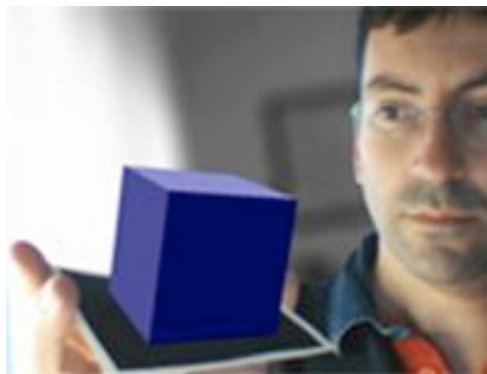


Gráficos por Computador

Imágenes y texto



Jordi Linares i Pellicer

Escola Politècnica Superior d'Alcoi

Dep. de Sistemes Informàtics i Computació

jlinares@dsic.upv.es

<http://www.dsic.upv.es/~jlinares>

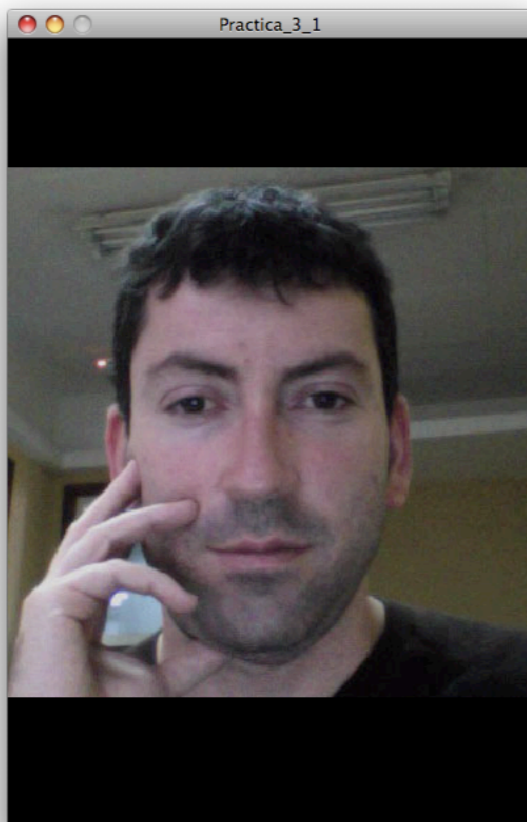
Imágenes

- *processing* dispone de la clase `PImage`, con la que se pueden crear objetos del tipo imagen
- La función `loadImage()` nos permite cargar una imagen en un objeto `PImage` en cualquiera de los formatos: GIF, JPG, TGA y PNG
- Los ficheros de las imágenes a cargar deben estar en el directorio `data` donde radica la aplicación (el PDE de *processing* tiene una opción 'Add File' que permite seleccionar esta imagen e insertarla en el directorio `data`)
- Para visualizar la imagen se puede usar la función `image()`:
 - `image(imagen, x, y)` -> Muestra la imagen a partir de las coordenadas `(x, y)` especificadas, sin ajuste de su tamaño
 - `image(imagen, x, y, ancho, alto)` -> Muestra la imagen a partir de la coordenadas `(x, y)` especificadas y reescalada al ancho y alto especificados
- Ejemplo:

```
PImage foto = loadImage("foto.jpg");  
image(foto, 0, 0);
```

Práctica 3-1

- Crear una nueva aplicación. Agregad una imagen en el directorio data
- Mostrad la imagen de forma que ésta se vea en su totalidad en la área de la ventana, manteniendo las proporciones originales y centrada (transformación isotrópica centrada)
- Para consultar el tamaño de la imagen se puede hacer también uso de los atributos `width` y `height`. Si `im` es del tipo `PImage`, sus dimensiones son `im.width` y `im.height`



Imágenes

`createImage(ancho, alto, modo de color)`

- Permite crear una nueva imagen
- El modo de color puede ser RGB, ARGB o ALPHA
 - RGB permite imágenes de 24 bits (8 por canal, 'true color')
 - ARGB añade un canal adicional para transparencia
 - ALPHA son imágenes de un único canal (para efectos de transparencias)

Imágenes

`get()`

`get(x, y)`

`get(x, y, width, height)`

- **Los métodos `get()` permiten obtener un pixel particular de la imagen, `get(x, y)`, o una nueva imagen como un fragmento de la imagen original, `get(x, y, ancho, alto)`. El método `get()` sin parámetros retorna una copia de la imagen sobre el que es aplicado.**

Imágenes

```
set(x, y, color)
```

```
set(x, y, imagen)
```

- Los métodos `set()` permiten cambiar el valor de un pixel de la imagen sobre la que se aplica, `set(x, y, color)`, o mapear una imagen a partir de las coordenadas `(x, y)` sobre la imagen sobre la que el método `set()` se está aplicando.

Imágenes

- Para facilitar el trabajo con colores, se puede utilizar el tipo `color` (que en realidad no es más que un entero), la función `color(r, g, b)` que nos permite crear un nuevo color a partir de los valores RGB correspondientes, y las funciones `red(color)`, `green(color)` y `blue(color)` que retornan el correspondiente canal a partir de un color RGB

Imágenes

- Ejemplo:

```
// Hay que asegurarse que foto.jpg está en el directorio 'data'
PImage foto_original = loadImage("foto.jpg");

PImage foto_modificada = createImage(foto_original.width,
                                     foto_original.height, RGB);

// 50% más de intensidad
for (int i = 0; i < foto_original.width; i++)
  for (int j = 0; j < foto_original.height; j++) {
    color c_o = foto_original.get(i, j);
    color c_d = color(min(255, red(c_o) * 1.5),
                     min(255, green(c_o) * 1.5),
                     min(255, blue(c_o) * 1.5));
    foto_modificada.set(i, j, c_d);
  }

// Mostramos las dos imágenes
size(500, 300);
image(foto_original, 0, 0, width/2, height);
image(foto_modificada, width/2, 0, width/2, height);
```



Imágenes

- Ejemplo:

```
// Efecto reflejo
PImage foto_original = loadImage("foto.jpg");
PImage foto_modificada = createImage(foto_original.width,
                                     foto_original.height / 3,
                                     ARGB);

// Cogemos un tercio de la imagen original,
// hacemos espejo vertical, y llevamos a cabo un gradiente
// de transparencias para crear un efecto de reflejo
int h23 = foto_original.height * 2 / 3;
int h13 = foto_original.height / 3;
for (int j = h23; j < foto_original.height; j++) {
    int alpha = int((j - h23) * (255.0 / h13)) - 128;
    for (int i = 0; i < foto_original.width; i++) {
        color c_o = foto_original.get(i, j);
        color c_d = color(red(c_o), green(c_o), blue(c_o), alpha);
        foto_modificada.set(i, h13 - (j - h23), c_d);
    }
}

size(1000, 700);
background(0);

// Imagen
image(foto_original, 50, 10);

// Imagen original con reflejo
image(foto_original, 500, 10);
image(foto_modificada, 500, foto_original.height + 10);
```

Imágenes



Imágenes

`save(nombre fichero)`

- Guarda la imagen en el formato gráfico que se especifique en la extensión del fichero.
- Los formatos disponibles son TIFF, TARGA, JPEG y PNG

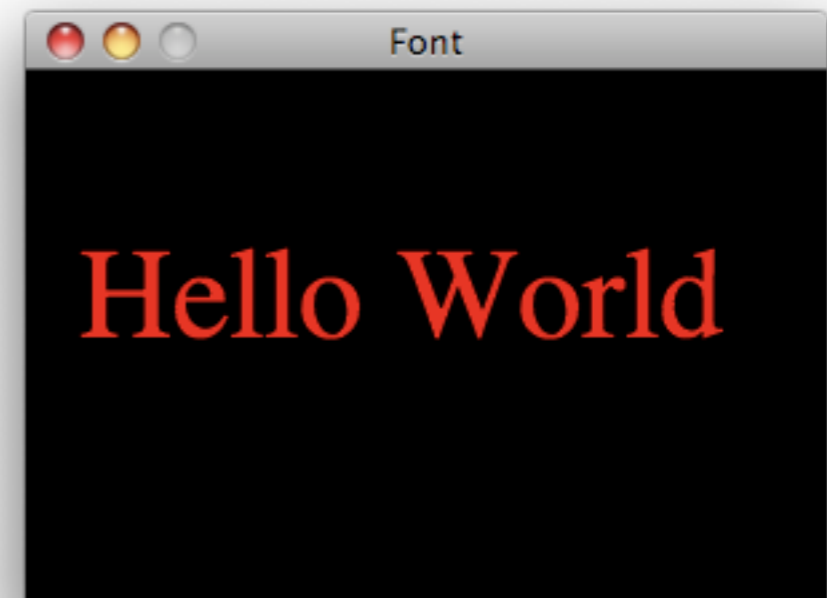
Tipografía

- *processing* dispone de la clase `PFont` para la creación de fuentes
- Las fuentes deben crearse y almacenarse en el directorio `data` de la aplicación. Para ello, el PDE de *processing* dispone de la opción 'Create Font' del menú de utilidades. Las fuentes se almacenan mediante la extensión `.vlw`
- Hay que cargar la fuente con `loadFont()` y seleccionarla con `loadFont()` antes de poder utilizarla con `text()`
- Existen gran cantidad de funciones y posibilidades: `textAlign()`, `textSize()`, `textMode()`, `textLeading()` **etc.**

Tipografía

- Ejemplo:

```
size(300, 200);  
background(0);  
  
// Cargamos fuente (previamente hay  
// que crearla con la correspondiente  
// opción del PDE de processing)  
PFont font = loadFont("Serif-48.vlw");  
  
// Seleccionamos fuente  
textFont(font);  
  
// Color fuente  
fill(255, 0, 0);  
  
// En x=10, y = 50 (hacia arriba)  
text("Hello World", 20, 100);
```



Práctica 3-2

- Calcular y visualizar el histograma de una imagen
- El histograma representa el número de veces que un determinado valor de intensidad está presente en una imagen concreta
- Puede calcularse para cada uno de los 3 canales por separado, o de forma integrada para los 3 canales (histograma RGB)
- El histograma RGB se calcula de la siguiente forma:
 - Abrir la imagen
 - Convertirla en escala de grises. Para ello se puede usar el método `filter()` de la siguiente forma: `im.filter(GRAY)`, que modifica la imagen (`im`) convirtiéndola en escala de grises (1 canal). Este único canal se encuentra ahora accesible en el canal RED: `red(im.get(x, y))` nos devuelve el correspondiente valor de gris tras la conversión anterior con `filter()`
 - Crear un array histograma: `int[] histograma = new int[256]`
 - Recorrer todos los píxeles de la imagen, canal RED, incrementando su contador en histograma: `histograma[red(im.get(x, y))]++`. Con ello, contabilizamos el número de veces que cada valor está presente en la imagen. Es conveniente también determinar el máximo valor de los almacenados en histograma.

Práctica 3-2

- Representar gráficamente el histograma, de forma que su anchura sea de 256, su altura de 100, y cada valor se dibuje como una línea
- Ejemplo:

