Tema 7: Proceso de Imágenes

J. Ribelles

SIE020: Síntesis de Imagen y Animación
Institute of New Imaging Technologies, Universitat Jaume I

Contenido

Introducción

Introducción

Desde sus orígenes, OpenGL ha tenido en cuenta en el diseño de su *pipeline* la posibilidad de manipular imágenes sin asociarle geometría alguna. Sin embargo, no es hasta que se produce la aparición de los procesadores gráficos programables cuando de verdad OpenGL se puede utilizar como una herramienta para procesado de imágenes.

Efectos de Imagen

Características

- El procesador de fragmentos recibe un valor de color de cuatro componentes (RGBA) que podemos modificar como más nos interese.
- En el caso de que nuestro efecto involucre utilizar sólo una imagen, esta puede ser enviada al procesador gráfico utilizando la orden *glDrawPixels*.
- En el caso de necesitar conocer los valores de color de otros píxeles de la imagen, entonces hay que proporcionar la imagen como textura.
- También, en el caso de utilizar más de una imagen, el resto de imágenes tendrán que ser proporcionadas como texturas.

Brillo

Descripción

Sólo hay que multiplicar el color de cada fragmento por un valor *alpha*. Si dicho valor es 1, la imagen no se altera, si es mayor que uno se aumentará el brillo, y si es menor que uno se disminuirá.

```
uniform float alfa;

in  vec4 color;
out vec4 colorFragmento;

void main()
{
  colorFragmento = color * alfa;
}
```

Negativo

Descripción

Únicamente hay que asignar como color final del fragmento el resultado de restarle a uno su valor de color original.

```
uniform float alfa;

in  vec4 color;
out vec4 colorFragmento;

void main()
{
  colorFragmento = 1.0 - color;
}
```

Escala de grises

Descripción

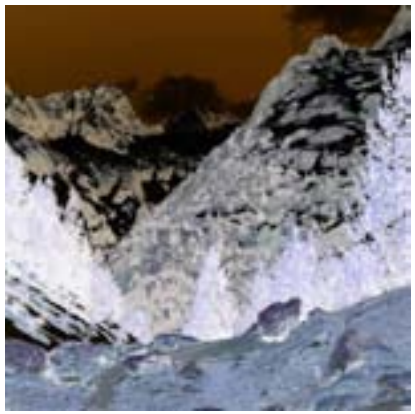
Únicamente hay que asignar como color final del fragmento el resultado de la media de sus tres componentes.

```
uniform float alfa;

in  vec4 color;
out vec4 colorFragmento;

void main()
{
  float media     = (color[0] + color[1] + color[2]) / 3.0;
  colorFragmento = vec4 (media, media, media, 1.0);
}
```

Ejemplos

(a) Original

(b) Brillo



(c) Negativo

(d) Escala de grises

Mezcla de imágenes

Descripción

Supone acceder en el *fragment shader* a un píxel de cada una de las imágenes para calcular el color final.

```
uniform float alfa;

in  vec4 color;
out vec4 colorFragmento;

void main()
{
  vec4 colorA = texture(imagenA, coordenadaTextura.st);
  vec4 colorB = texture(imagenB, coordenadaTextura.st);

  colorFragmento = mix (colorA, colorB, alfa) ;
}
```

Ejemplos

(e) alfa = 0'33

(f) alfa = 0'66

Convolución

Descripción

- Es una operación mateḿatica fundamental en procesamiento de imágenes.
- Consiste en calcular para cada píxel la suma de productos entre la imagen fuente y una matriz mucho más pequeña a la que se le denomina filtro de convolución.
- Lo que la operación de convolución realice depende de los valores de dicho filtro.

$$Res(x,y) = \sum_{j=0}^{n-1}\sum_{i=0}^{m-1} Img(x+(i-\frac{m-1}{2}), y+(j-\frac{n-1}{2}))\cdot Filtro(i,j) \quad (1)$$

Ejemplos

1	1	1
1	1	1
1	1	1

(a)

0	-1	0
-1	5	-1
0	-1	0

(b)

0	1	0
1	-4	1
0	1	0

(c)

Cuadro: Filtros de convolución: (a) suavizado, (b) nitidez y (c) detección de bordes.

Fragment Shader

```
const   int  tallaFiltro 9;

uniform vec2  desplazamiento[tallaFiltro];
uniform float filtro[tallaFiltro];

uniform sampler2D imagen;

in  vec2 coordenadaTextura;
out vec4 colorFragmento;

void main()
{
  int i;
  vec4 suma = vec4(0.0);

  for (i = 0; i < tallaFiltro; i++)
  {
    vec4 aux = textura(imagen,
                       coordenadaTextura.st + desplazamiento[i]);
    suma = suma + (aux * filtro[i]);
  }

  colorFragmento = suma;
}
```

Antialiasing

Descripción

- Aliasing: artefacto gráfico derivado de la conversión de entidades continuas a discretas.
- Antialiasing: técnicas destinadas a eliminar ese efecto escalera.
- Hoy en día, la potencia de los procesadores gráficos permite que desde el propio panel de control del controlador gráfico el usuario pueda solicitar la solución de este problema.

En OpenGL

Descripción

- Implementa la técnica conocida como sobremuestreo, *multisampling*, también conocido por *Full Scene Anti-Aliasing, FSAA*.
- Consiste en obtener más de un valor para cada píxel de la imagen final de manera que el color definitivo sea el resultado de una operación de mezcla realizada sobre el conjunto de todas las muestras obtenidas.
- Tanto la operación como la localización de las muestras dependen del fabricante del hardware.
- Cada fragmento se extiende para incluir, para cada muestra, información adicional de color, profundidad, etc. Toda esta información se almacena en un nuevo buffer llamado *multisample buffer*. Cada etapa del *pipeline* se realiza sobre cada muestra.

En cuanto al código

- Su creación se solicita a través de la FreeGLUT así: glutInitDisplayMode(... | GLUT_MULTISAMPLE).
- Después, el sobremuestreo se habilita con: glEnable(GL_MULTISAMPLE).
- Para realmente saber si se está haciendo o no el sobremuestreo, hay que comprobar que el resultado de la variable *buffers* tras la ejecución de la siguiente línea es uno: glGetIntegerv(GL_SAMPLES_BUFFERS, &buffers);
- Y para conocer cuántas muestras se está calculando por cada píxel hay que consultar el valor de la variable *muestras* después de ejecutar la siguiente orden: glGetIntegerv(GL_SAMPLES, &muestras);

Transformaciones

Pasos

- Leer la imagen y crear un objeto textura con ella.
- Definir un rectángulo sobre el que pegar la textura.
- Escribir un *vertex shader* que realice la transformación.

Warping

Define una malla de polígonos en lugar de un único rectángulo, y modifica los vértices de manera conveniente.

Recuperación y Almacenamiento

Dos pasos

- Llevarlo en primer lugar a memoria principal;
- y después a un fichero.

```
int      talla     = anchoVentana*altoVentana*3;
GLubyte *pixeles = (GLubyte *) calloc (sizeof(GLubyte), talla);

if (pixeles != NULL)
{
  // primer paso
  glPixelStorei (GL_PACK_ALIGNMENT, 1);
  glReadPixels  (0, 0, anchoVentana, altoVentana, GL_RGB, GL_UNSIGNES_BYTE, pixeles);

  // segundo paso
  FILE *fichero = fopen (filename, "wb");
  fwrite (pixeles, sizeof(GLubyte), talla, fichero);
  fclose (fichero);

  free   (pixeles);
}
```