Sistemas multimedia (2018-2019)
Grado en Ingeniería Informática
Universidad de Granada

Domentación Paint. Práctica de Evaluación.

Montserrat Rodríguez Zamorano

4 de julio de 2019

Índice

1. Descripción del sistema

Para la evaluación de la asignatura de *Sistemas multimedia* se quiere realizar una aplicación multimedia que permita gestionar gráficos, imágenes, sonido y vídeo. Se podrá crear, editar, procesar y visualizar contenido multimedia de distintos tipos.

Esta aplicación tendrá una ventana principal con diferentes barras de herramientas y botones que permitirán realizar las distintas operaciones sobre el contenido multimedia. Se intentará que la interfaz sea lo más intuitiva posible y facilitar así el uso de la aplicación al usuario final.

Con este fin, se realiza una aplicación que se llamará *Paint*.

2. Requisitos

2.1. Requisitos funcionales

2.1.1. Carácter general

- **[RFCG-1]** Creación de una nueva imagen en una nueva ventana.
- **[RFCG-2** Abrir un fichero de imagen.
- **[RFCG-3]** Abrir un fichero de sonido.
- **[RFCG-4]** Abrir un fichero de vídeo.
- **[RFCG-5]** Guardar una imagen y sus figuras dibujadas.
- **[RFCG-6]** Ocultar las barras de herramientas.
- **[RFCG-7]** Visualizar las barras de herramientas.
- **[RFCG-8]** Consultar el nombre del programa, versión y autor.

2.1.2. Dibujo

- **[RFD-1]** Dibujar las siguientes formas geométricas con sus propios atributos independientes.
  - Línea.
  - Rectángulo.
  - Elipse.
  - Rectángulo con esquinas redondeadas.
- **[RFD-2]** Mantener todas las figuras que se vayan dibujando.
- **[RFD-3]** Elegir el color de trazo de dibujo.
  - Rojo.
  - Azul.
  - Negro.
  - Blanco.
  - Verde.
- **[RFD-4]** No rellenar la imagen.
- **[RFD-5]** Rellenar con color el dibujo.

    - Rojo.
    - Azul.
    - Negro.
    - Blanco.
    - Verde.
- **[RFD-6]** Seleccionar una figura dibujada.
- **[RFD-7]** Editar una figura dibujada.
- **[RFD-8]** Mover una figura dibujada.
- **[RFD-9]** Consultar los atributos de una figura dibujada.
- **[RFD-10]** Asociar un grado de transparencia a una figura.
- **[RFD-11]** Activar alisado de bordes de una figura.
- **[RFD-12]** Desactivar alisado de bordes de una figura.

2.1.3. Procesamiento de imágenes

- **[RFPI-1]** Duplicar una imagen.
- **[RFPI-2]** Modificar el brillo de una imagen.
- **[RFPI-3]** Aplicar filtro para emborronar una imagen.
- **[RFPI-4]** Aplicar filtro para enfocar una imagen.
- **[RFPI-5]** Aplicar filtro de relieve a una imagen.
- **[RFPI-6]** Aplicar contraste a una imagen.
- **[RFPI-7]** Iluminar una imagen.
- **[RFPI-8]** Oscurecer una imagen.
- **[RFPI-9]** Extraer las bandas de una imagen.
- **[RFPI-10]** Invertir los colores de una imagen.
- **[RFPI-11]** Convertir una imagen a los siguientes espacios:
    - RGB
    - YCC
    - GRAY
- **[RFPI-12]** Girar una imagen a cualquier ángulo.
- **[RFPI-13]** Tintar una imagen.
- **[RFPI-14]** Escoger nivel de tintado de una imagen.
- **[RFPI-15]** Ecualizar una imagen.
- **[RFPI-16]** Umbralizar una imagen en niveles de gris.
- **[RFPI-17]** Escoger nivel de umbralización.

- **[RFPI-18]** Aplicar filtro sepia a una imagen.
- **[RFPI-19]** Aplicar filtro violeta a una imagen.
- **[RFPI-20]** Aplicar filtro de media entre las bandas de colores.
- **[RFPI-21]** Aplicar filtro cosinosuidal.

2.1.4. Sonido

- **[RFS-1]** Reproducir audios.
- **[RFS-2]** Grabar sonidos.
- **[RFS-3]** Pausar la reproducción.
- **[RFS-4]** Parar la grabación.
- **[RFS-5]** Parar la reproducción.

2.1.5. Vídeo

- **[RFV-1]** Mostrar la secuencia que capte la Webcam.
- **[RFV-2]** Capturar una imagen desde la Webcam.

2.2. Requisitos no funcionales

- **[RNF-1]** Se mostrará en la barra de estado el pixel en el que está situado el ratón.
- **[RNF-2]** Se habilitarán en cada momento sólo los botones que pueden utilizarse. Por ejemplo, si se abre una ventana de vídeo se deshabilitarán aquellos correspondientes a las imágenes.
- **[RNF-3]** Los botones tendrán asociados un *ToolTipText* para facilitar el uso del programa.
- **[RNF-4]** Al seleccionar una figura se activarán sus propiedades en la barra de atributos.
- **[RNF-5]** Si hay una figura seleccionada, al pulsar el ratón sobre en otro punto, deberá deseleccionarse la figura.
- **[RNF-6]** El título de una nueva ventana abierta será el nombre del fichero si se trata de una imagen abierta o guardada.
- **[RNF-7]** El título de una nueva ventana abierta será *Nueva* si se trata de una imagen creada por el usuario.
- **[RNF-8]** El título de una nueva ventana abierta será *Captura* si se trata de una captura captada de un vídeo o de la WebCam.
- **[RNF-9]** La *BoundingBox* será un rectángulo de color azul, con línea discontinua.
- **[RNF-10]** Cuando se cree una imagen se lanzará un diálogo que permita elegir las dimensiones de la imagen.

3. Análisis

Se hará un análisis del problema y de las posibles soluciones, intentando que sea lo más independiente posible del lenguaje de programación. En el apartado de *Diseño* se especificarán detalles relacionados con la programación.

En primer lugar, se plantea cómo debería ser el área de dibujo, que almacenará los atributos con los que se pintará la siguiente figura. Con este fin se desarrollará una clase *Lienzo*. Esta clase será la encargada de pintar las figuras. Cada figura dibujada en el área de dibujo tendrá los atributos activos en ese momento en el lienzo, si bien sus atributos podrán modificarse a posteriori.

La solución planteada durante el desarrollo de las prácticas no es muy flexible: la biblioteca `java.awt.Graphics` ofrece importantes limitaciones, ya que no permite que los objetos tengan propiedades de color, forma o grosor, entre otras. De esta forma, todas las figuras de un mismo lienzo se pintarán con los mismos atributos.

Para que cada figura tenga sus propios atributos se desarrollará una clase propia *Figura*, que almacenará cada uno de los atributos de la figura y permitirá que cada figura tenga los suyos propios: color, grosor, etc. Sin embargo, la línea, por ejemplo, no puede tener relleno, por lo que no tiene sentido guardar este atributo. Se desarrollará una clase propia para cada una de las figuras que pueden dibujarse: Línea, rectángulo, elipse, rectángulo redondeado.

Para intentar mejorar el diseño de esta clase, en lugar de que las formas hereden directamente de *Figura*, se dividirán en dos grupos: figuras rellenables y figuras lineales. De esta forma, las operaciones comunes relativas al relleno podrán realizarse en el ámbito de la figura rellenable, y no tener que implementarlas en cada una de las clases derivadas. Con esta jerarquía de clases, evitamos que figuras lineales tengan atributos que no deben tener, como por ejemplo, tipo de relleno, así como evitamos que en figuras con propiedades similares (elipse y rectángulo, por ejemplo) se repitan funciones o atributos.

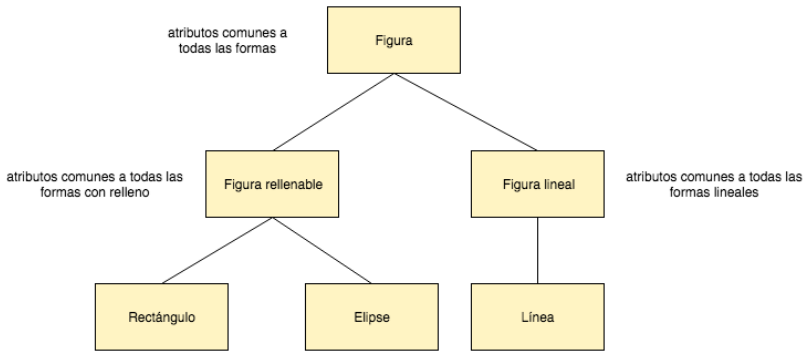

Figura 3.1: Esquema del diseño de la clase *Figura*

Uno de los métodos que tendrá que implementar en cada clase por separado será *setLocation*. En principio se pretendía aprovechar el método que proporcionaba `Graphics`, pero se ha optado por implementarlo en cada clase de forma manual, ya que para `Rectangle2D` y `Ellipse2D` no está disponible el uso de esta función.

En cuanto al manejo de eventos, se usará en muchas ocasiones las funciones que ofrece Java para la

gestión de eventos (por ejemplo, para el uso de botones en la ventana principal). Sin embargo, para la comunicación entre el lienzo y la ventana principal será necesario crear una clase manejadora para la gestión de eventos relacionados con el lienzo. La existencia de esta clase permitirá por ejemplo informar de la falta de atributos necesarios para el dibujo de las figuras y poder lanzar mensajes de error desde la ventana principal (por ejemplo: no se ha seleccionado una forma de dibujo) o informar a la ventana principal de la creación de una nueva figura, para que aparezca en una lista desplegable de figuras seleccionables.

4. Diseño

Siguiendo la propuesta que se ha planteado en el análisis, se plantea las siguientes jerarquías de clases. Mientras la propuesta anterior intentaba ser independiente del lenguaje de programación, en este apartado se especifican aspectos concretos de la implementación en *Java*.

Se hará una descripción general, pero para más detalles se pueden consultar los diagramas de clase.

4.1. VentanaPrincipal

La clase `VentanaPrincipal` implementará la visión principal del programa. Heredará de `JFrame` y contendrá múltiples elementos `Swing`:

- **Barras de herramientas:** serán del tipo `JToolBar`, y se tratan de las barras superior, izquierda y derecha. Serán las que contendrán la mayoría de los demás elementos `Swing`.
- **Barra de estado:** se trata de un elemento `JPanel` que contendrá elementos del tipo `JLabel`. El texto de estas etiquetas indicarán, por ejemplo, que una figura ha sido añadida.
- **Listas desplegables:** se utilizarán `JComboBox`.
- **Botones `JToggleButton`:** pueden quedarse seleccionados, por lo que se utilizarán para las formas, o para los botones de sonido, pudiendo agruparse en distintos grupos `ButtonGroup`.
- **Botones `JButton`:** En el caso de por ejemplo, los filtros, no tiene demasiado interés que un botón quede seleccionado, por lo que para la mayoría de las operaciones de procesamiento de imágenes se utilizará este tipo de botones.
- **Escritorio:** se trata de un elemento del tipo `JDesktopPane`. Este escritorio será aquel que contendrá las ventanas multimedia de distintos tipos.

4.2. Lienzo

4.3. Ventanas internas

Se definirá una clase `VentanaMultimedia` que heredará `JInternalFrame`. De esta clase derivarán diferentes ventanas en función del archivo multimedia que contengan.

- **VentanaMultimediaImagen:** se encargará de almacenar el contenido multimedia de tipo imagen. Tendrá un objeto de clase `LienzoImagen`, que será un rectángulo en blanco en el caso de que sea creada por el usuario, y una imagen en el caso de que sea una imagen abierta. Contendrá también un elemento de tipo `JScrollPane` para el caso en que la ventana no pueda contener la imagen completa.
- **VentanaMultimediaCamara:** se encargará de almacenar las imágenes que pueden verse desde la WebCam así como las funciones encargadas de gestionar este contenido.
- **VentanaMultimediaVLCPlayer:** se encargará de almacenar vídeos así como las funciones encargadas de gestionar este contenido.

4.4. Figura

Se definirá una clase abstracta `Figura`. Esta clase almacenará los atributos de la figura. Las clases `FiguraRellenable` y `FiguraLineal` heredarán de `Figura`. Aunque en este caso no hay más de una figura lineal, este esquema permite que en posteriores versiones se añadan curvas u otro tipo de figuras lineales. De `FiguraRellenable` heredarán `Rectangulo`,`Elipse` y `RectanguloRedondeado`.

De `FiguraLineal` heredará la clase `Linea`. Cada una de estas clases tendrá asociado un objeto `Shape`, además de los atributos de las clases de las que hereda.

Es esta jerarquía de clases la que permite que cada figura tenga sus propios atributos, así como la posibilidad de cambiarlos.

4.5. Gestión de eventos

Se hará uso de las funciones de gestión de eventos que ofrece *Java*. Sin embargo, es necesario implementar las funciones para la gestión de eventos en el lienzo, de forma que podamos establecer una comunicación entre lienzo y ventana principal.

- **LienzoEvent:**
- **LienzoListener:**
- **LienzoAdapter:**
- **MiManejadorLienzo:**

4.6. Procesamiento de imágenes

4.6.1. Operador *LookUpOp*

4.6.2. Componente a componente

La función componente a componente modificará cada una de las componentes (R, G, B) de forma separada.

En este caso se ha implementado un filtro violeta. Para aplicar un tono morado a la imagen, se tendrá en cuenta que la mezcla de rojo y azul es la que da lugar al morado. Por tanto, para obtener el efecto deseado se reducirá el valor de la componente verde y se aumentará el valor de la componente roja y azul.

Además, como se ha aumentado el valor de las componentes de forma significativa, habrá un aumento de la luminosidad en estos colores y de la intensidad de la imagen en general.

```
for(int x=0; x<src.getWidth(); x++){
    for(int y=0; y<src.getHeight(); y++){
        int destR, destG, destB;
        int srcR, srcG, srcB;
        Color colorSrc = new Color(src.getRGB(x, y));
        Color colorDest;
        //color del pixel
        srcR = colorSrc.getRed();
        srcG = colorSrc.getGreen();
        srcB = colorSrc.getBlue();
        //operacion componente a componente
        destR = (int) Math.min(255,1.3*srcR);
        destG = (int) Math.min(255,0.2*srcG);
        destB = (int) Math.min(255,1.5*srcB);

        colorDest = new Color(destR, destG, destB);
        dest.setRGB(x, y, colorDest.getRGB());
     }
}
```

Listing 1: Operación componente a componente

El resultado de aplicar el filtro puede verse a continuación. Como se puede ver, se obtiene el resultado deseado.

Figura 4.1: Aplicación filtro violeta sobre `Fry.jpg`

Figura 4.2: Aplicación filtro violeta

4.6.3. Primera operación píxel a píxel

La función píxel a píxel modificará cada componente teniendo en cuenta los demás componentes, de forma que el valor de cada componente podrá ser una combinación de las otras.

La operación implementada en este caso es una especie de "filtro rojizo", aunque no llega a serlo como tal. El valor del color rojo se conservará tal y como está, mientras que los valores verde y azul serán una media de los otros tres, por lo que aquellos píxeles en los que estos valores sean predominantes tendrán un color grisáceao.

Se espera que en la primera aplicación del filtro, la imagen tomará este color rojizo-anaranjado, pues habrá sido la única componente que habrá mantenido su valor. Sin embargo, al estar esta componente presente en la media que asigna los valores verde y azul, a través de las sucesivas aplicaciones del filtro estos dos valores convergerán al valor de la componente roja, convirtiéndose en un filtro blanco y negro.

```
for(int x=0; x<src.getWidth(); x++){
    for(int y=0; y<src.getHeight(); y++){
        int destR, destG, destB;
        int srcR, srcG, srcB;
        Color colorSrc = new Color(src.getRGB(x, y));
```

```
        Color colorDest;
        //color del pixel
        srcR = colorSrc.getRed();
        srcG = colorSrc.getGreen();
        srcB = colorSrc.getBlue();
        //operacion pixel a pixel
        destR = srcR;
        destG = Math.min(255,(srcR+srcG+srcB)/3);
        destB = Math.min(255,(srcR+srcG+srcB)/3);

        colorDest = new Color(destR, destG, destB);
        dest.setRGB(x, y, colorDest.getRGB());
    }
}
```

Listing 2: Operación pixel a pixel

Figura 4.3: Aplicaciones sucesivas operación píxel a píxel (I)

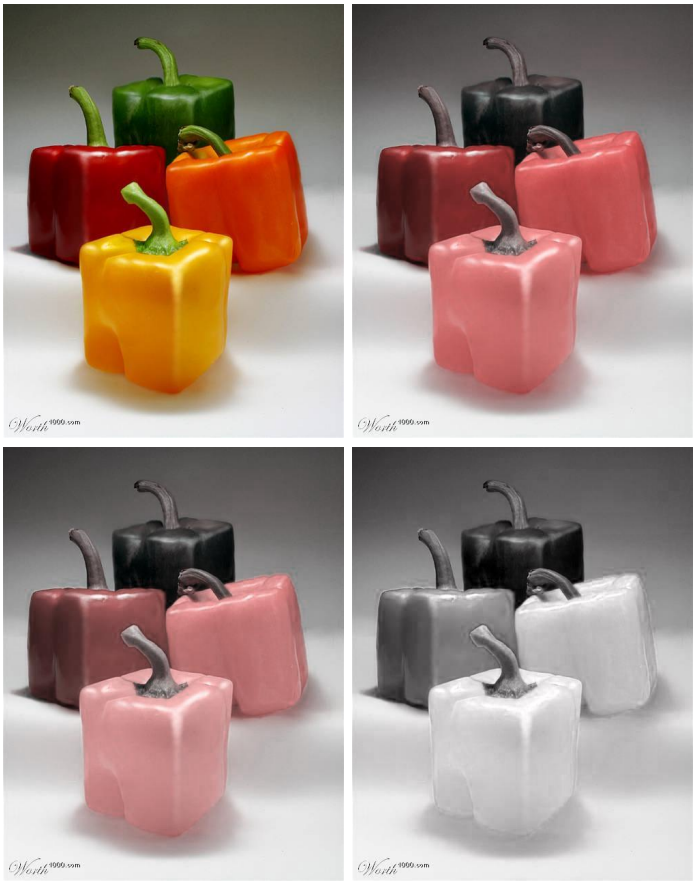

Figura 4.4: Aplicaciones sucesivas operación píxel a píxel (II)

4.7. Segunda operación píxel a píxel

Esta función es una variante de la primera, en la que se pretende que el filtro anterior no sea a través de las aplicaciones un filtro blanco y negro, sino un filtro que potencie el color rojo. Para ello, simplemente se elimina la componente roja de la media, de forma que los valores azules y verde no convergerán al valor de la componente roja.

```
for(int x=0; x<src.getWidth(); x++){
    for(int y=0; y<src.getHeight(); y++){
        int destR, destG, destB;
        int srcR, srcG, srcB;
        Color colorSrc = new Color(src.getRGB(x, y));
        Color colorDest;
        //color del pixel
        srcR = colorSrc.getRed();
        srcG = colorSrc.getGreen();
        srcB = colorSrc.getBlue();
        //operacion pixel a pixel
        destR = srcR;
        destG = (srcG+srcB)/2;
        destB = (srcG+srcB)/2;
```

```
            colorDest = new Color(destR, destG, destB);
            dest.setRGB(x, y, colorDest.getRGB());
        }
}
```

Listing 3: Operación pixel a pixel

A continuación pueden verse ejemplos de la aplicación de este filtro, que ofrece el resultado esperado.

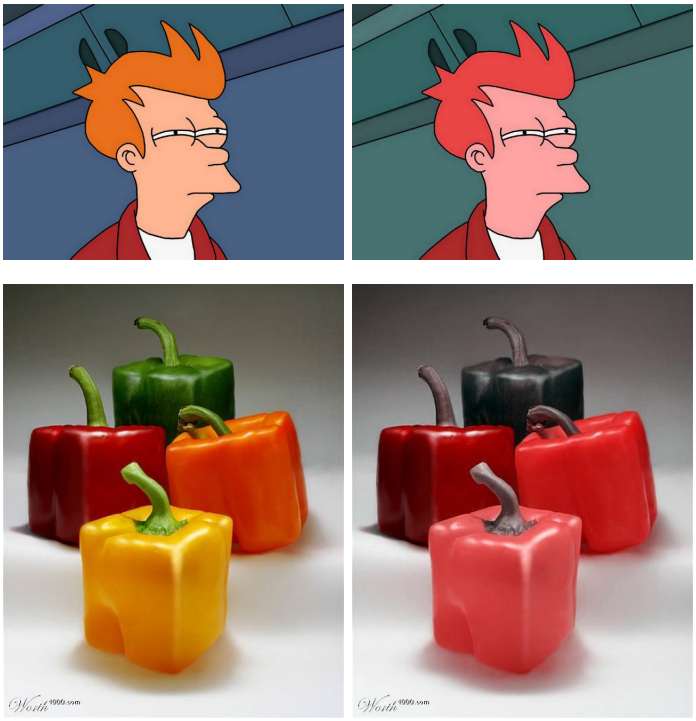

Figura 4.5: Aplicación filtro rojizo

5. Implementación

A continuación se muestra el resultado de la implementación de las clases diseñadas en el apartado anterior.

5.1. Ventana principal

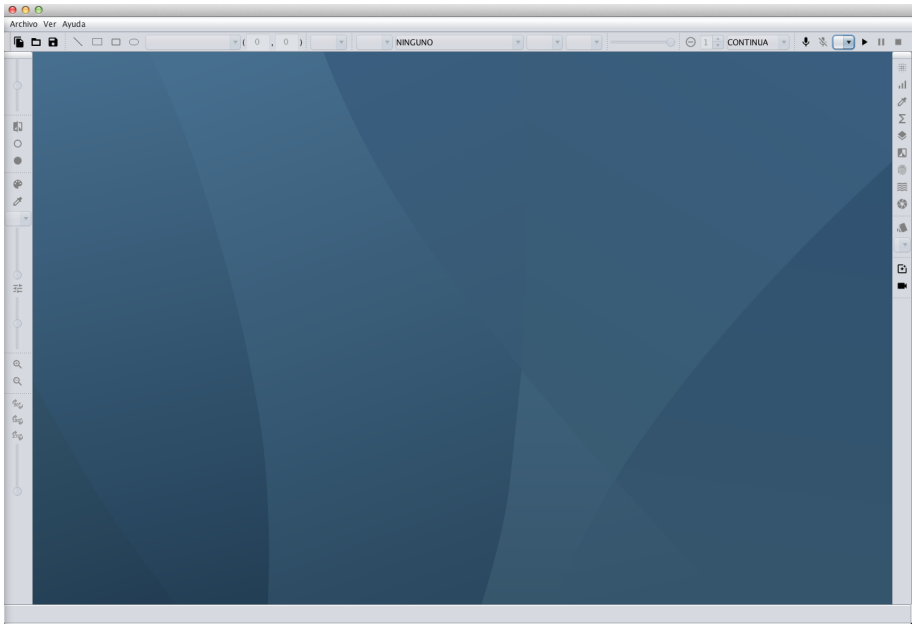

Figura 5.1: Vista `VentanaPrincipal`

5.2. VentanaImagen

Figura 5.2: Vista `VentanaImagen`

6. Validación

En este apartado se comprobará que el programa implementado en base a la solución propuesta cumple con los requisitos enumerados en el primer apartado.

6.1. Gráficos

6.2. Procesamiento de imágenes

6.2.1. Duplicar

Se comprobará si la duplicación de la imagen se ha realizado correctamente, de forma que si se modifica una, los cambios no se aplican en la otra.

Figura 6.1: Duplicar imagen

6.2.2. Modificar brillo

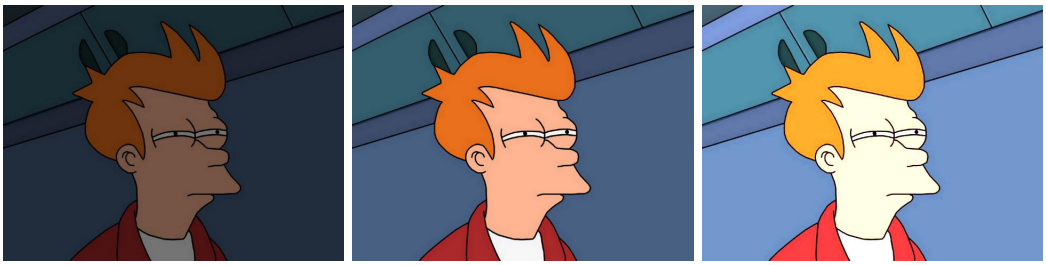

Figura 6.2: Modificación del brillo

6.2.3. Filtro de emborronamiento

Se ha aplicado varias veces el filtro para que sea visible.

Figura 6.3: Aplicación filtro emborronamiento

6.2.4. Filtro de enfoque

Figura 6.4: Aplicación filtro enfoque

6.2.5. Filtro de relieve

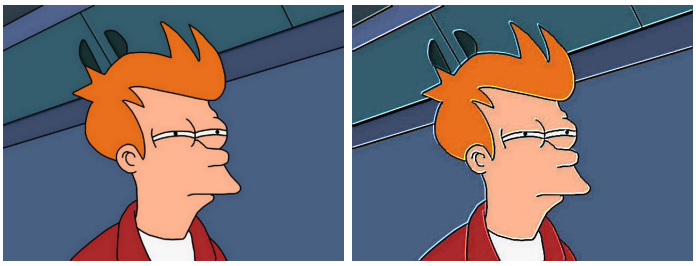

Figura 6.5: Aplicación filtro relieve

6.2.6. Contraste normal

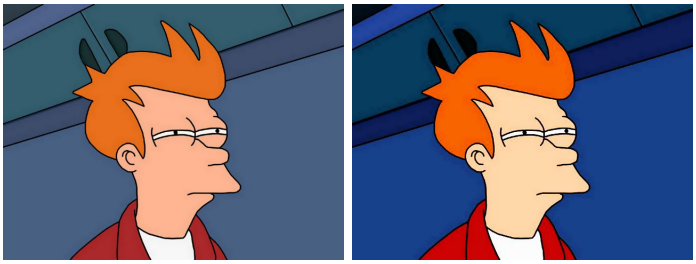

Figura 6.6: Aplicación contraste

6.2.7. Iluminado

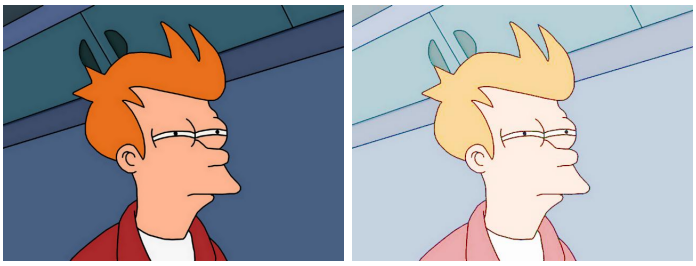

Figura 6.7: Aplicación iluminación

6.2.8. Oscurecido

Figura 6.8: Aplicación oscurecido

6.2.9. Invertir colores

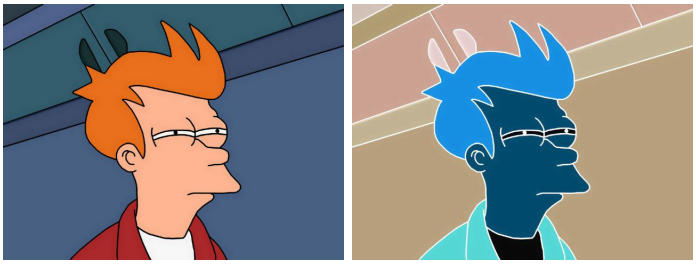

Figura 6.9: Aplicación filtro negativo

6.2.10. Conversión a espacios RGB, YCC, GRAY

Para comprobar que la conversión se ha realizado correctamente, se extraen las bandas de colores y se comparan. En la imagen se pueden ver la tercera banda de una imagen en espacio de color RGB y YCC, respectivamente.

Figura 6.10:

6.2.11. Giro libre

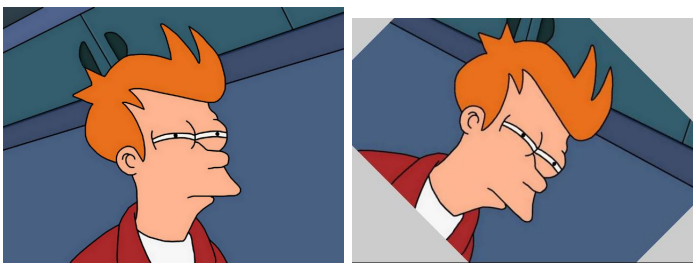

Figura 6.11: Giro libre

6.2.12. Escalado

6.2.13. Tintado

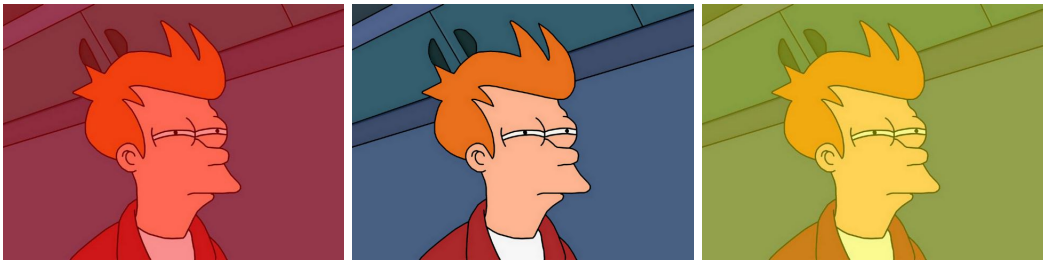

Figura 6.12: Tintado de la imagen

6.2.14. Ecualización

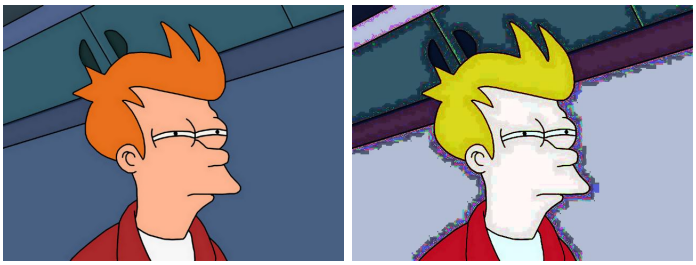

Figura 6.13: Aplicación ecualización

6.2.15. Filtro sepia

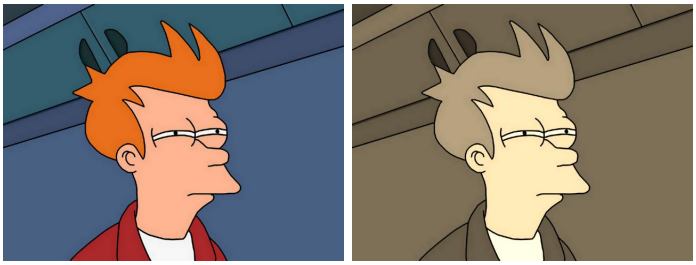

Figura 6.14: Aplicación filtro sepia

6.2.16. Umbralización

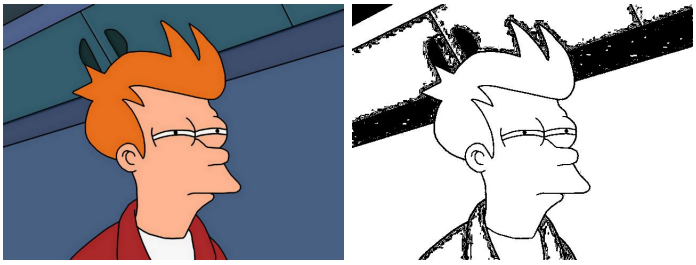

Figura 6.15: Umbralizar imagen

6.2.17. Operador *LookupOp* basado en una función propia

Figura 6.16: Operación cosinusoide

6.2.18. Operación de diseño propio: filtro violeta

Figura 6.17: Aplicación filtro violeta

6.3. Sonido

6.4. Vídeo

A continuación se muestran las funcionalidades del vídeo y la cámara.

6.4.1. WebCam

En primer lugar, comprobamos que puede abrirse correctamente la cámara. Una vez hecho esto, se comprobará si se realizan las capturas de pantalla correctamente. Si no hay ventana de cámara abierta, no se podrá sacar ninguna captura de pantalla.

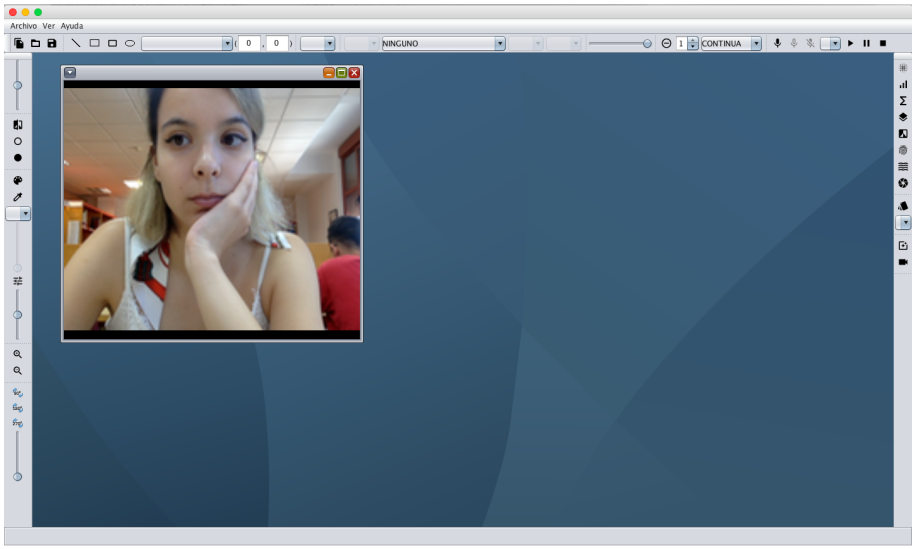

Figura 6.18: Webcam abierta

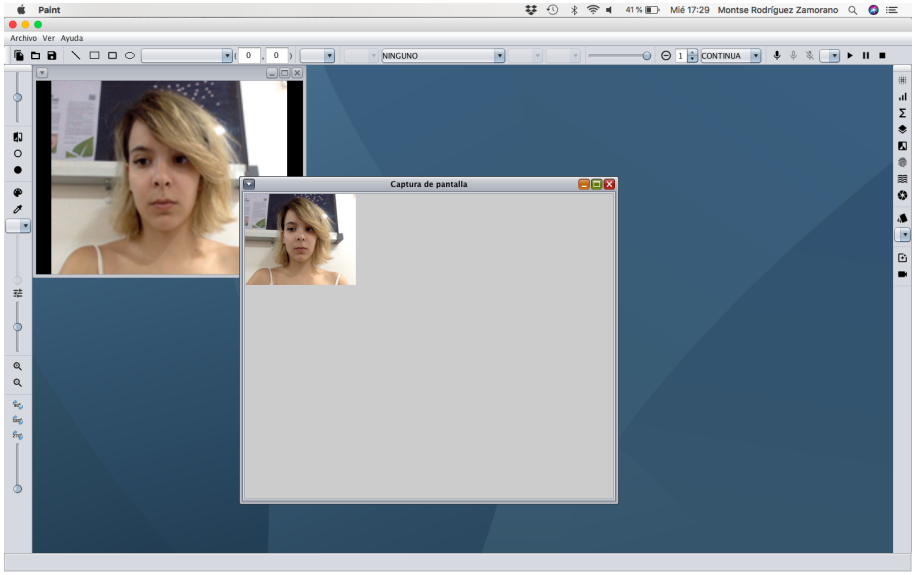

Figura 6.19: Captura de pantalla