Gràfics
Curs 2022/23 Q1

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

Escribe VS+FS para simular variaciones en la saturación (S) y la tonalidad (H, hue) del color, imitando un poco una de las obras más conocidas del artista Andy Warhol:


Serie Marilyn Monroe


Textura


Resultat esperat

El VS realizará las tareas imprescindibles. El FS será el encargado de elegir el color del fragmento, de acuerdo con las coordenadas de textura que recibirá del VS, y que para el objeto plane están dentro [0,1]. Como puede ver en la figura, la textura estará repetida cuatro veces, pero cada cuadrado tiene una variación diferente en la tonalidad. Puedes utilizar las siguientes funciones, que permiten realizar conversiones entre los espacios de color RGB y HSV (también en el archivo rgb2hsv.frag, que puedes copiar en tu 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);
}
```

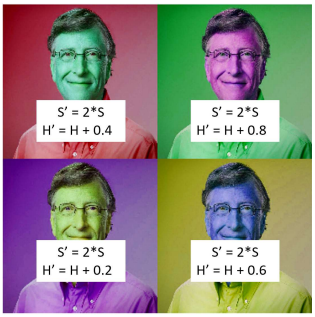
Recuerda que, en espacio HSV:

• H (hue) representa la tonalidad, habitualmente como un valor de 0 a 360º, en nuestro caso en [0,1].

• S (saturación) representa la pureza del color; cuanto menor sea la saturación, mayor tonalidad grisácea. También normalizado en [0,1].

• V (value) representa el brillo del color. También normalizado en [0,1].

Esta figura muestra cuál debe ser la variación en H y S respecto al color original del texel de la textura:



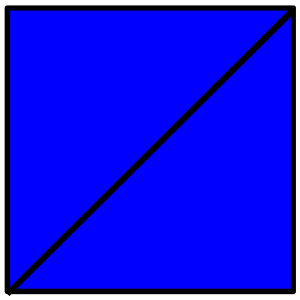
Observa que se dobla la saturación, y que a la tonalidad H se le añade un valor en {0.2, 0.4, 0.6, 0.8}.

Identificadores obligatorios:

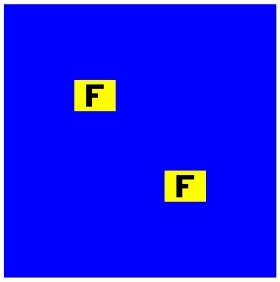
`warhol.vert, warhol.frag` (¿Has escrito **warhol** correctamente? ¿En minúsculas?)
`uniform sampler2D colormap;`

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

Escribe VS+GS+FS para dibujar un pequeño rectángulo con la letra “F” en el centro de cada triángulo:



Triángulos de plane.obj



Resultado esperado

El VS realizará las tareas imprescindibles.

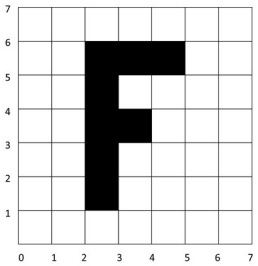
El GS realizará las tareas por defecto (dibujar el triángulo original) y además será el encargado de dibujar el rectángulo con la letra F. Este rectángulo estará centrado en el centro del triángulo.

Usaremos estos dos uniforms:

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

El tamaño del rectángulo será de 2*size, donde size está en NDC (Normalized Device Coordinates, con valores en [-1, 1] para puntos dentro de la pirámide de visión de la cámara). También introduciremos un pequeño offset en la z para evitar que el triángulo oculte parte del rectángulo. Por tanto, si C es el centro del triángulo en NDC, los vértices del rectángulo a emitir en el GS tendrán coordenadas de la forma (C.x ± size, C.y ± size, C.z + depth, 1.0). Observa que hemos añadido una w=1.0 para que la salida del GS esté en clip space.

El FS asignará al fragmento el color que le llegue del GS, salvo cuando el fragmento pertenezca al rectángulo añadido. En este caso, es necesario texturar el fragmento de forma procedural, a partir de las coordenadas de textura que recibirá del GS. Asumiendo que tiene coordenadas de textura en [0,7], aquí tienes los fragmentos que tendrán que ser negros (el resto serán de color amarillo):



Identificadores obligatorios:

`labeler.vert, labeler.geom, labeler.frag` (¿has escrito **labeler** correctamente?)
Todos los uniforms del enunciado.

RGBDepth (RGBDepth.*)

Usa `GLarenaPL` de tu instalación local del visualitzador

Escribe un **plugin** que, utilizando dos pasadas de rendering, permita visualizar, de forma independiente, los tres canales de color del framebuffer y la profundidad de la escena.

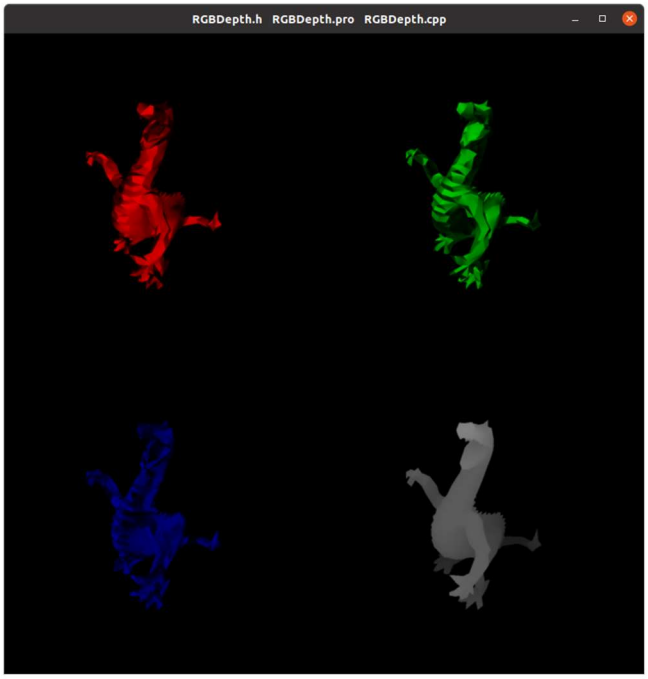
En la primera pasada, habrá que pintar la escena de forma que para cada fragmento se guarde el color en las componentes rgb y la profundidad en la componente a. Es necesario que los pixeles no pintados tengan color negro. Te proporcionamos unos shaders (RGBDepth.*) que realizan este trabajo; tu código deberá utilizarlos. Fíjate que debes proporcionar los datos necesarios a los uniforms de estos shaders.

Al finalizar esta primera pasada, deberás guardar el framebuffer en una textura.

En la segunda pasada, deberás pintar 4 rectángulos diferentes (con una disposición de matriz 2x2) dentro del viewport. En cada rectángulo se visualizará uno de estos canales: superior-izquierda, rojo; superior-derecha, verde; inferior-izquierda azul; e inferior-derecha, profundidad. Te proporcionamos unos shaders que realizan esta tarea (SplitView.*); tu código deberá utilizarlos. Fíjate que debes proporcionar los datos necesarios a los uniforms de estos shaders.

Es necesario que todas las tareas de configuración y pintado se realicen en las funciones correspondientes de la interfaz de los plugins que permitan maximizar el rendimiento del plugin.

A continuación tienes un ejemplo del resultado esperado:



Identificadores obligatorios:

RGBDepth.cpp, RGBDepth.h, RGBDepth.pro (¿seguro que has escrito RGBDepth correctamente?)

Todos los uniforms de los shaders RGBDepth.vert, RGBDepth.frag, SplitView.vert y SplitView.frag.