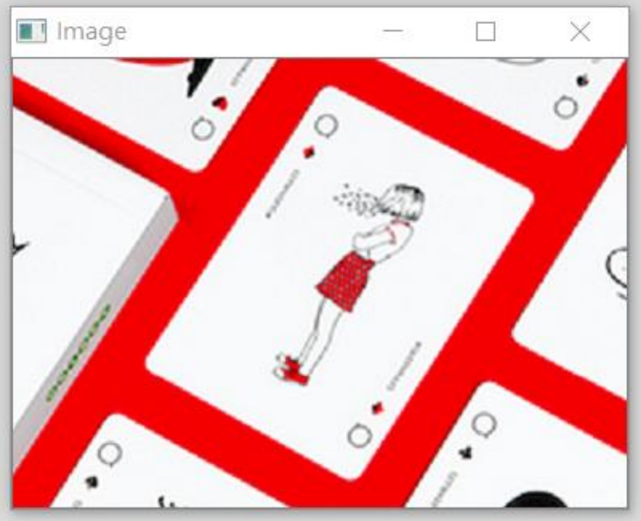
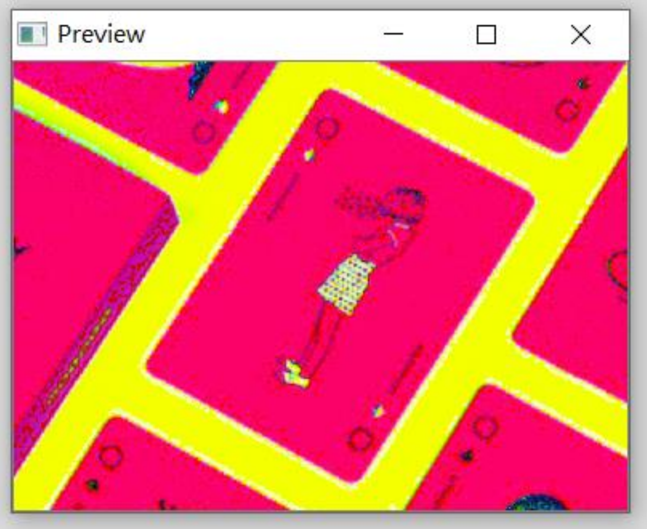
Qt Photoshop

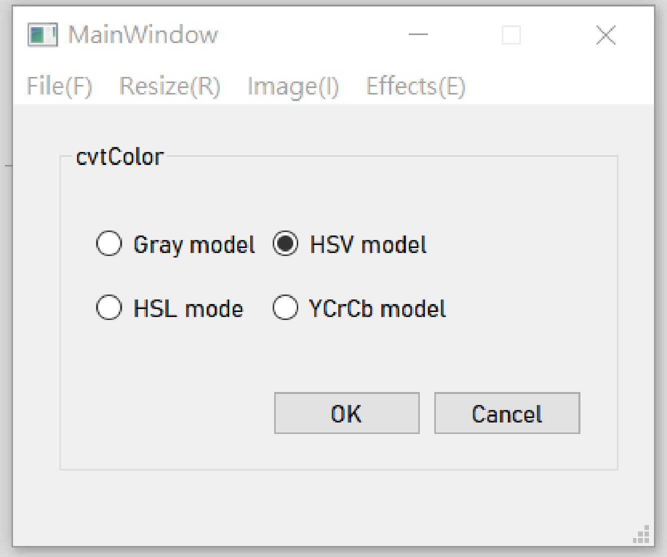
B08611035 柯鉑霆



Original Image



Preview Image



cvtColor

○ Gray model ● HSV model

○ HSL mode ○ YCrCb model

OK Cancel

Console Windows

Using stack widgets to switch to different function

With dialog button for user to check that using this preview effects or not.

OUTLINE

Drag / Drop

Crop / Resize / Rotate

Blur / Sharpen

Color Space

Channel

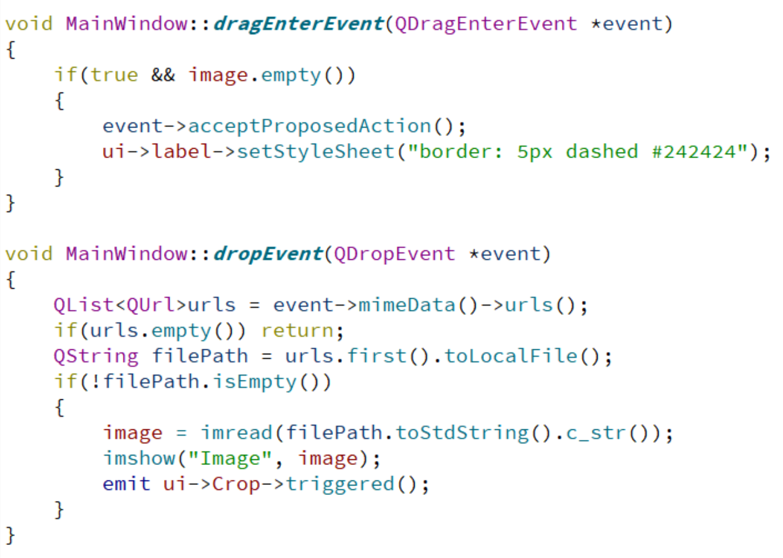
Color / Special Filter

DRAG / DROP

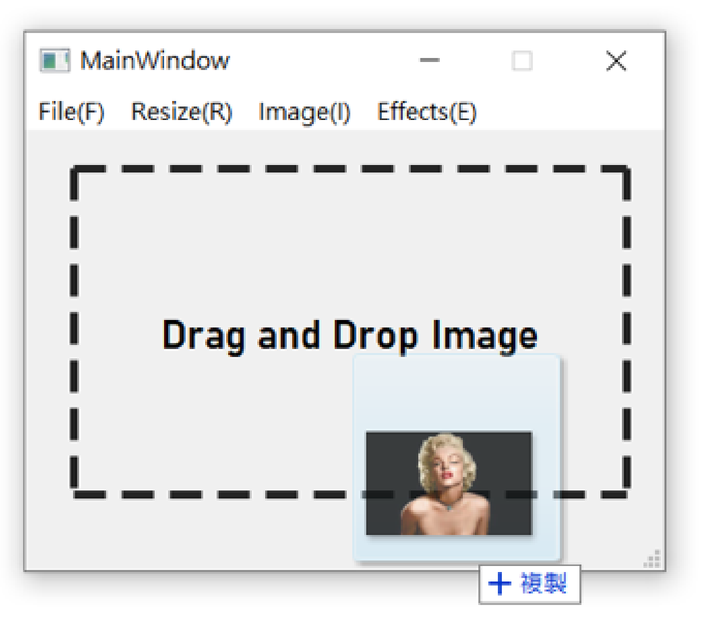
QDragEnterEvent

QDropEvent

Qt Style Sheet for border of label



```
void MainWindow::dragEnterEvent(QDragEnterEvent *event)
{
    if(true && image.empty())
    {
        event->acceptProposedAction();
        ui->label->setStyleSheet("border: 5px dashed #242424");
    }
}

void MainWindow::dropEvent(QDropEvent *event)
{
    QList<QUrl>urls = event->mimeData()->urls();
    if(urls.empty()) return;
    QString filePath = urls.first().toLocalFile();
    if(!filePath.isEmpty())
    {
        image = imread(filePath.toStdString().c_str());
        imshow("Image", image);
        emit ui->Crop->triggered();
    }
}
```



CROP

setMouseCallBack

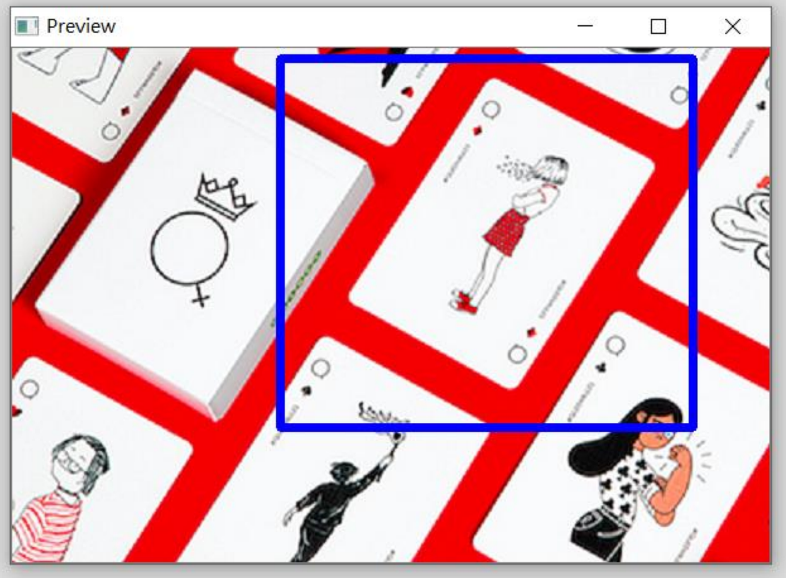
Self – revise onMouse

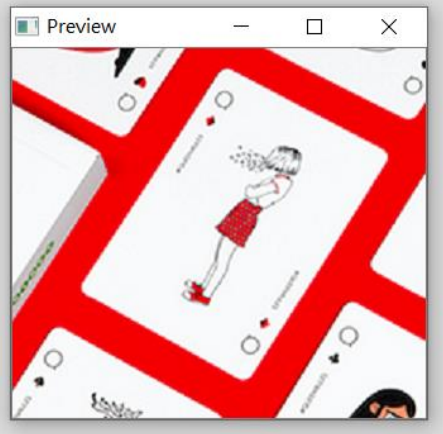
Type :

1. Self-selected Region crop

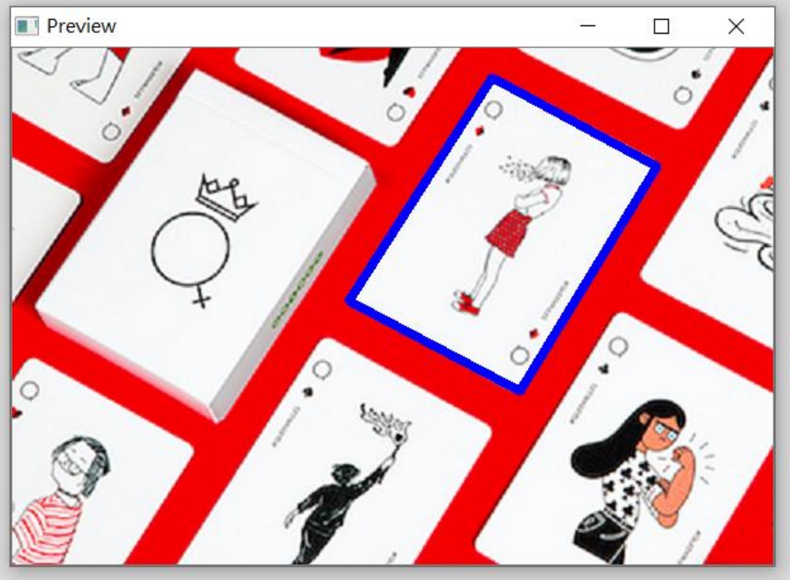
2. Rectangle Region crop

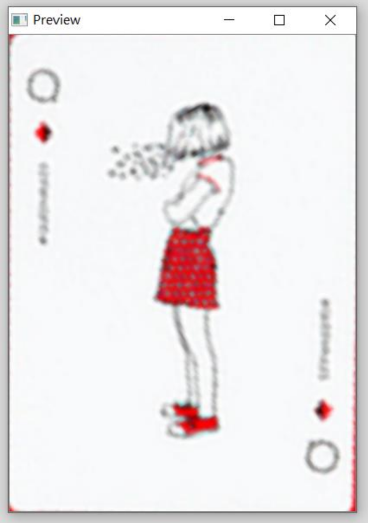
Rectangle Region

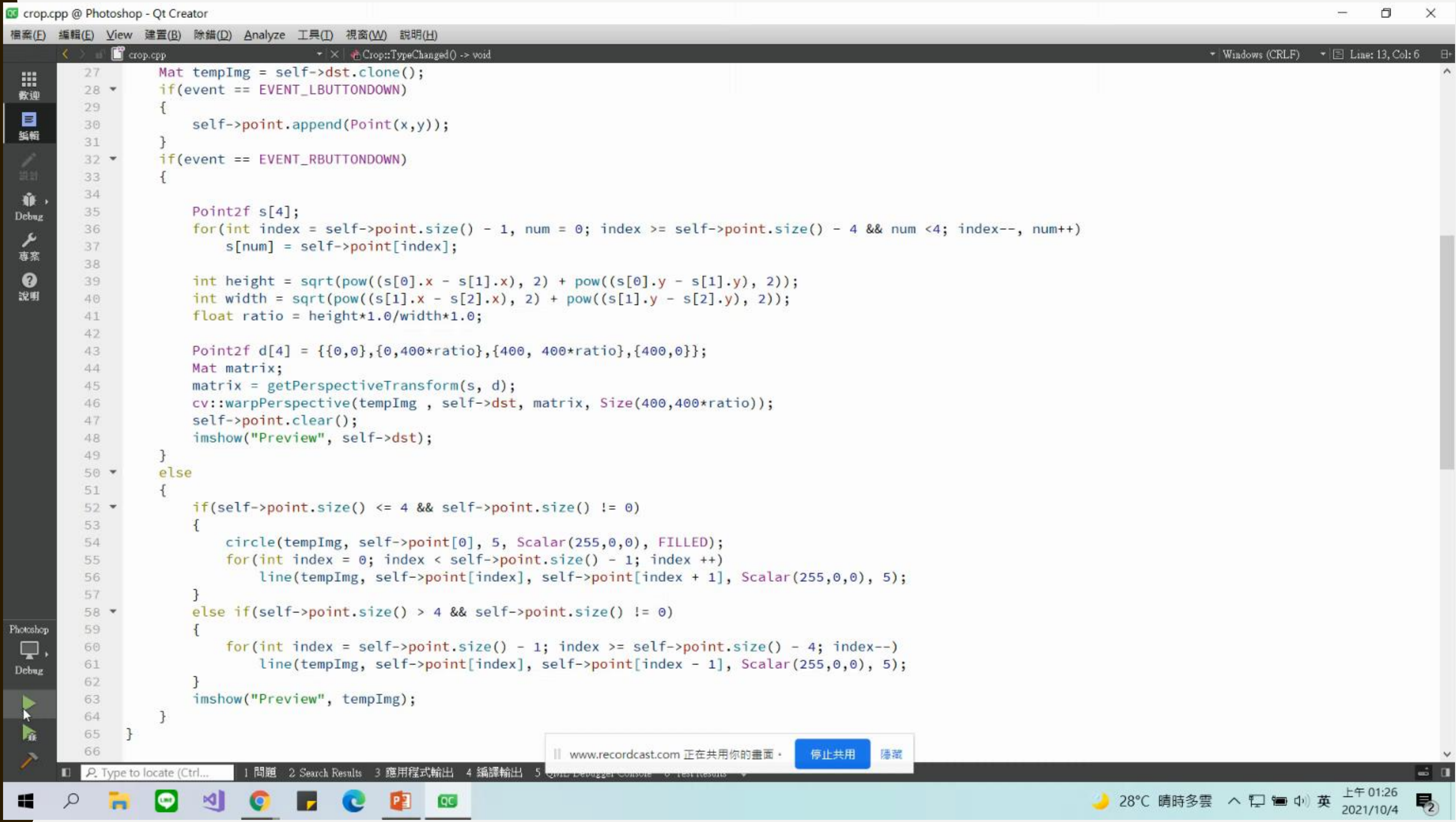




Self Select Region






crop.cpp @ Photoshop - Qt Creator
檔案(F) 編輯(E) View 建置(B) 除錯(D) Analyze 工具(T) 視窗(W) 說明(H)
crop.cpp
Crop::TypeChanged() -> void
Windows (CRLF)
Line: 13, Col: 6
歡迎
編輯
設計
Debug
專案
說明
Photoshop
Debug
Type to locate (Ctrl...
1 問題 2 Search Results 3 應用程式輸出 4 編譯輸出 5 QML Debugger Console 6 Test Results
www.recordcast.com 正在共用你的畫面。
停止共用
隱藏
28°C 晴時多雲
英
上午 01:26
2021/10/4

```cpp
    Mat tempImg = self->dst.clone();
    if(event == EVENT_LBUTTONDOWN)
    {
        self->point.append(Point(x,y));
    }
    if(event == EVENT_RBUTTONDOWN)
    {

        Point2f s[4];
        for(int index = self->point.size() - 1, num = 0; index >= self->point.size() - 4 && num <4; index--, num++)
            s[num] = self->point[index];

        int height = sqrt(pow((s[0].x - s[1].x), 2) + pow((s[0].y - s[1].y), 2));
        int width = sqrt(pow((s[1].x - s[2].x), 2) + pow((s[1].y - s[2].y), 2));
        float ratio = height*1.0/width*1.0;

        Point2f d[4] = {{0,0},{0,400*ratio},{400, 400*ratio},{400,0}};
        Mat matrix;
        matrix = getPerspectiveTransform(s, d);
        cv::warpPerspective(tempImg , self->dst, matrix, Size(400,400*ratio));
        self->point.clear();
        imshow("Preview", self->dst);
    }
    else
    {
        if(self->point.size() <= 4 && self->point.size() != 0)
        {
            circle(tempImg, self->point[0], 5, Scalar(255,0,0), FILLED);
            for(int index = 0; index < self->point.size() - 1; index ++)
                line(tempImg, self->point[index], self->point[index + 1], Scalar(255,0,0), 5);
        }
        else if(self->point.size() > 4 && self->point.size() != 0)
        {
            for(int index = self->point.size() - 1; index >= self->point.size() - 4; index--)
                line(tempImg, self->point[index], self->point[index - 1], Scalar(255,0,0), 5);
        }
        imshow("Preview", tempImg);
    }
}

```

RESIZE / ROTATE

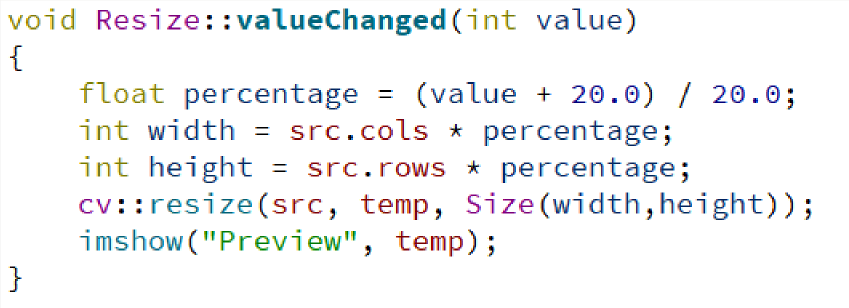
RESIZE :

The width and height of the image time multiplied by ratio respectively

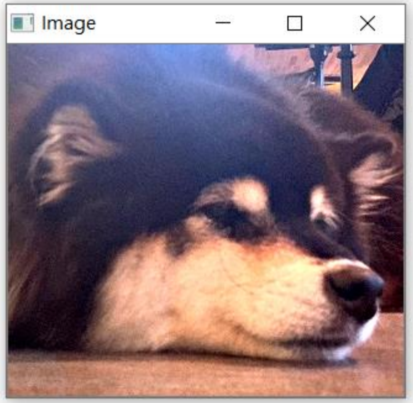
cv : : resize / Qslider

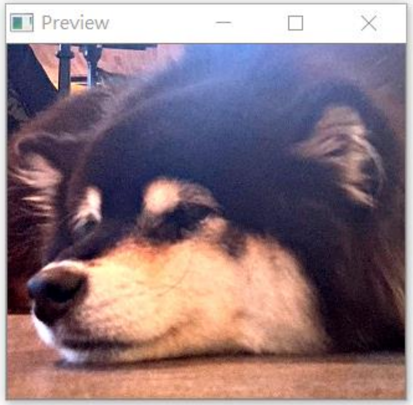
ROTATE :

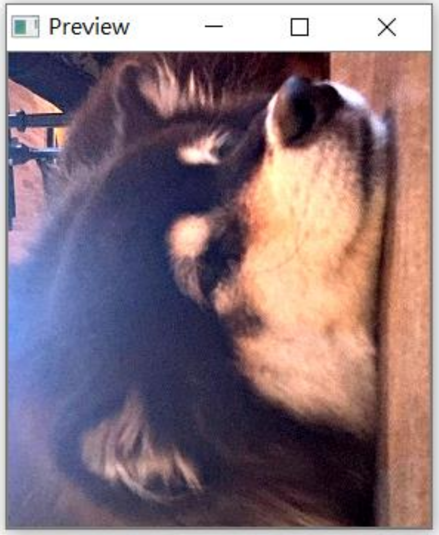
Flip / Transpose function

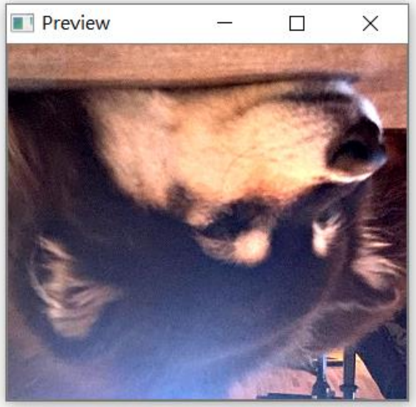


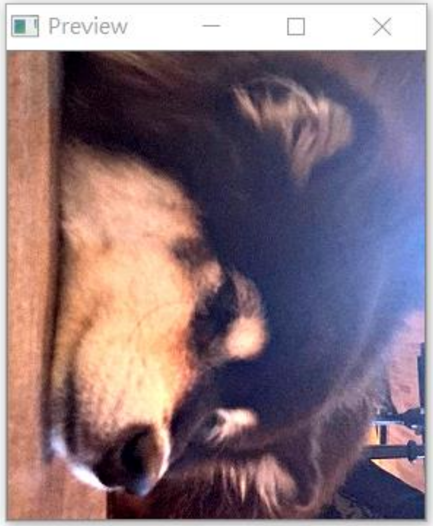
```
void Resize::valueChanged(int value)
{
    float percentage = (value + 20.0) / 20.0;
    int width = src.cols * percentage;
    int height = src.rows * percentage;
    cv::resize(src, temp, Size(width,height));
    imshow("Preview", temp);
}
```

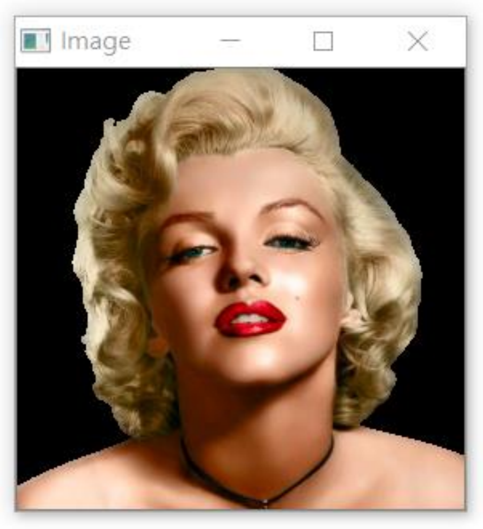


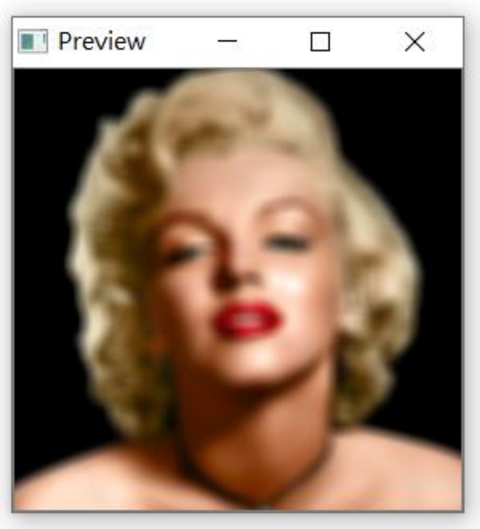


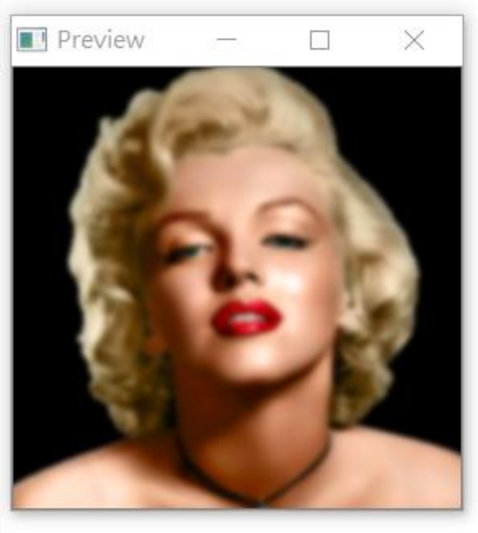


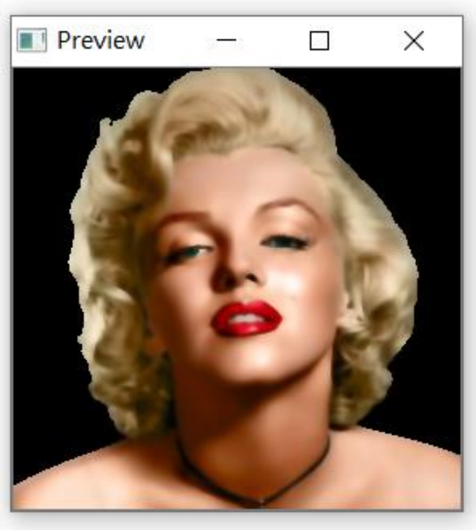


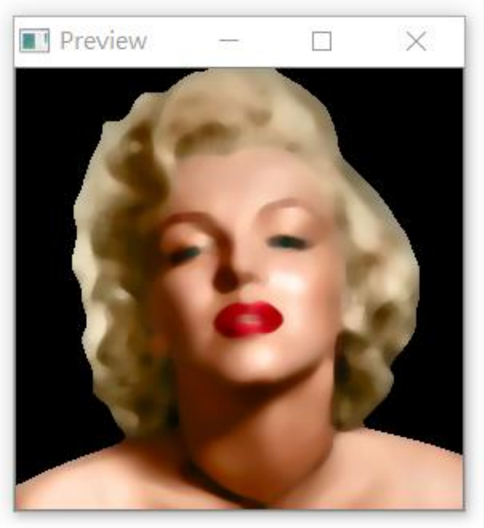












BLUR

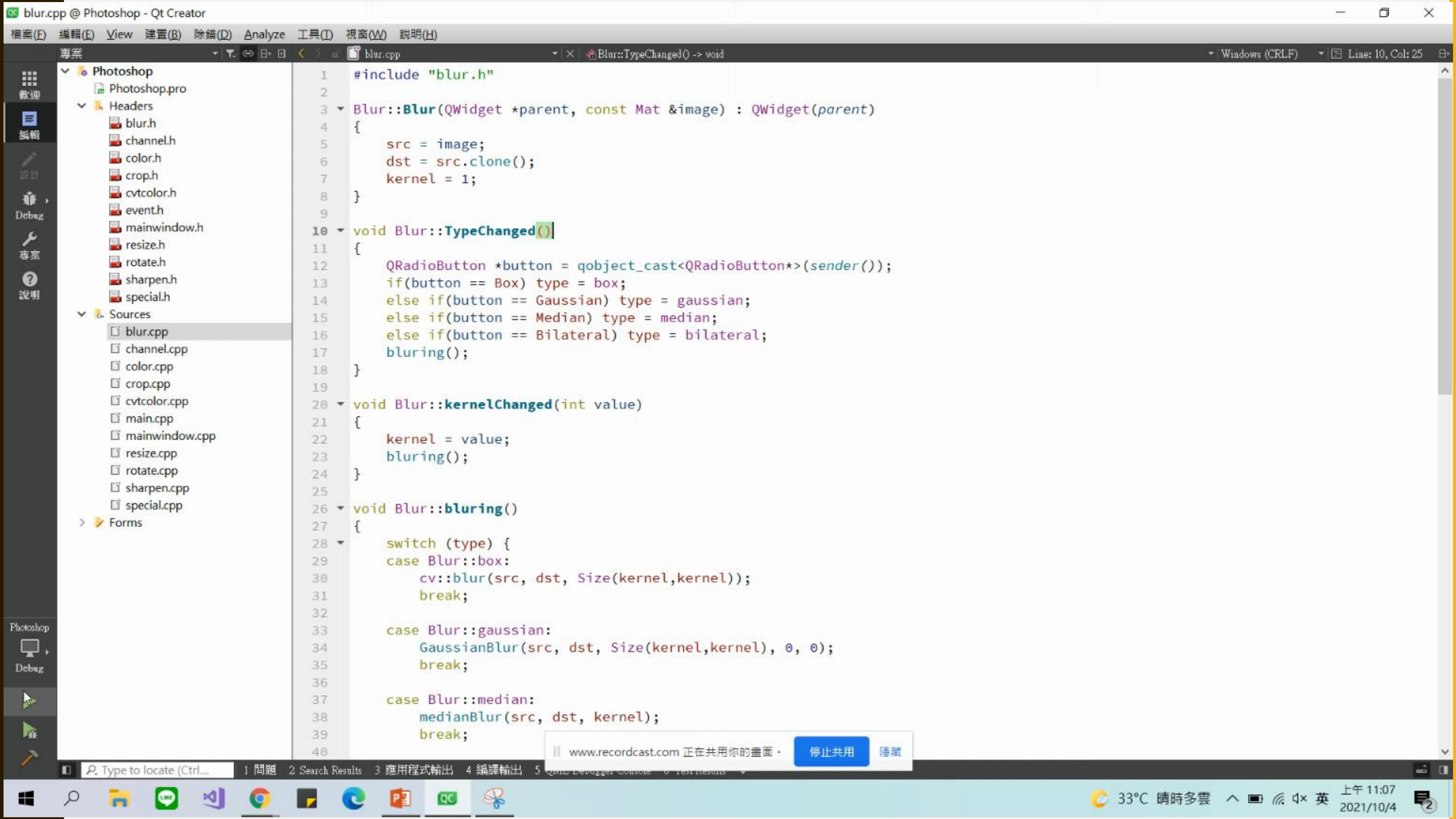
Box Filter

Gaussian Filter

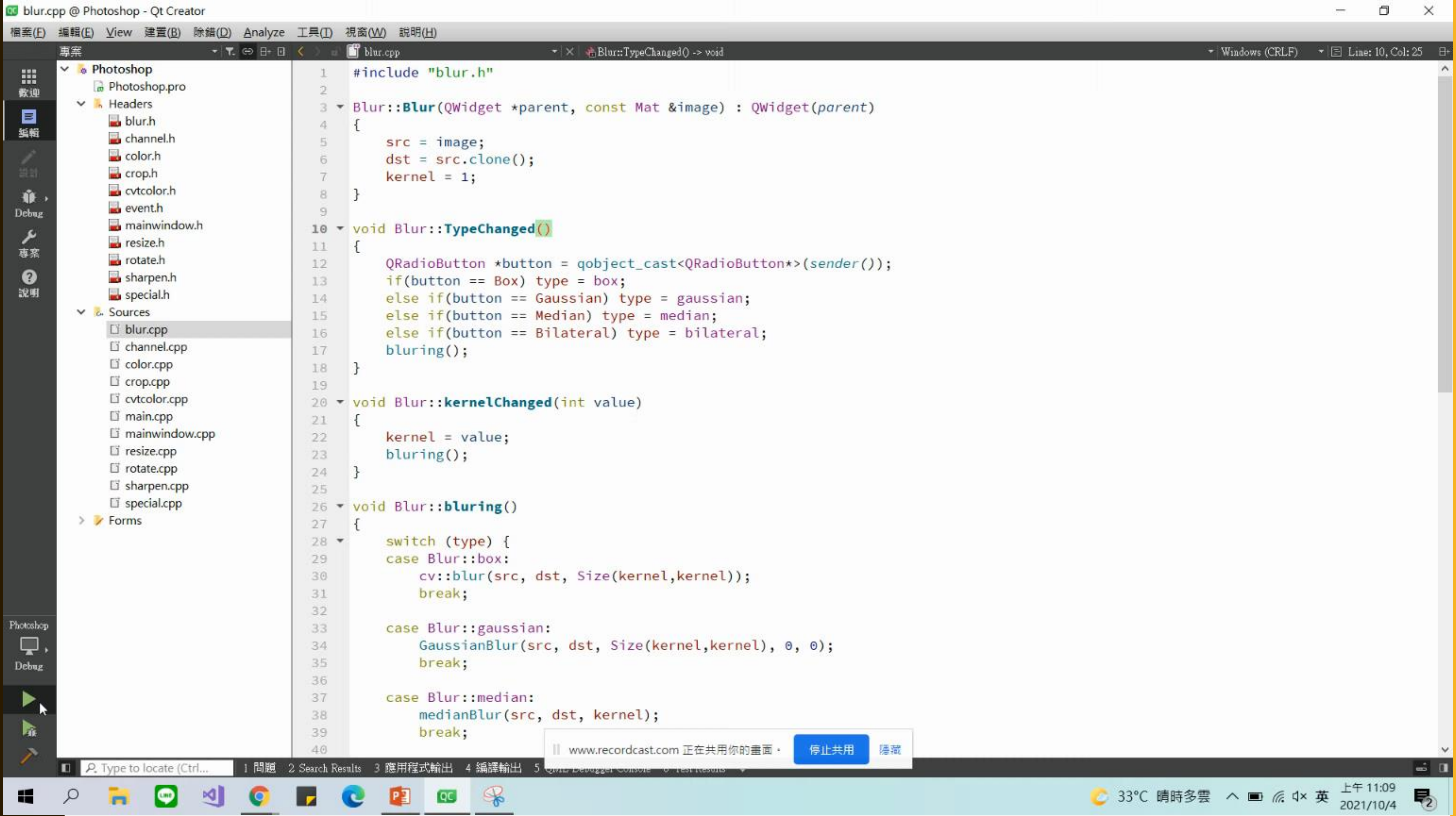
Median Filter

Bilateral Filter

Spin Box adjust the size of kernel

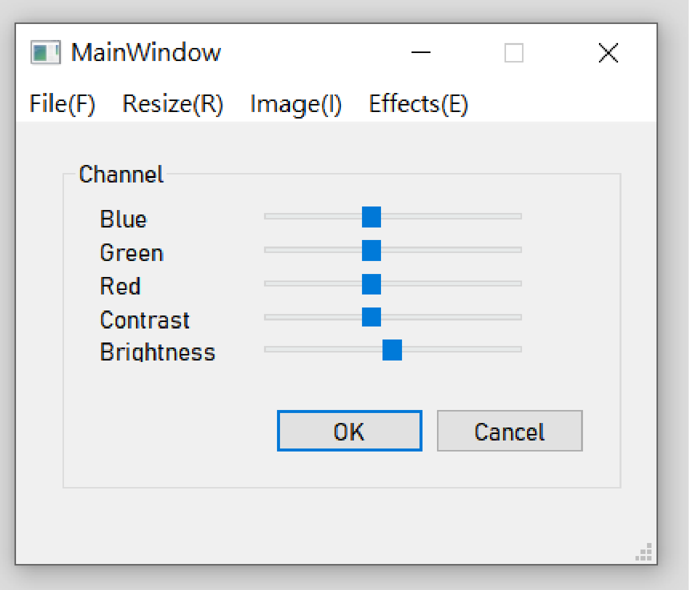


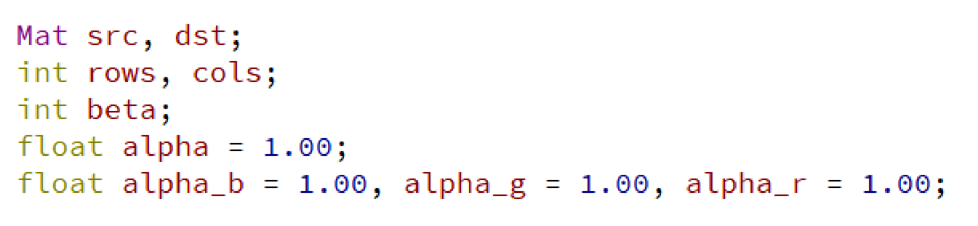
```cpp
#include "blur.h"

Blur::Blur(QWidget *parent, const Mat &image) : QWidget(parent)
{
    src = image;
    dst = src.clone();
    kernel = 1;
}

void Blur::TypeChanged()
{
    QRadioButton *button = qobject_cast<QRadioButton*>(sender());
    if(button == Box) type = box;
    else if(button == Gaussian) type = gaussian;
    else if(button == Median) type = median;
    else if(button == Bilateral) type = bilateral;
    bluring();
}

void Blur::kernelChanged(int value)
{
    kernel = value;
    bluring();
}

void Blur::bluring()
{
    switch (type) {
    case Blur::box:
        cv::blur(src, dst, Size(kernel,kernel));
        break;

    case Blur::gaussian:
        GaussianBlur(src, dst, Size(kernel,kernel), 0, 0);
        break;

    case Blur::median:
        medianBlur(src, dst, kernel);
        break;
```



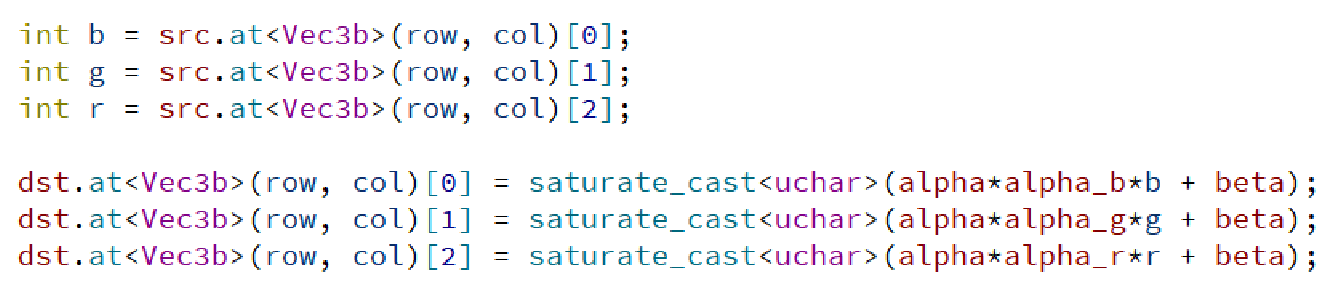
```cpp
#include "blur.h"

Blur::Blur(QWidget *parent, const Mat &image) : QWidget(parent)
{
    src = image;
    dst = src.clone();
    kernel = 1;
}

void Blur::TypeChanged()
{
    QRadioButton *button = qobject_cast<QRadioButton*>(sender());
    if(button == Box) type = box;
    else if(button == Gaussian) type = gaussian;
    else if(button == Median) type = median;
    else if(button == Bilateral) type = bilateral;
    bluring();
}

void Blur::kernelChanged(int value)
{
    kernel = value;
    bluring();
}

void Blur::bluring()
{
    switch (type) {
    case Blur::box:
        cv::blur(src, dst, Size(kernel,kernel));
        break;

    case Blur::gaussian:
        GaussianBlur(src, dst, Size(kernel,kernel), 0, 0);
        break;

    case Blur::median:
        medianBlur(src, dst, kernel);
        break;
```

CHANNEL





```
Mat src, dst;
int rows, cols;
int beta;
float alpha = 1.00;
float alpha_b = 1.00, alpha_g = 1.00, alpha_r = 1.00;
```



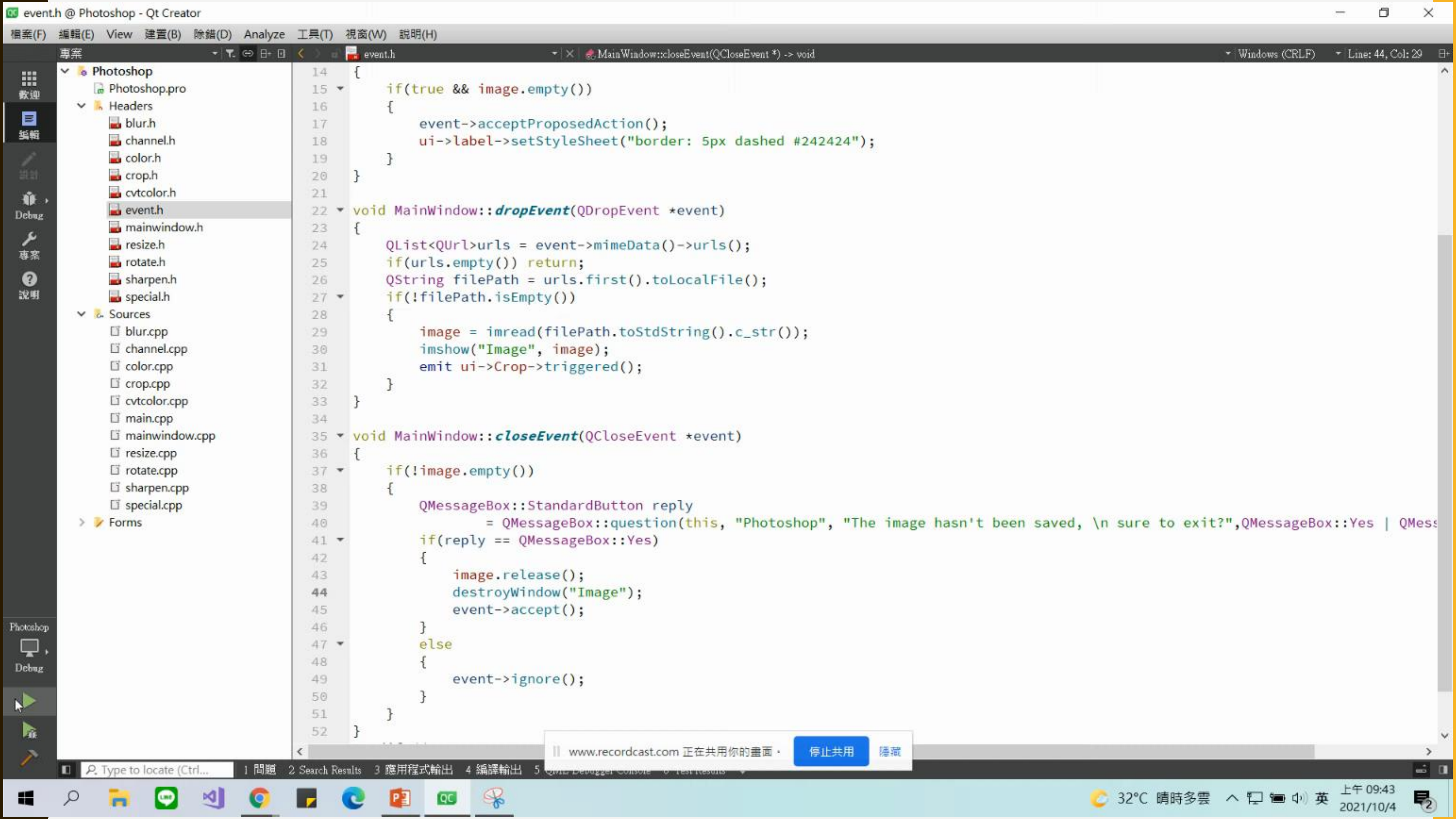
```
int b = src.at<Vec3b>(row, col)[0];
int g = src.at<Vec3b>(row, col)[1];
int r = src.at<Vec3b>(row, col)[2];

dst.at<Vec3b>(row, col)[0] = saturate_cast<uchar>(alpha*alpha_b*b + beta);
dst.at<Vec3b>(row, col)[1] = saturate_cast<uchar>(alpha*alpha_g*g + beta);
dst.at<Vec3b>(row, col)[2] = saturate_cast<uchar>(alpha*alpha_r*r + beta);
```

Set sliders to the average at beginning

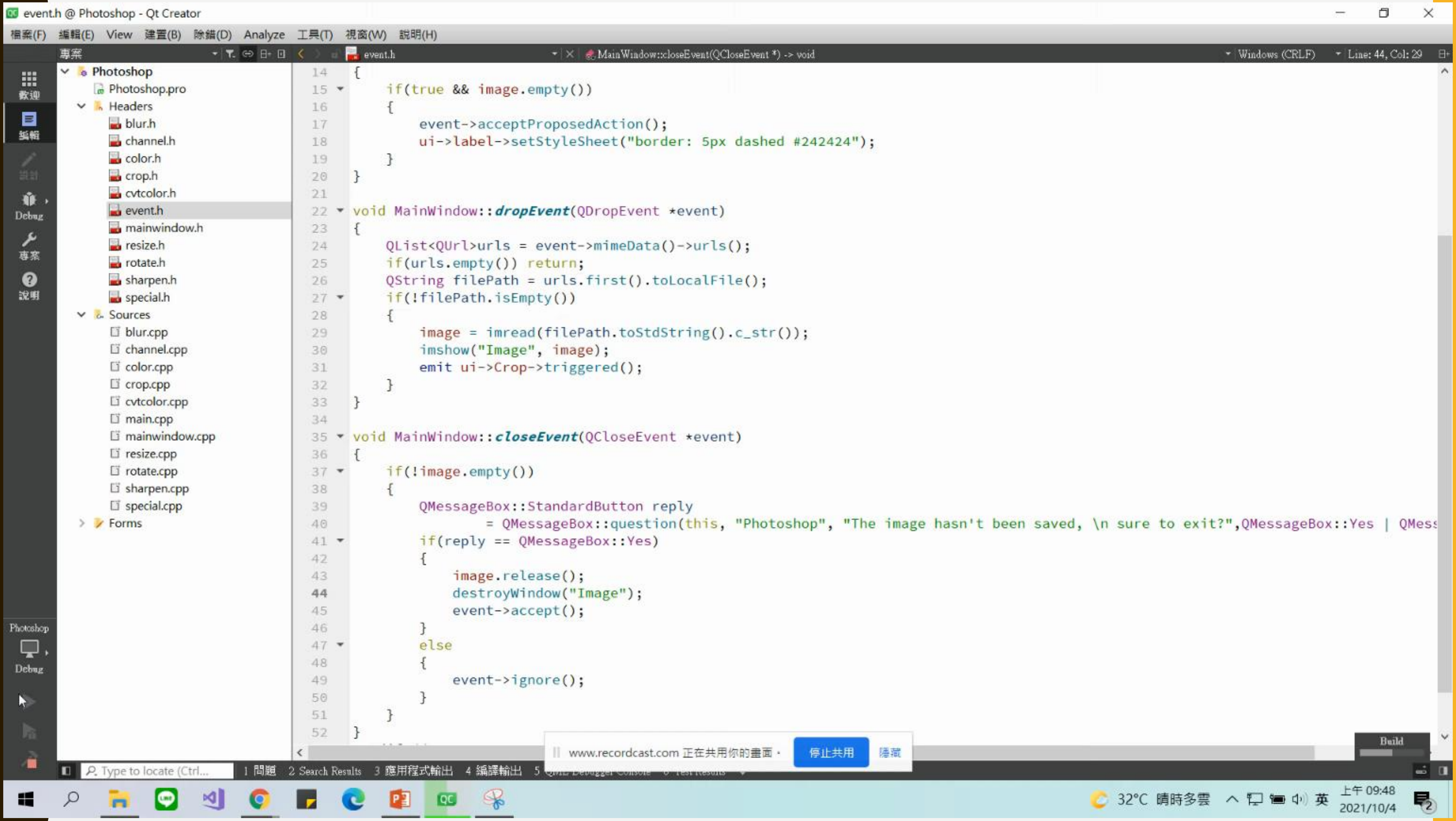
Linear Transformation :

Alpha for adjust the contrast of the image.

Beta for adjusting the brightening of the image.



```
{
    if(true && image.empty())
    {
        event->acceptProposedAction();
        ui->label->setStyleSheet("border: 5px dashed #242424");
    }
}

void MainWindow::dropEvent(QDropEvent *event)
{
    QList<QUrl>urls = event->mimeData()->urls();
    if(urls.empty()) return;
    QString filePath = urls.first().toLocalFile();
    if(!filePath.isEmpty())
    {
        image = imread(filePath.toStdString().c_str());
        imshow("Image", image);
        emit ui->Crop->triggered();
    }
}

void MainWindow::closeEvent(QCloseEvent *event)
{
    if(!image.empty())
    {
        QMessageBox::StandardButton reply
                = QMessageBox::question(this, "Photoshop", "The image hasn't been saved, \n sure to exit?",QMessageBox::Yes | QMess
        if(reply == QMessageBox::Yes)
        {
            image.release();
            destroyWindow("Image");
            event->accept();
        }
        else
        {
            event->ignore();
        }
    }
}
```



```
{
    if(true && image.empty())
    {
        event->acceptProposedAction();
        ui->label->setStyleSheet("border: 5px dashed #242424");
    }
}

void MainWindow::dropEvent(QDropEvent *event)
{
    QList<QUrl>urls = event->mimeData()->urls();
    if(urls.empty()) return;
    QString filePath = urls.first().toLocalFile();
    if(!filePath.isEmpty())
    {
        image = imread(filePath.toStdString().c_str());
        imshow("Image", image);
        emit ui->Crop->triggered();
    }
}

void MainWindow::closeEvent(QCloseEvent *event)
{
    if(!image.empty())
    {
        QMessageBox::StandardButton reply
                = QMessageBox::question(this, "Photoshop", "The image hasn't been saved, \n sure to exit?",QMessageBox::Yes | QMess
        if(reply == QMessageBox::Yes)
        {
            image.release();
            destroyWindow("Image");
            event->accept();
        }
        else
        {
            event->ignore();
        }
    }
}
```

COLOR EFFECTS

Single Color :

Red / Blue / Green

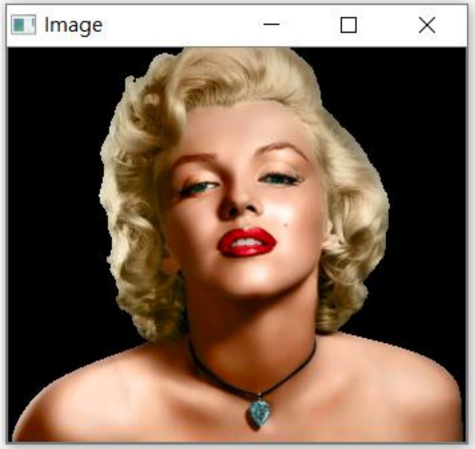
Effects :

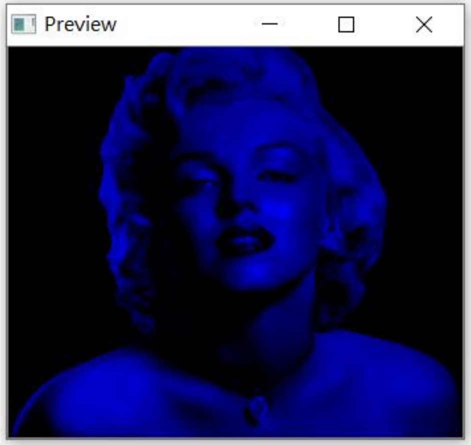
Comic Like

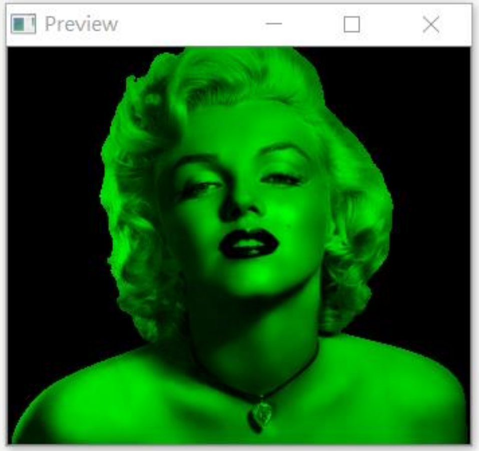
Light Blue

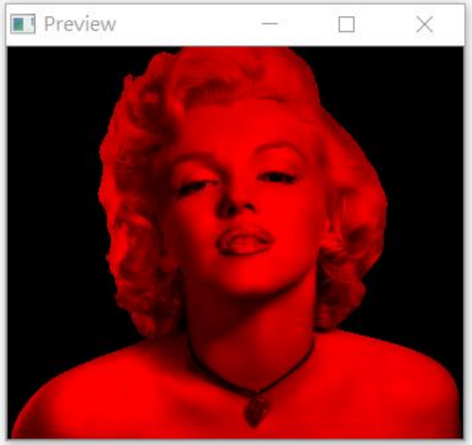
Vintage Like

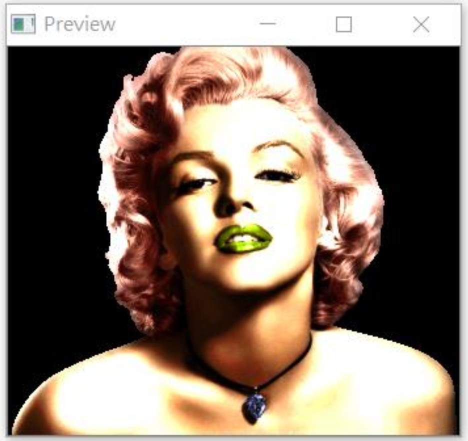
Adjust to the proper proportion of each channels

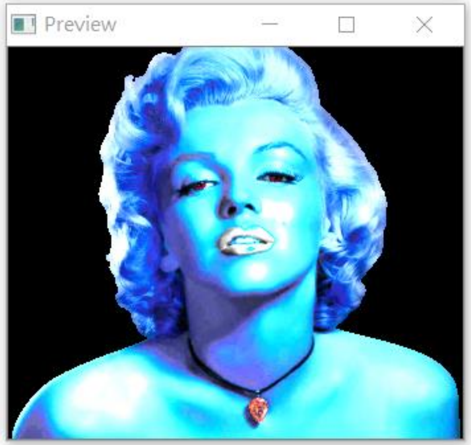


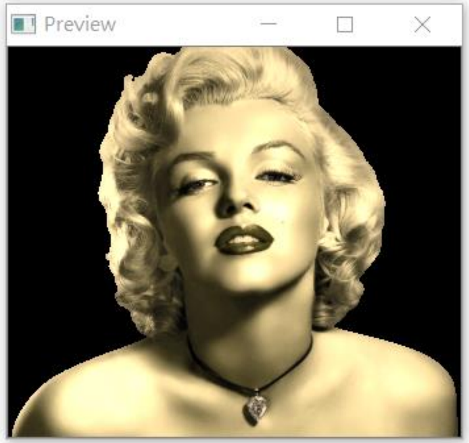


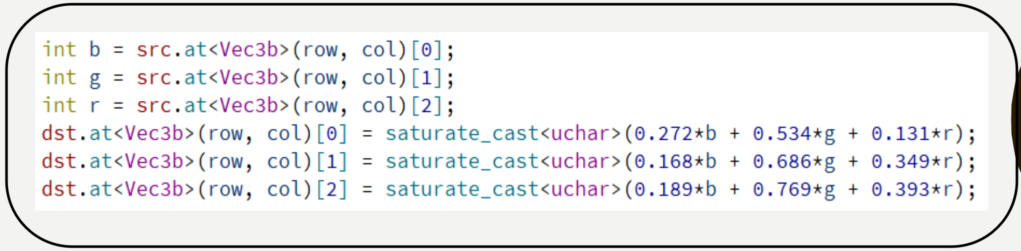












```
int b = src.at<Vec3b>(row, col)[0];
int g = src.at<Vec3b>(row, col)[1];
int r = src.at<Vec3b>(row, col)[2];
dst.at<Vec3b>(row, col)[0] = saturate_cast<uchar>(0.272*b + 0.534*g + 0.131*r);
dst.at<Vec3b>(row, col)[1] = saturate_cast<uchar>(0.168*b + 0.686*g + 0.349*r);
dst.at<Vec3b>(row, col)[2] = saturate_cast<uchar>(0.189*b + 0.769*g + 0.393*r);
```

SPECIAL EFFECTS

Leak Light Effects

Noise

Lightening

Vignetting Effect