Informática Gráfica
Curso 2023/24
QFotoPaint

Autores:
Pablo Carrasco Egea - 1.2
José Manuel Lorca Aguilar - 1.1

Índice:

1. Añadir foto del portapapeles:

Esta función se sitúa en el menú archivo del frontend, en este caso, la acción "Abrir portapapeles". Se utiliza para obtener una imagen del portapapeles del sistema, convertirla en un formato que pueda ser procesado por la biblioteca OpenCV y luego procesarla.

Primero, se accede al portapapeles del sistema utilizando QApplication::clipboard(), recomendada por el profesor. Luego, se obtienen los datos del portapapeles en formato QMimeData, que es un formato que puede contener varios tipos de datos, incluyendo imágenes, texto, etc.

A continuación, se comprueba si los datos obtenidos del portapapeles contienen una imagen. Si es así, se extrae la imagen y se convierte en un objeto QImage.

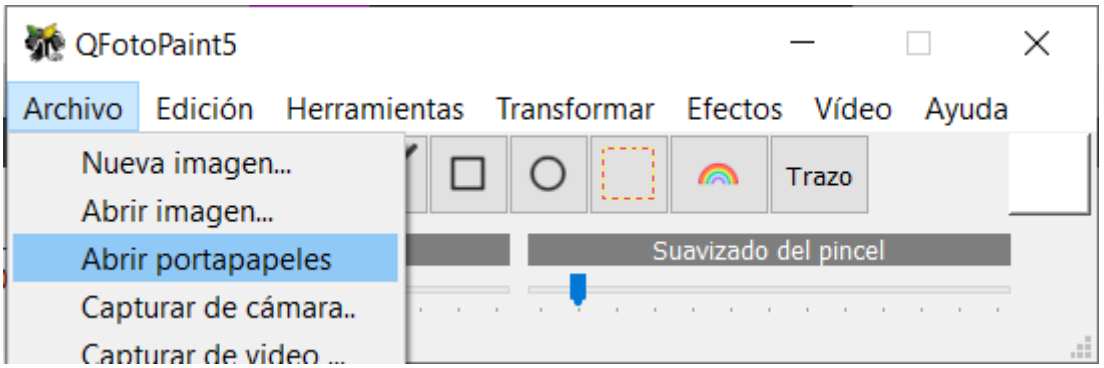
Luego, se convierte la imagen de QImage a una matriz OpenCV (cv::Mat) para poder procesarla con OpenCV. Esto se hace solo si la imagen no es nula. La imagen se convierte a BGR (Blue, Green, Red) ya que OpenCV utiliza este formato de color por defecto.

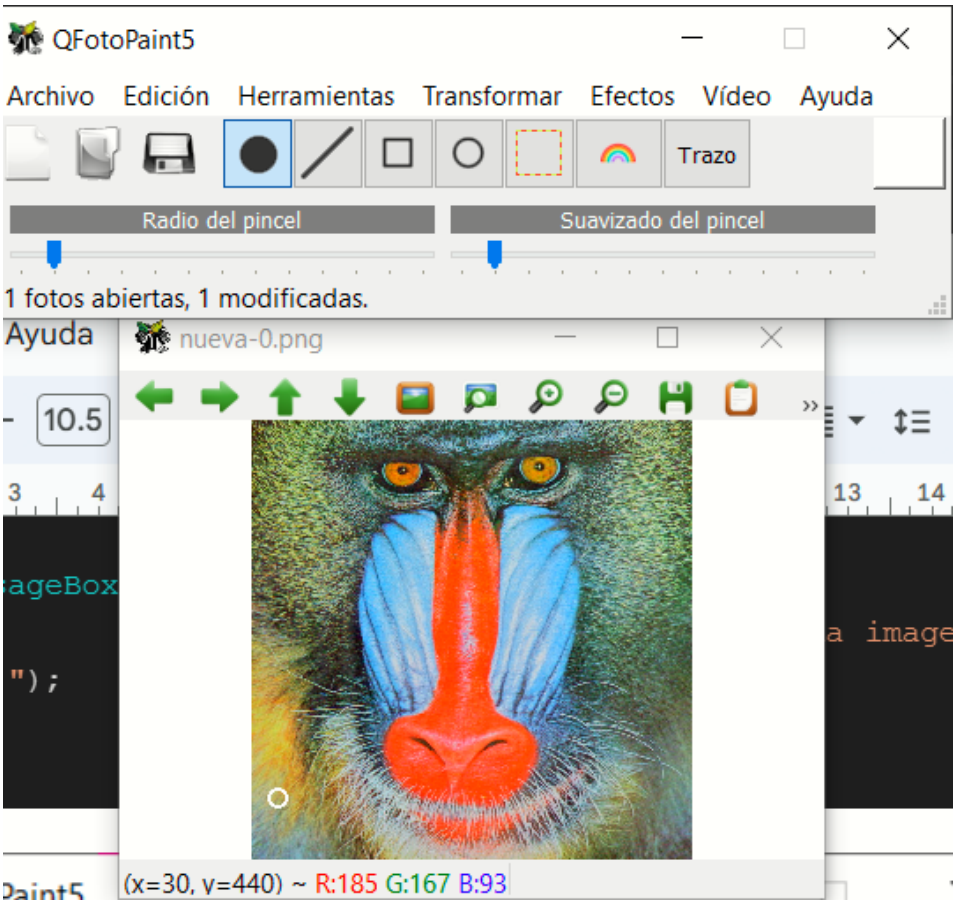
Después de la conversión, el código verifica si hay un espacio libre para la nueva imagen. Si hay un espacio libre, se crea una nueva imagen.

Si los datos del portapapeles no contienen una imagen, se muestra una advertencia al usuario con un cuadro de mensaje QMessageBox::warning() que dice "No se encontró ninguna imagen en el portapapeles".

```
void MainWindow::on_actionAbrir_portapapeles_triggered()
{
    QClipboard *clipboard = QApplication::clipboard();
    const QMimeData *mimeData = clipboard->mimeData();

    if (mimeData->hasImage()) {
        QImage image = qvariant_cast<QImage>(mimeData->imageData());

        // Convertir QImage a matriz OpenCV (Mat)
        Mat cvImage;
        if (!image.isNull()) {
                    cvImage = Mat(image.height(), image.width(), CV_8UC4,
const_cast<uchar*>(image.constBits()), image.bytesPerLine());
                cvtColor( cvImage,  cvImage, COLOR_RGBA2BGR);
        }

        int pl = comprobar_primera_libre();
        if (pl != -1) {
```

```
            crear_nueva(pl, cvImage);
        }
    } else {
        QMessageBox::warning(this, "Portapapeles vacío",
                                   "No se encontró ninguna imagen en el
portapapeles.");
    }
}
```





2. Ver Información de la foto:

Esta función se activa cuando se dispara cuando se pulsa el botón del menú edición, y su propósito principal es mostrar información sobre una imagen activa.

Primero, se obtiene el índice de la imagen activa con la función foto_activa(). Si hay una imagen activa, se accede a la imagen activa.

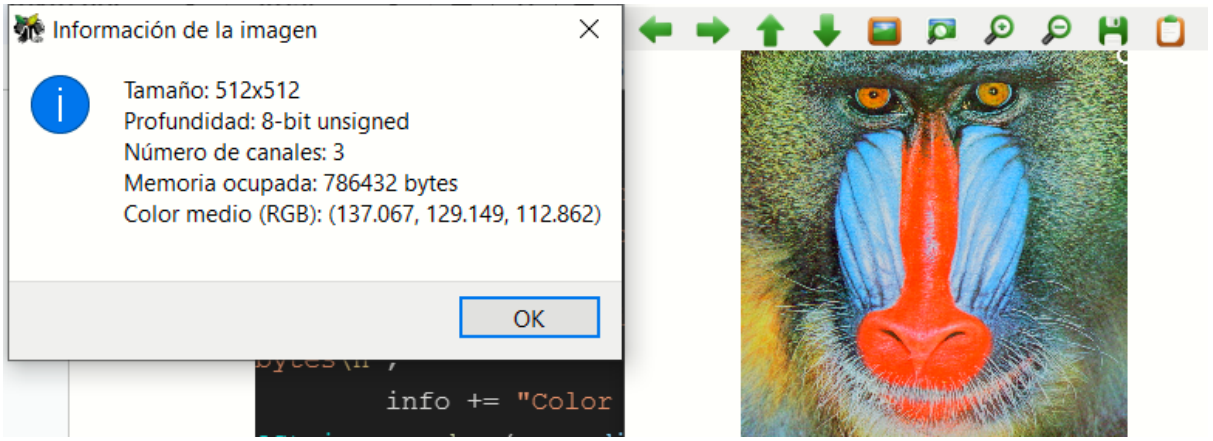
Luego, se obtiene información sobre la imagen. Esto incluye el ancho y el alto de la imagen, la profundidad de la imagen, el número de canales de la imagen y la memoria ocupada por la imagen. También se calcula el color medio de la imagen con la función mean().

La profundidad de la imagen se traduce a una cadena de texto legible utilizando una estructura switch. Dependiendo del valor de depth, se asigna una cadena de texto correspondiente a la variable profundidad.

Finalmente, se muestra la información de la imagen en una ventana de mensaje. Se crea una cadena de texto info que contiene toda la información de la imagen, y luego se muestra esta cadena en un cuadro de mensaje con QMessageBox::information().

```
void MainWindow::on_actionInformacion_foto_triggered() {
    int fa = foto_activa();
    if (fa != -1) {
        Mat imagen = foto[fa].img; // Acceder a la imagen activa

        // Obtener información sobre la imagen
        int width = imagen.cols;
        int height = imagen.rows;
        int depth = imagen.depth();
        int channels = imagen.channels();
        size_t memory = imagen.total() * imagen.elemSize(); // Memoria
ocupada

        Scalar promedioColor = mean(imagen); // Color medio de la
imagen

        // Traducción de la profundidad
        QString profundidad;
        switch (depth) {
            case CV_8U:
                profundidad = "8-bit unsigned";
                break;
            case CV_8S:
                profundidad = "8-bit signed";
```

```
                break;
            case CV_16U:
                profundidad = "16-bit unsigned";
                break;
            case CV_16S:
                profundidad = "16-bit signed";
                break;
            case CV_32S:
                profundidad = "32-bit signed";
                break;
            case CV_32F:
                profundidad = "32-bit float";
                break;
            case CV_64F:
                profundidad = "64-bit float";
                break;
            default:
                profundidad = "Desconocida";
        }

        // Mostrar la información en una ventana de mensaje
        QString info = "Tamaño: " + QString::number(width) + "x" +
QString::number(height) + "\n";
        info += "Profundidad: " + profundidad + "\n";
        info += "Número de canales: " + QString::number(channels) +
"\n";
        info += "Memoria ocupada: " + QString::number(memory) + "
bytes\n";
        info += "Color medio (BGR): (" +
QString::number(promedioColor[2]) + ", " +

QString::number(promedioColor[1]) + ", " +

QString::number(promedioColor[0]) + ")\n";

        QMessageBox::information(this, "Información de la imagen",
info);
    }
}
```


Información de la imagen
Tamaño: 512x512
Profundidad: 8-bit unsigned
Número de canales: 3
Memoria ocupada: 786432 bytes
Color medio (RGB): (137.067, 129.149, 112.862)
OK
info += "Color

3. Trazos

Para esta función, básicamente hemos copiado la función de puntos, y entre cada punto que se dibuje en la foto, añadimos una línea, para que no haya espacios entre los puntos.

Con cada ejecución de cb_trazo(), guardaremos en una variable de tipo Point el último punto que se ha dibujado.

Como se puede observar, después de pintar con puntos, usamos la función line() que hace una línea entre el último punto que hemos guardado en la anterior ejecución y el actual.

También hemos añadido un nuevo tipo de herramienta al enum tipo_herramienta de imagenes.h

El siguiente paso será añadir un case en el callback para cuando la herramienta trazo esté activa. En ese caso se ejecutará la función cb_trazo.

Por último, añadimos dos botones en la mainwindow para que se pueda activar la herramienta de trazo.

```
void cb_trazo(int factual, int x, int y)
{
    Mat im = foto[factual].img;  // Ojo: esto no es una copia, sino a la misma imagen
    if (difum_pincel == 0)
    {
        circle(im, Point(x, y), radio_pincel, color_pincel, -1, LINE_AA);
    }
    else
    {
        int tam = radio_pincel + difum_pincel;
        Rect roi(x - tam, y - tam, 2 * tam + 1, 2 * tam + 1);
        int posx = tam, posy = tam;
        if (roi.x < 0)
        {
            roi.width += roi.x;
            posx += roi.x;
            roi.x = 0;
        }
        if (roi.y < 0)
        {
            roi.height += roi.y;
            posy += roi.y;
            roi.y = 0;
        }
        if (roi.x + roi.width > im.cols)
        {
            roi.width = im.cols - roi.x;
        }
        if (roi.y + roi.height > im.rows)
        {
            roi.height = im.rows - roi.y;
        }
        Mat trozo = im(roi);
        Mat res(trozo.size(), im.type(), color_pincel);
        Mat cop(trozo.size(), im.type(), CV_RGB(0, 0, 0));
        circle(cop, Point(posx, posy), radio_pincel, CV_RGB(255, 255, 255), -1, LINE_AA);
        blur(cop, cop, Size(difum_pincel * 2 + 1, difum_pincel * 2 + 1));
        multiply(res, cop, res, 1.0 / 255.0);
        bitwise_not(cop, cop);
        multiply(trozo, cop, trozo, 1.0 / 255.0);
        trozo = res + trozo;
    }

    // Dibujar líneas entre puntos consecutivos
    static Point anterior= Point(x, y);
    line(im, anterior, Point(x, y), color_pincel, radio_pincel * 2, LINE_AA);
    anterior = Point(x, y);

    imshow(foto[factual].nombre, im);
    foto[factual].modificada = true;
}
```

cb_trazo() en imagenes.cpp

```
enum tipo_herramienta {HER_PUNTO, HER_LINEA, HER_SELECCION,
                       HER_RECTANGULO, HER_ELIPSE, HER_ARCO_IRIS, HER_TRAZO};
```

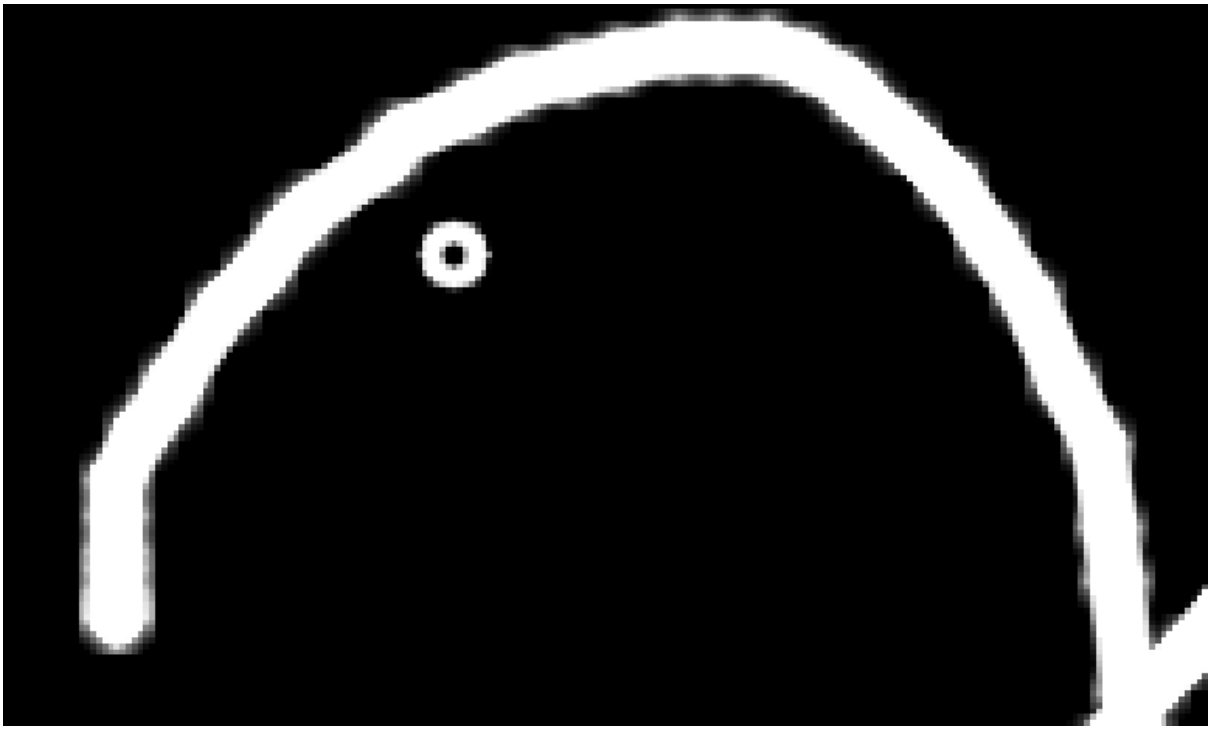
enum tipo_herramienta de imagenes.h

```
        // 2.6. Herramienta TRAZO
    case HER_TRAZO:
        if (flags==EVENT_FLAG_LBUTTON)
            cb_trazo(factual, x, y);
        else
            ninguna_accion(factual, x, y);
        break;
    }
```

Nuevo case en callback() de imagenes.cpp

```
void MainWindow::on_toolButton_10_clicked()
{
     herr_actual= HER_TRAZO;
}

void MainWindow::on_actionTrazo_triggered()
{
    herr_actual= HER_TRAZO;
    ui->toolButton_10->setChecked(true);
}
```

Nuevos botones en MainWindow



Ejemplo de trazo