PRÁCTICA 9

Procesamiento de imágenes

Parte 1

Descripción de la práctica

El objetivo de esta práctica es realizar una aplicación que amplíe la realizada en la práctica 8, introduciendo nuevas funcionalidades relativas al procesamiento imágenes. Concretamente, deberá incluir las siguientes funcionalidades:

- Posibilidad de variar el brillo de la imagen.
- Aplicación de filtros básicos (emborronamiento, enfoque, relieve, fronteras, etc.)

El aspecto visual de la aplicación será el mostrado en la Figura 1. El menú incluirá, además de la opción "Archivo", una nueva opción "Imagen" en la cual iremos incorporando ítems asociados a operaciones básicas (con parámetros fijos); para esta práctica, esta opción incluirá los ítems "ResacaleOp" y "ConvolveOp". En la parte inferior, se incluirá un deslizador que permita modificar el brillo de la imagen activa, así como una lista desplegable con los distintos filtros que se puedan aplicar.

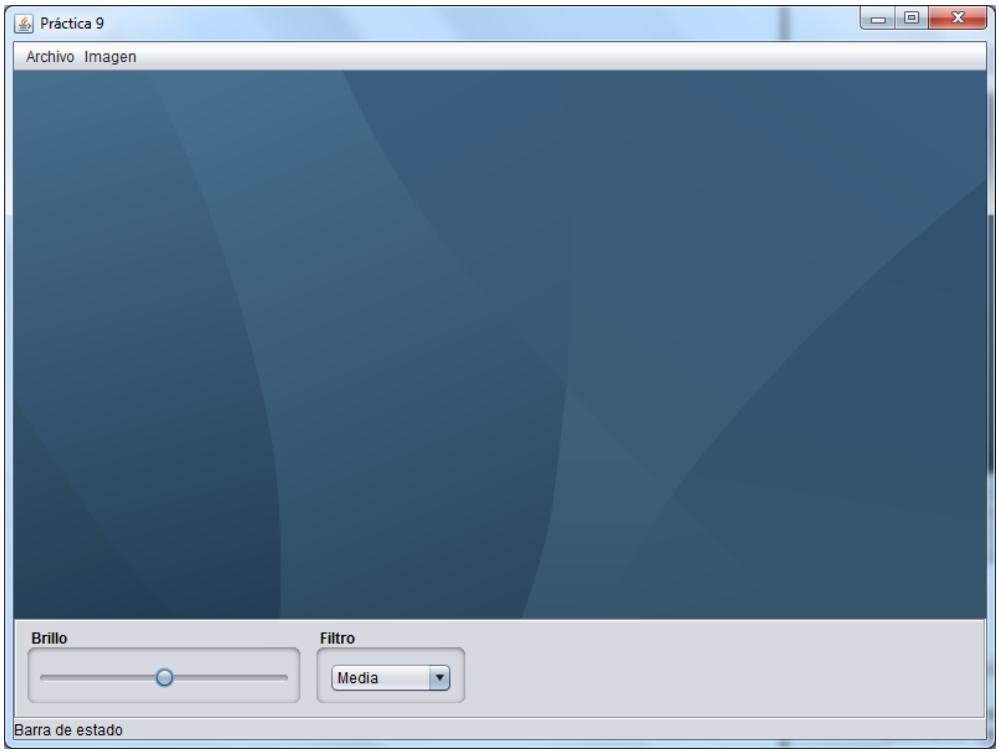

Figura 1: Aspecto de la aplicación

Pruebas iniciales

En una primera parte, probaremos los operadores "`RescaleOp`" y "`ConvolveOp`" usando parámetros fijos (i.e., sin interacción del usuario para definir sus valores). Para ello, incluiremos dos opciones en el menú "Imagen" cuya selección implicará la aplicación de la correspondiente operación en la imagen seleccionada.

En estos casos, se probará el código mostrado en las transparencias de teoría (que incluiremos en el manejador del evento "acción" asociado al menú), teniendo en cuenta que dicha operación se aplicará sobre la imagen mostrada en la ventana activa. Esto supondrá que, al código visto en teoría, habrá que añadirle las sentencias para el acceso y actualización de la imagen; por ejemplo, para el caso de la operación "RescaleOp":

```
private void menuRescaleOpActionPerformed(ActionEvent evt) {
  VentanaInterna vi = (VentanaInterna) (escritorio.getSelectedFrame());
  if (vi != null) {
    BufferedImage imgSource = vi.getLienzo().getImage();
    if(imgSource!=null){
      try{
         RescaleOp rop = new RescaleOp(1.0F, 100.0F, null);
         BufferedImage imgdest = rop.filter(imgSource, null);
         vi.getLienzo().setImage(imgdest);
         vi.getLienzo().repaint();
      } catch(IllegalArgumentException e){
         System.err.println(e.getLocalizedMessage());
      }
    }
  }
}
```

Variación del brillo

En este apartado modificaremos el brillo de la imagen aplicando el operador `RescaleOp`, pero sin establecer un valor fijo (como en el apartado anterior), sino permitiendo que éste sea definido por el usuario mediante un deslizador.

En este caso hay que tener en cuenta que la imagen sobre la que se aplica la operación ha de ser la original, y no la obtenida temporalmente durante el proceso[1]; esta consideración es importante a la hora de usar los métodos set/get de la clase `LienzoImagen`. Para abordar este aspecto, se pueden considerar diferentes alternativas:

- Independiente al lienzo. En este caso, una posible opción sería definir una variable de tipo `BufferedImage` en la ventana principal (donde se manejan los eventos asociados al deslizador); dicha variable representará la imagen fuente sobre la que se aplicará la operación, de forma que se le asignará valor cuando se inicie el cambio de brillo (por ejemplo, en el momento que el deslizador adquiere el foco[2])
- Vinculado al lienzo. En este caso, la clase `LienzoImagen` incluirá más de una imagen, distinguiendo entre la imagen fuente (sobre la que se aplicaría la operación) y la imagen vista (la que mostraría el lienzo mediante su método `paint`)[3]. Este hecho habrá que tenerlo en cuenta en las llamadas externas al lienzo (al abrir una imagen, al aplicar una determinada operación, etc.).

[1] Cada nuevo valor del deslizador (que será notificado mediante el evento `stateChange`) implicará calcular la imagen (temporal) resultado de aplicar ese valor de brillo (esta imagen se irá mostrando en la ventana mientras el usuario mueve el deslizador)

[2] En este caso, se aconseja asignarle valor `null` cuando pierda el foco

[3] En este enfoque, tendremos métodos set/get para cada imagen definida en el lienzo. Se recomienda que el método "set" de la imagen fuente implique la llamada al método "set" de la imagen vista (con la misma imagen); en un determinado momento podrán "desvincularse" ambas imágenes mediante la llamada al método "set" de la imagen vista (y volver a vincularse con la llamada al método "set" de la imagen fuente).

Aplicación de filtros

En este apartado aplicaremos diferentes filtros basados en el operador de convolución. Concretamente, se considerarán los siguientes filtros:

- Emborronamiento media
- Emborronamiento binomial
- Enfoque
- Relieve
- Detector de fronteras laplaciano

En la parte inferior de la aplicación se incluirá una lista desplegable con los cinco filtros anteriores, de forma que cuando el usuario seleccione uno de ellos, éste se aplique automáticamente a la imagen seleccionada.

Cada uno de estos filtros está asociado a una máscara de convolución. En la clase `KernelProducer`, que se adjunta a esta práctica, se incluyen varios ejemplos de máscaras, así como un método `createKernel` que devuelve objetos `Kernel` correspondiente a máscaras predefinidas. Por ejemplo, el siguiente código crearía una máscara para el filtro media:

```
Kernel k = KernelProducer.createKernel(KernelProducer.TYPE_MEDIA_3x3);
```