: besides usual tasks, it should pass to VS the **normal N in eye space.**

FS: should access to texture **window.png** (with usual texture coordinates) to get a color that will be referred to as **C**.

If the alpha component of C is 1.0 (opaque part of the first window), the fragment color should be C. Otherwise, to compute the fragment color you should access to texture **interior.png** (with texture coordinates **vtexCoord+0.5*N.xy**) to get a second color that will be referred to **D**.

If the alpha component of D is 1.0 (opaque part of inner window), the fragment color should be D. Otherwise, the fragment color should be that of texture **exterior.png** at point **vtexCoord+0.7*N.xy**.

Notice that we are using an offset in texture coordinates which depends on the normal components in eye space, so that the visible part of the inner room and outside will depend on the plane orientation.

Expected result (**plane.obj**), from different view points:



Identifiers (compulsory):

```
magic.vert, magic.frag
uniform sampler2D window;
uniform sampler2D interior1;  // notice digit 1
uniform sampler2D exterior2;  // notice digit 2
```