Warhol (warhol.*) Useu: `~/assig/grau-g/Viewer/GLarenaSL`

Escriu **VS+FS** per tal de simular varicions en la saturació (S) i la tonalitat (H, hue) del color, imitant una mica una de les obres més conegudes de l'artista Andy Warhol:

Serie Marilyn Monroe

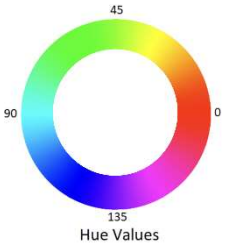
Textura

Resultat esperat

El **VS** farà les tasques imprescindibles. El **FS** serà l'encarregat de triar el color del fragment, d'acord amb les coordenades de textura que rebrà del VS, i que per l'**objecte plane** estan dins [0,1].

Com podeu veure a la figura, la textura estarà repetida quatre cops, però cada quadrat té una variació diferent en la tonalitat. Podeu utilitzar les següents funcions, les quals permeten fer conversions entre els espais de color RGB i HSV (també al fitxer **rgb2hsv.frag**, que podeu copiar al vostre warhol.frag):

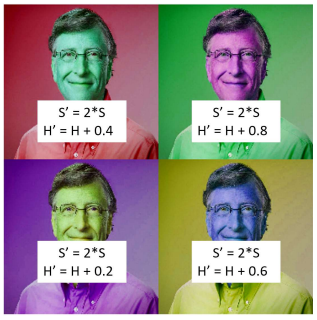
```
//http://gamedev.stackexchange.com/questions/59797/glsl-shader-change-hue-saturation-brightness
vec3 rgb2hsv(vec3 c) {
    vec4 K = vec4(0.0, -1.0 / 3.0, 2.0 / 3.0, -1.0);
    vec4 p = mix(vec4(c.bg, K.wz), vec4(c.gb, K.xy), step(c.b, c.g));
    vec4 q = mix(vec4(p.xyw, c.r), vec4(c.r, p.yzx), step(p.x, c.r));
    float d = q.x - min(q.w, q.y); float e = 1.0e-10;
    return vec3(abs(q.z + (q.w - q.y) / (6.0 * d + e)), d / (q.x + e), q.x);
}

vec3 hsv2rgb(vec3 c) {
    vec4 K = vec4(1.0, 2.0 / 3.0, 1.0 / 3.0, 3.0);
    vec3 p = abs(fract(c.xxx + K.xyz) * 6.0 - K.www);
    return c.z * mix(K.xxx, clamp(p - K.xxx, 0.0, 1.0), c.y);
}
```



Recordeu que, en espai HSV:

- H (hue) representa la tonalitat, habitualment com un valor de 0 a 360º, en el nostre cas en [0,1].
- S (saturation) representa la puresa del color; com menor sigui la saturació, major tonalitat grisenca. També normalitzat en [0,1].
- V (value) representa la brillantor del color. També normalitzat en [0,1].

Aquesta figura mostra quina ha ser la variació en H i S respecte el color original del texel de la textura:



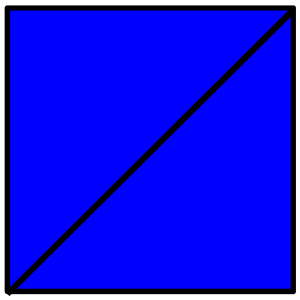
Observa que cal doblar la saturació, i que a la tonalitat H se li afegeix un valor en {0.2, 0.4, 0.6, 0.8}.

Identificadors obligatoris:

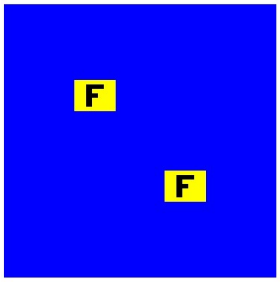
`warhol.vert, warhol.frag` (Has escrit **warhol** correctament? En minúscules?)
`uniform sampler2D colormap;`

Labeler (labeler.*) Useu: `~/assig/grau-g/Viewer/GLarenaSL`

Escriu **VS+GS+FS** per tal de dibuixar un petit rectangle amb la lletra "F" al centre de cada triangle:



Triangles de l'objecte plane.obj



Resultat esperat

El **VS** farà les tasques imprescindibles.

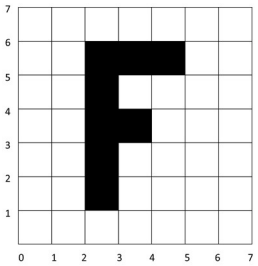
El **GS** farà les tasques per defecte (dibuixar el triangle original) i a més a més serà l'encarregat de dibuixar el rectangle amb la lletra F. Aquest rectangle estarà centrat al centre del triangle.

Usarem aquests dos uniforms:

```
uniform float size = 0.07;
uniform float depth = -0.01;
```

La mida del rectangle serà de 2*size, on size està en NDC (Normalized Device Coordinates, amb valors en [-1, 1] per punts dins la piràmide de visió de la càmera). També introduirem un petit offset a la z per tal d'evitar que el triangle oculti part del rectangle. Per tant, si C és el centre del triangle en NDC, els vèrtexs del rectangle que cal emetre al GS tindran coordenades de la forma (C.x ± size, C.y ± size, C.z + depth, 1.0). Observa que hem afegit una w=1.0 perquè la sortida del GS cal que sigui en clip space.

El **FS** assignarà al fragment el color que li arribi del GS, tret de quan el fragment pertany al rectangle afegit. En aquest cas, cal texturar el fragment de forma procedural, a partir de les coordenades de textura que rebrà del GS. Assumint que teniu coordenades de textura en [0,7], aquí teniu els fragments que haran de ser negres (la resta seran de color groc):



Identificadors obligatoris:

`labeler.vert, labeler.geom, labeler.frag` (segur que has escrit **labeler** correctament?)
Tots els uniforms de l'enunciat.

RGBDepth (RGBDepth.*)

Useu `GLarenaPL` de la vostra instal·lació local del visualitzador

Escriu un **plugin** que, utilitzant dues passades de rendering, permeti visualitzar, de manera independent, els tres canals de color del framebuffer i la profunditat de l'escena.

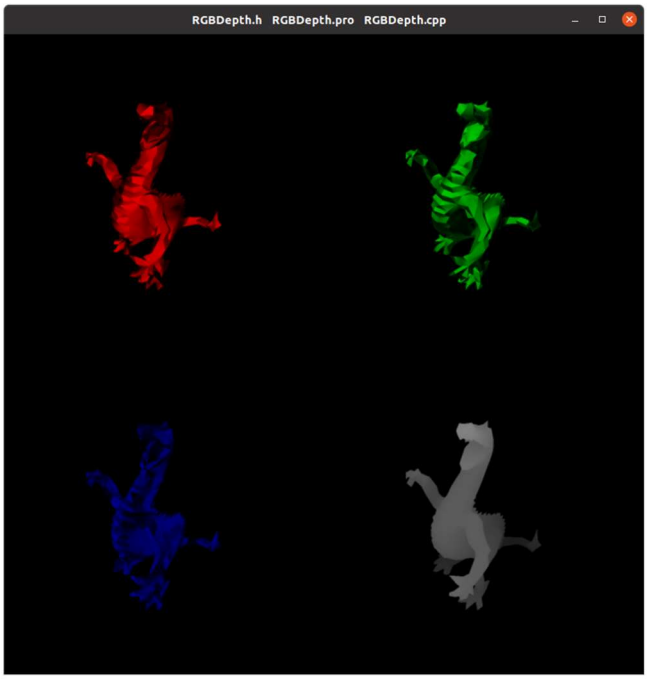
En la primera passada, caldrà pintar l'escena de manera que per a cada fragment es guardi el color en les components *rgb* i la profunditat en la component *a*. Caldrà que els fragments no pintats tinguin color negre. Us proporcionem uns shaders (RGBDepth.*) que fan aquesta feina; el vostre codi els haurà d'utilitzar. Fixeu-vos que cal que proporcioneu les dades necessàries als uniforms d'aquests shaders.

Al finalitzar aquesta primera passada, caldrà que guardeu el framebuffer en una textura.

En la segona passada, caldrà que pinteu 4 rectangles diferents (amb una disposició de matriu 2x2) dins del viewport. A cada rectangle es visualitzarà un d'aquests canals: superior-esquerra, vermell; superior-dreta, verd; inferior-esquerra blau; i inferior-dreta, profunditat. Us proporcionem uns shaders que realitzen aquesta tasca (SplitView.*); el vostre codi els haurà d'utilitzar. Fixeu-vos que cal que proporcioneu les dades necessàries als uniforms d'aquests shaders.

Cal que totes les tasques de configuració i pintat es realitzin en les funcions corresponents de la interfície dels plugins que permetin maximitzar el rendiment del vostre plugin.

A continuació teniu un exemple del resultat esperat:



Identificadors obligatoris:

`RGBDepth.cpp, RGBDepth.h, RGBDepth.pro` (segur que has escrit **RGBDepth** correctament?)
Tots els uniforms dels shaders RGBDepth.vert, RGBDepth.frag, SplitView.vert i SplitView.frag.