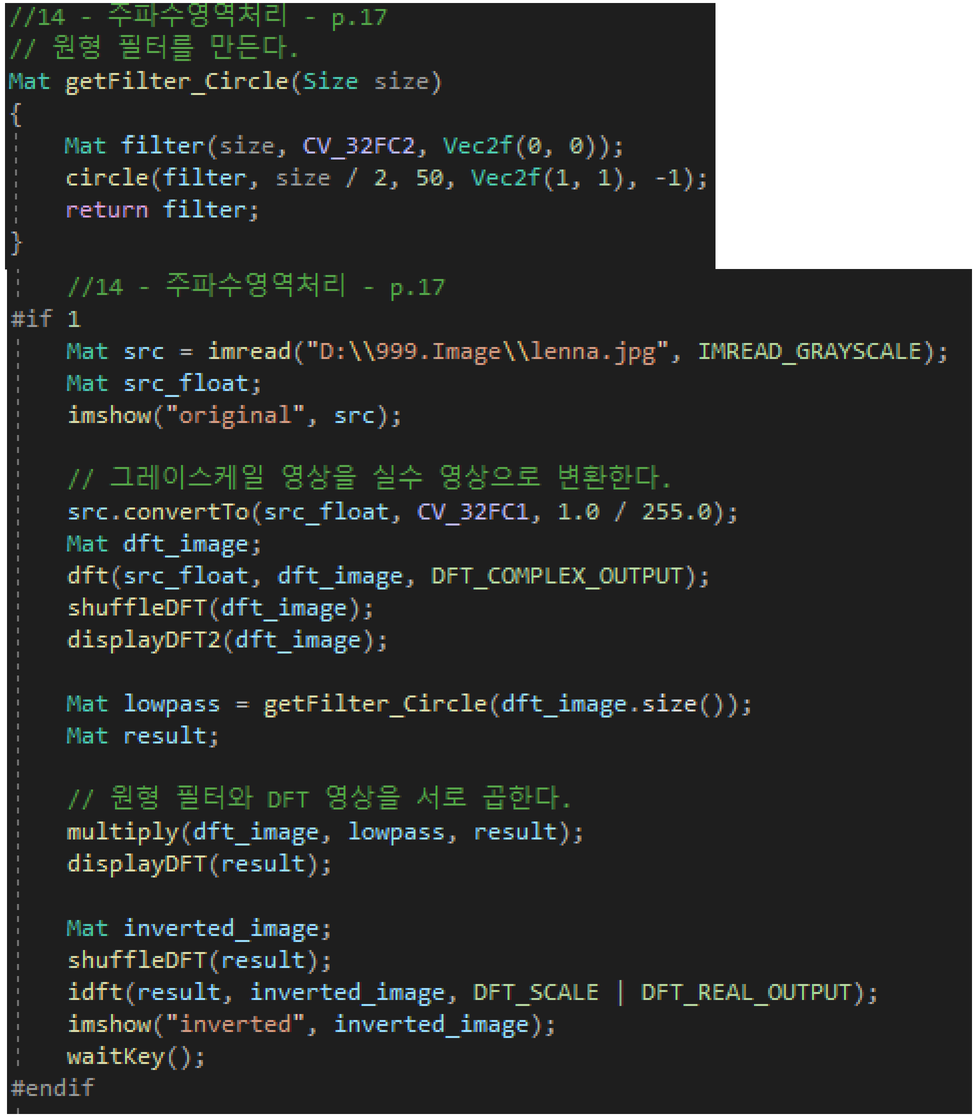
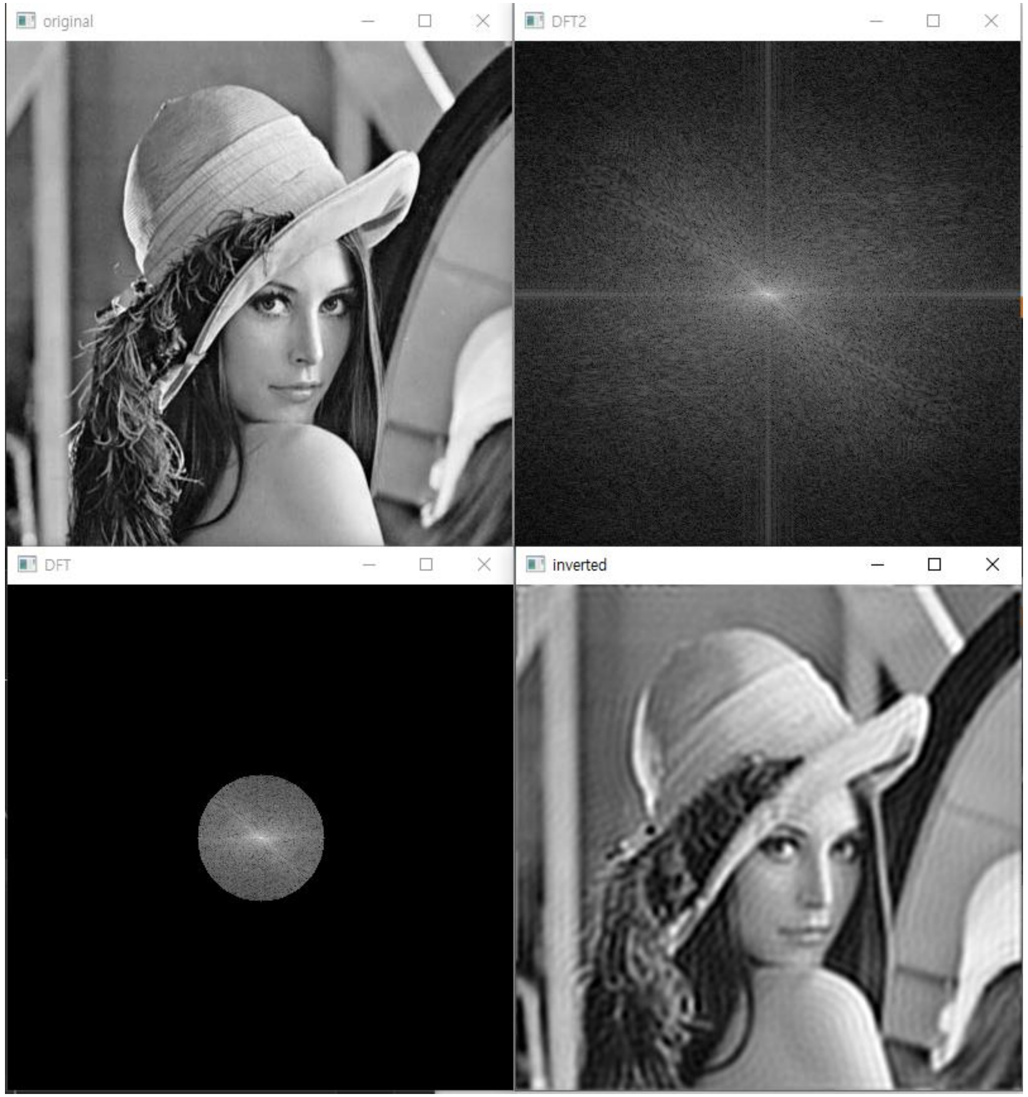
영상처리 실제 - 10주차 실습

2023254009 최현동

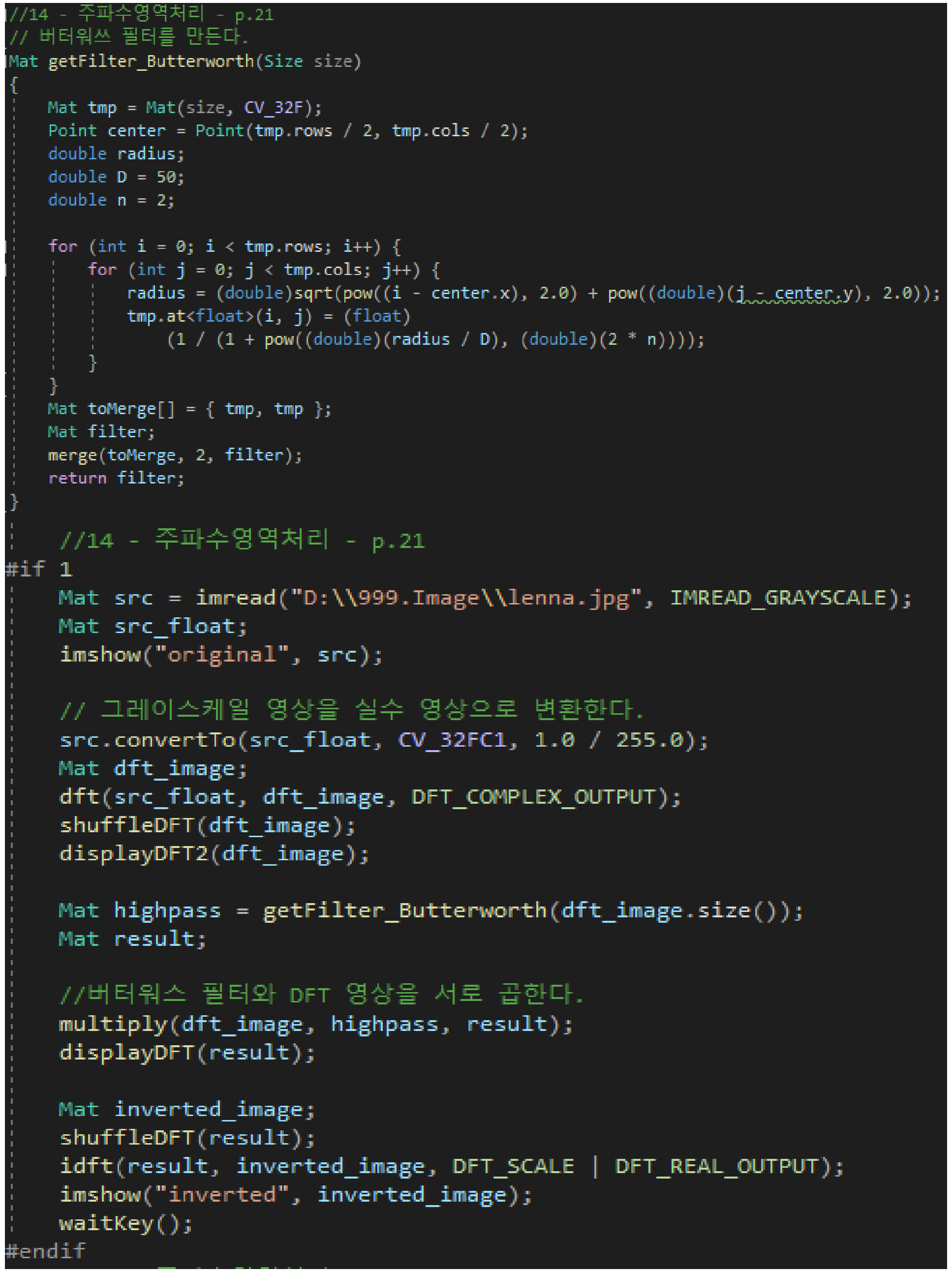
: 14.주파수영역처리 – p.17



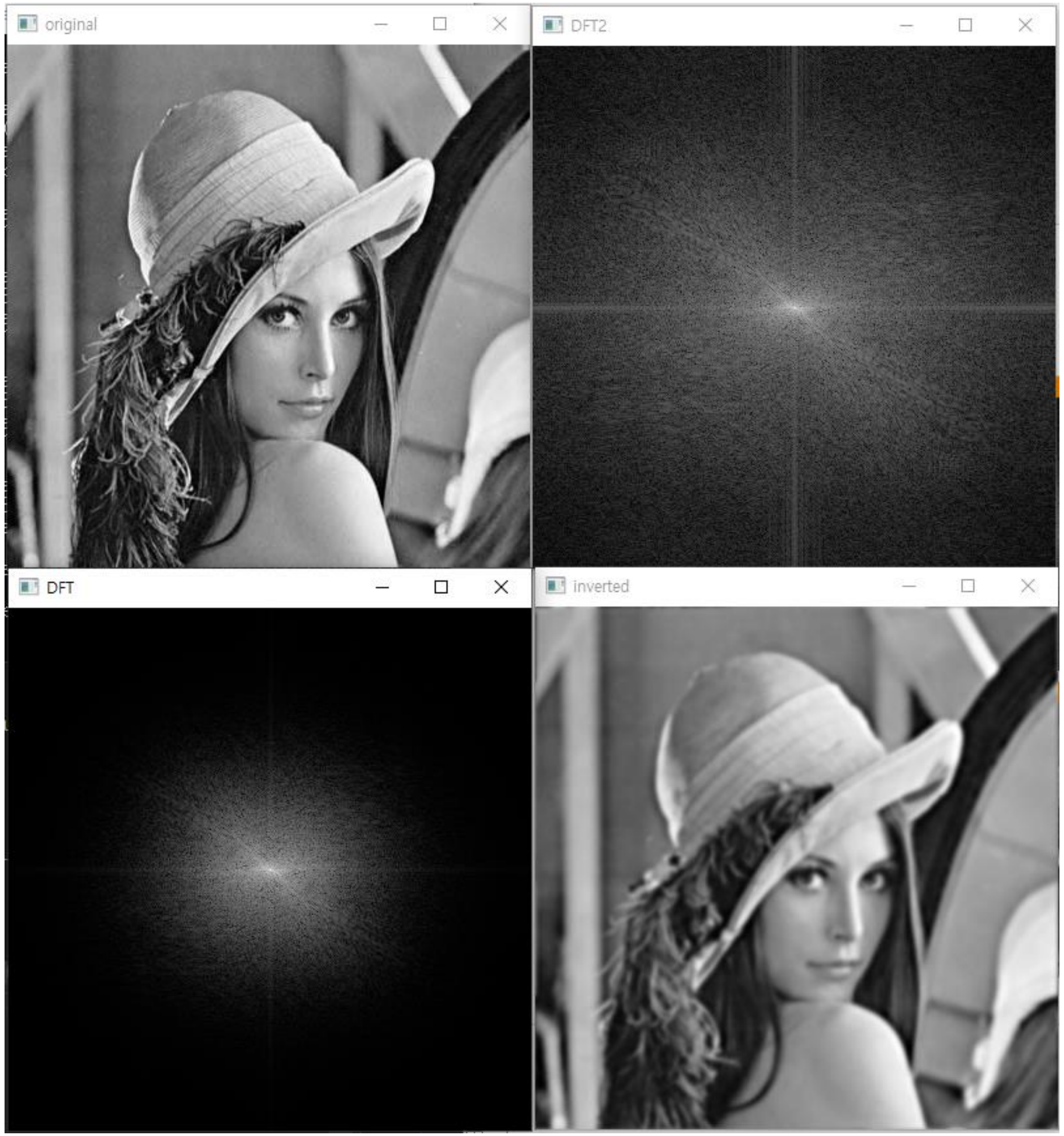
```
//14 - 주파수영역처리 - p.17
// 원형 필터를 만든다.
Mat getFilter_Circle(Size size)
{
    Mat filter(size, CV_32FC2, Vec2f(0, 0));
    circle(filter, size / 2, 50, Vec2f(1, 1), -1);
    return filter;
}
    //14 - 주파수영역처리 - p.17
#if 1
    Mat src = imread("D:\\999.Image\\lenna.jpg", IMREAD_GRAYSCALE);
    Mat src_float;
    imshow("original", src);

    // 그레이스케일 영상을 실수 영상으로 변환한다.
    src.convertTo(src_float, CV_32FC1, 1.0 / 255.0);
    Mat dft_image;
    dft(src_float, dft_image, DFT_COMPLEX_OUTPUT);
    shuffleDFT(dft_image);
    displayDFT2(dft_image);

    Mat lowpass = getFilter_Circle(dft_image.size());
    Mat result;

    // 원형 필터와 DFT 영상을 서로 곱한다.
    multiply(dft_image, lowpass, result);
    displayDFT(result);

    Mat inverted_image;
    shuffleDFT(result);
    idft(result, inverted_image, DFT_SCALE | DFT_REAL_OUTPUT);
    imshow("inverted", inverted_image);
    waitKey();
#endif
```



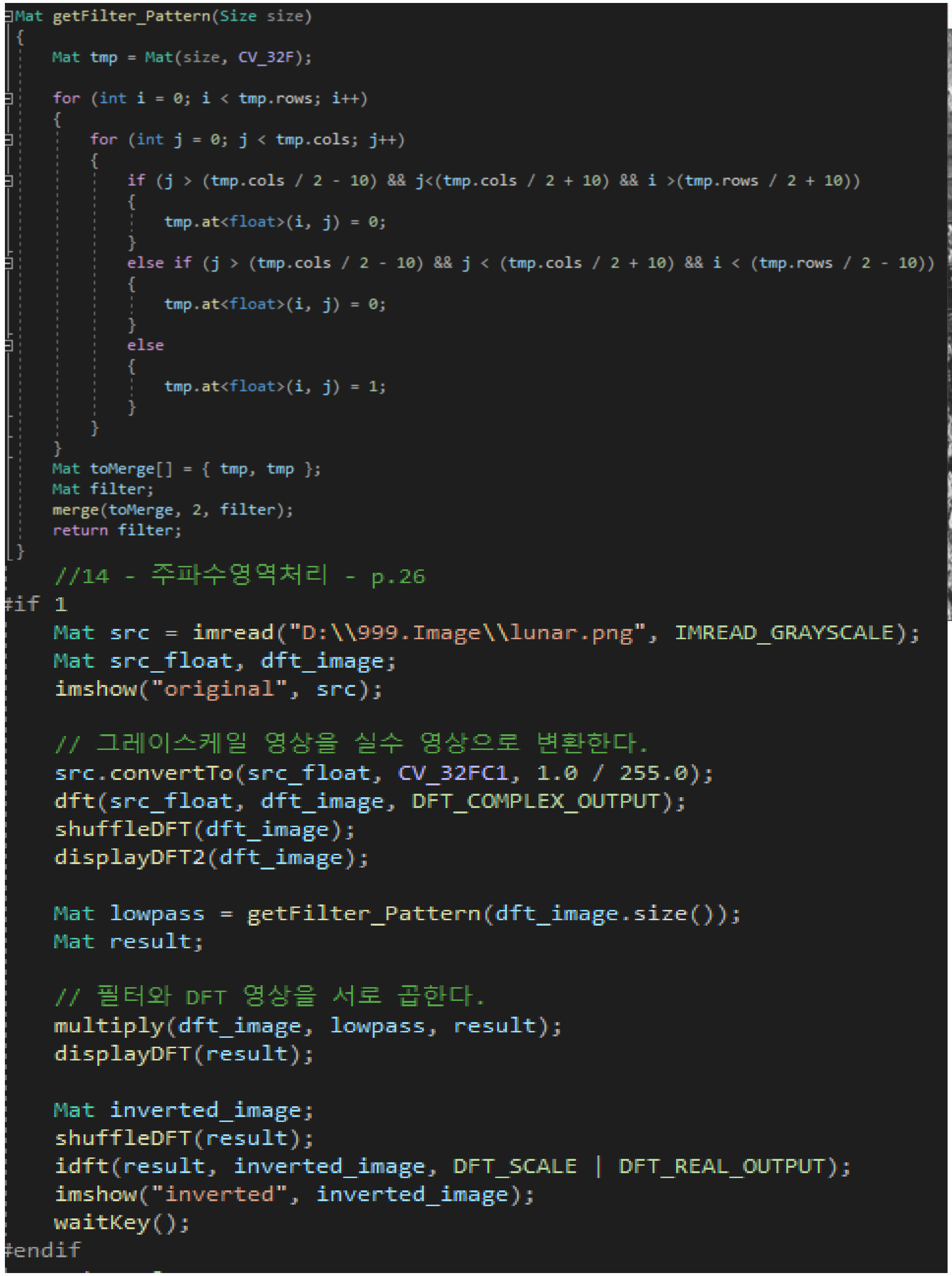
: 14.주파수영역처리 – p.21



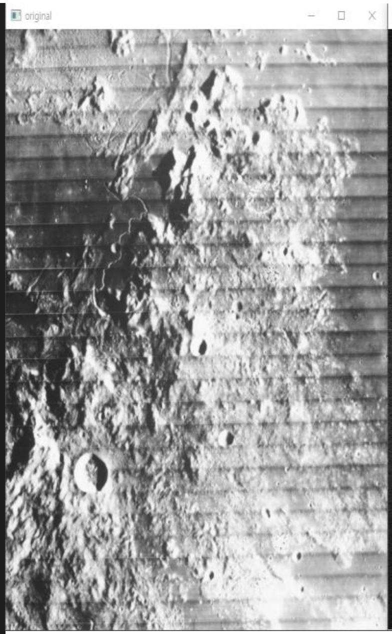
```
//14 - 주파수영역처리 - p.21
// 버터워쓰 필터를 만든다.
Mat getFilter_Butterworth(Size size)
{
    Mat tmp = Mat(size, CV_32F);
    Point center = Point(tmp.rows / 2, tmp.cols / 2);
    double radius;
    double D = 50;
    double n = 2;

    for (int i = 0; i < tmp.rows; i++) {
        for (int j = 0; j < tmp.cols; j++) {
            radius = (double)sqrt(pow((i - center.x), 2.0) + pow((double)(j - center.y), 2.0));
            tmp.at<float>(i, j) = (float)
                (1 / (1 + pow((double)(radius / D), (double)(2 * n))));
        }
    }
    Mat toMerge[] = { tmp, tmp };
    Mat filter;
    merge(toMerge, 2, filter);
    return filter;
}
    //14 - 주파수영역처리 - p.21
#if 1
    Mat src = imread("D:\\999.Image\\lenna.jpg", IMREAD_GRAYSCALE);
    Mat src_float;
    imshow("original", src);

    // 그레이스케일 영상을 실수 영상으로 변환한다.
    src.convertTo(src_float, CV_32FC1, 1.0 / 255.0);
    Mat dft_image;
    dft(src_float, dft_image, DFT_COMPLEX_OUTPUT);
    shuffleDFT(dft_image);
    displayDFT2(dft_image);

    Mat highpass = getFilter_Butterworth(dft_image.size());
    Mat result;

    //버터워스 필터와 DFT 영상을 서로 곱한다.
    multiply(dft_image, highpass, result);
    displayDFT(result);

    Mat inverted_image;
    shuffleDFT(result);
    idft(result, inverted_image, DFT_SCALE | DFT_REAL_OUTPUT);
    imshow("inverted", inverted_image);
    waitKey();
#endif
```

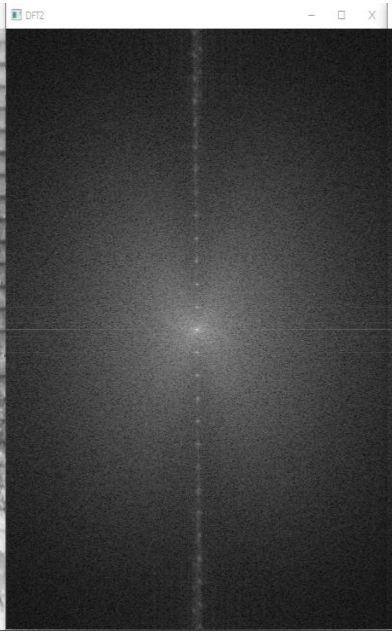


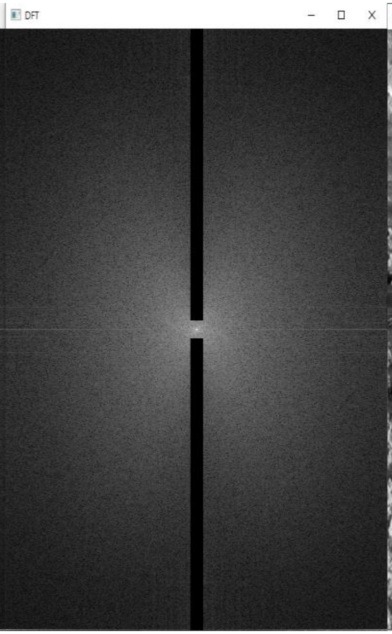
: 14.주파수영역처리 – p.26

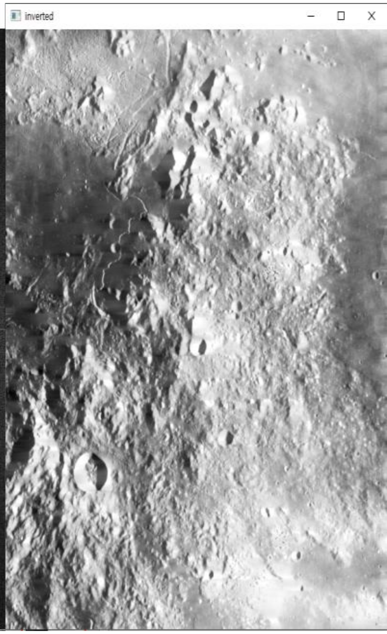


```
Mat getFilter_Pattern(Size size)
{
    Mat tmp = Mat(size, CV_32F);

    for (int i = 0; i < tmp.rows; i++)
    {
        for (int j = 0; j < tmp.cols; j++)
        {
            if (j > (tmp.cols / 2 - 10) && j<(tmp.cols / 2 + 10) && i >(tmp.rows / 2 + 10))
            {
                tmp.at<float>(i, j) = 0;
            }
            else if (j > (tmp.cols / 2 - 10) && j < (tmp.cols / 2 + 10) && i < (tmp.rows / 2 - 10))
            {
                tmp.at<float>(i, j) = 0;
            }
            else
            {
                tmp.at<float>(i, j) = 1;
            }
        }
    }
    Mat toMerge[] = { tmp, tmp };
    Mat filter;
    merge(toMerge, 2, filter);
    return filter;
}
    //14 - 주파수영역처리 - p.26
#if 1
    Mat src = imread("D:\\999.Image\\lunar.png", IMREAD_GRAYSCALE);
    Mat src_float, dft_image;
    imshow("original", src);

    // 그레이스케일 영상을 실수 영상으로 변환한다.
    src.convertTo(src_float, CV_32FC1, 1.0 / 255.0);
    dft(src_float, dft_image, DFT_COMPLEX_OUTPUT);
    shuffleDFT(dft_image);
    displayDFT2(dft_image);

    Mat lowpass = getFilter_Pattern(dft_image.size());
    Mat result;

    // 필터와 DFT 영상을 서로 곱한다.
    multiply(dft_image, lowpass, result);
    displayDFT(result);

    Mat inverted_image;
    shuffleDFT(result);
    idft(result, inverted_image, DFT_SCALE | DFT_REAL_OUTPUT);
    imshow("inverted", inverted_image);
    waitKey();
#endif
```

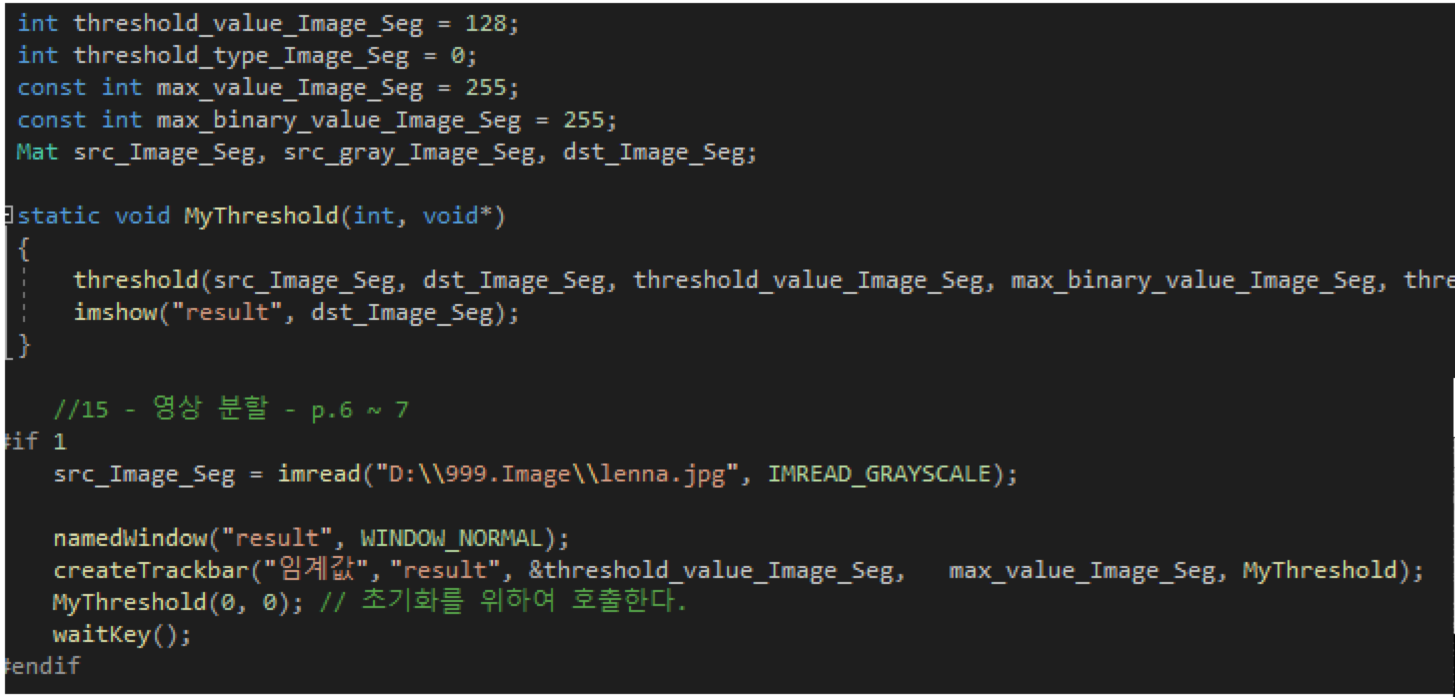




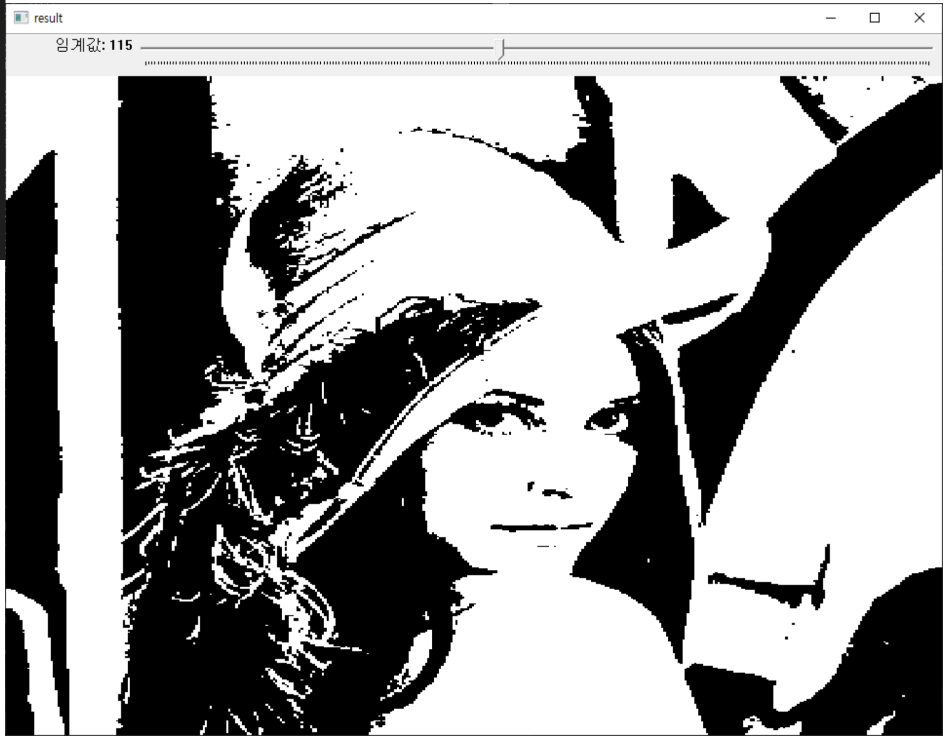




: 15.영상분할 – p.6 ~ 7

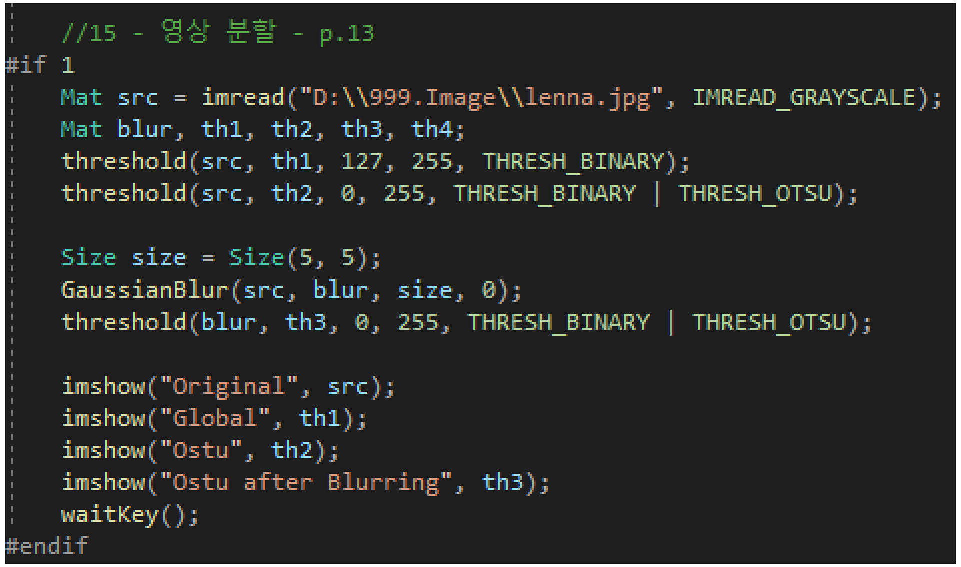


```
int threshold_value_Image_Seg = 128;
int threshold_type_Image_Seg = 0;
const int max_value_Image_Seg = 255;
const int max_binary_value_Image_Seg = 255;
Mat src_Image_Seg, src_gray_Image_Seg, dst_Image_Seg;

static void MyThreshold(int, void*)
{
    threshold(src_Image_Seg, dst_Image_Seg, threshold_value_Image_Seg, max_binary_value_Image_Seg, threshold_type_Image_Seg);
    imshow("result", dst_Image_Seg);
}

   //15 - 영상 분할 - p.6 ~ 7
#if 1
   src_Image_Seg = imread("D:\\999.Image\\lenna.jpg", IMREAD_GRAYSCALE);

   namedWindow("result", WINDOW_NORMAL);
   createTrackbar("임계값","result", &threshold_value_Image_Seg,   max_value_Image_Seg, MyThreshold);
   MyThreshold(0, 0); // 초기화를 위하여 호출한다.
   waitKey();
#endif
```

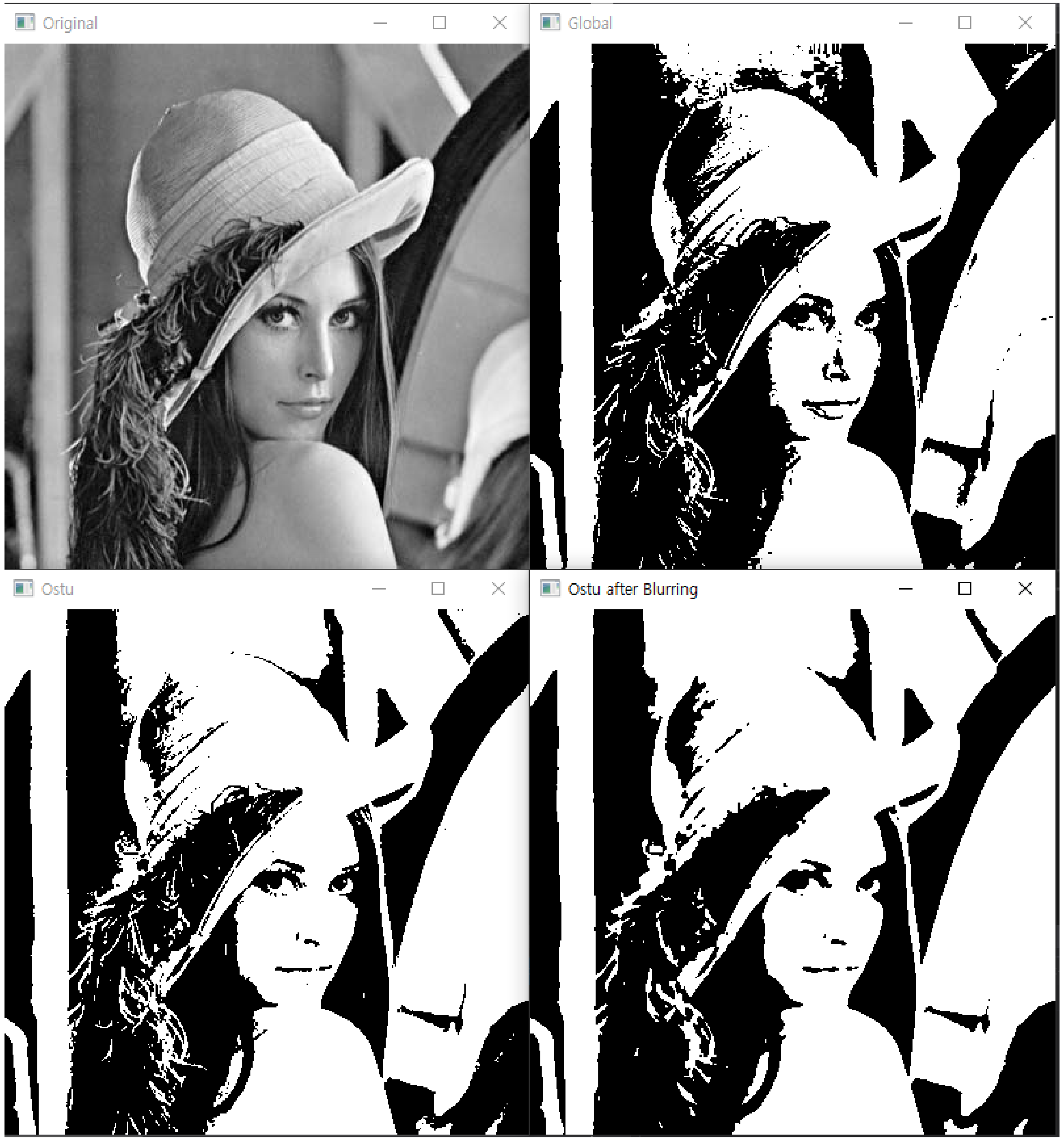


임계값: 115

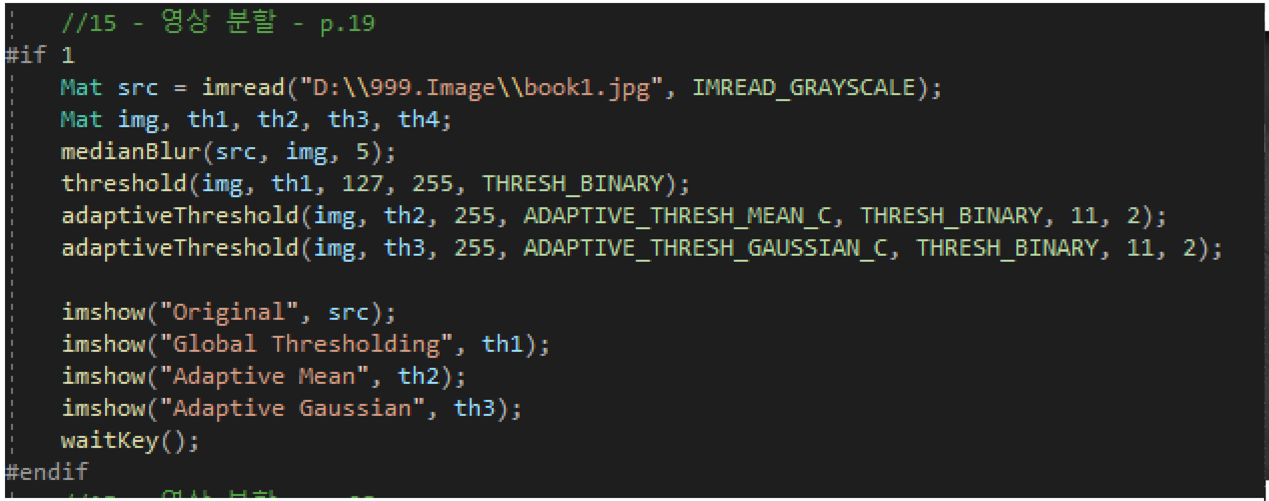
: 15.영상분할 – p.13



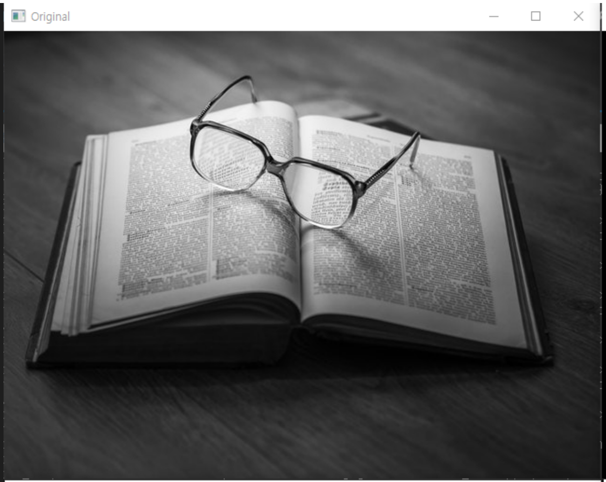
```
    //15 - 영상 분할 - p.13
#if 1
    Mat src = imread("D:\\999.Image\\lenna.jpg", IMREAD_GRAYSCALE);
    Mat blur, th1, th2, th3, th4;
    threshold(src, th1, 127, 255, THRESH_BINARY);
    threshold(src, th2, 0, 255, THRESH_BINARY | THRESH_OTSU);

    Size size = Size(5, 5);
    GaussianBlur(src, blur, size, 0);
    threshold(blur, th3, 0, 255, THRESH_BINARY | THRESH_OTSU);

    imshow("Original", src);
    imshow("Global", th1);
    imshow("Ostu", th2);
    imshow("Ostu after Blurring", th3);
    waitKey();
#endif
```

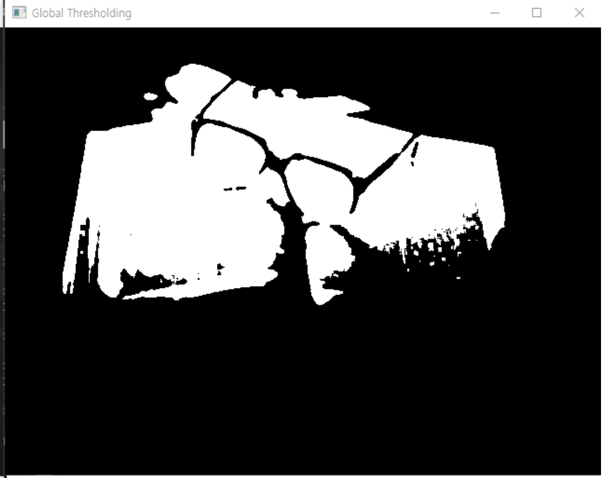


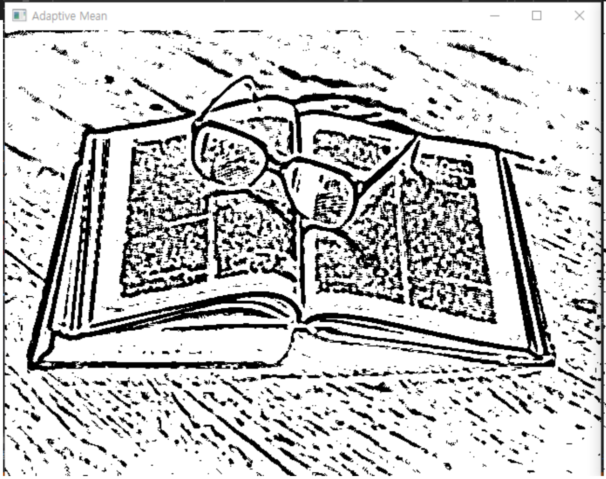
: 15.영상분할 – p.19

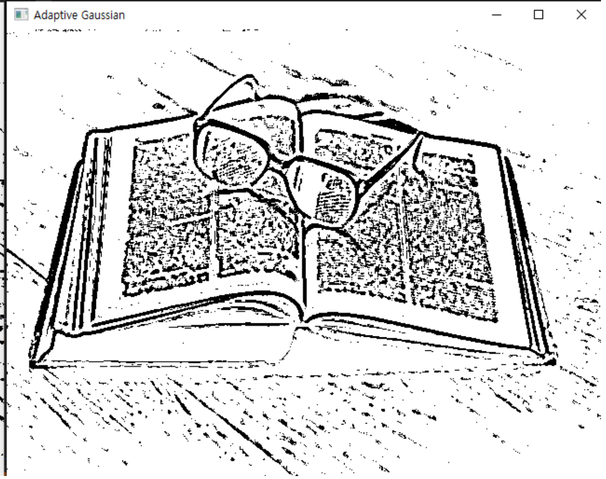


```
    //15 - 영상 분할 - p.19
#if 1
    Mat src = imread("D:\\999.Image\\book1.jpg", IMREAD_GRAYSCALE);
    Mat img, th1, th2, th3, th4;
    medianBlur(src, img, 5);
    threshold(img, th1, 127, 255, THRESH_BINARY);
    adaptiveThreshold(img, th2, 255, ADAPTIVE_THRESH_MEAN_C, THRESH_BINARY, 11, 2);
    adaptiveThreshold(img, th3, 255, ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY, 11, 2);

    imshow("Original", src);
    imshow("Global Thresholding", th1);
    imshow("Adaptive Mean", th2);
    imshow("Adaptive Gaussian", th3);
    waitKey();
#endif
```

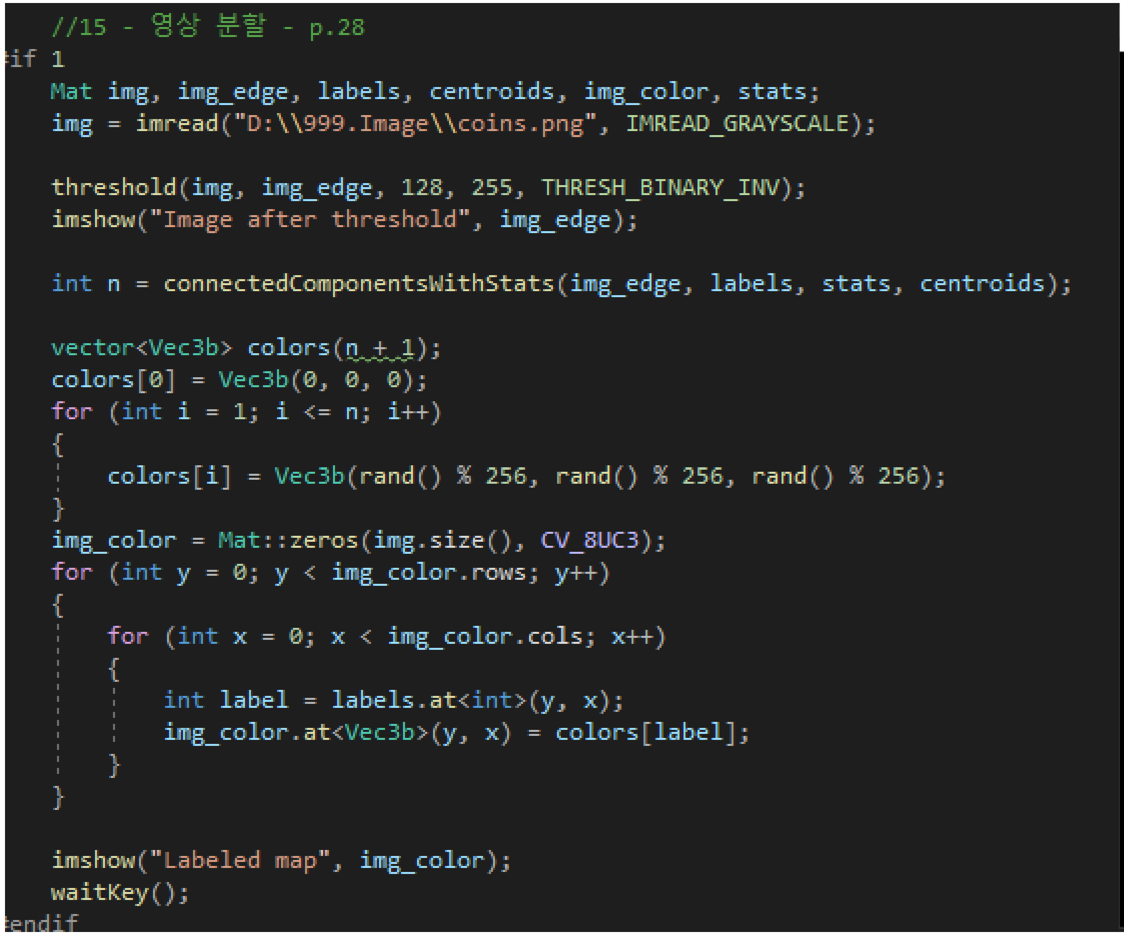




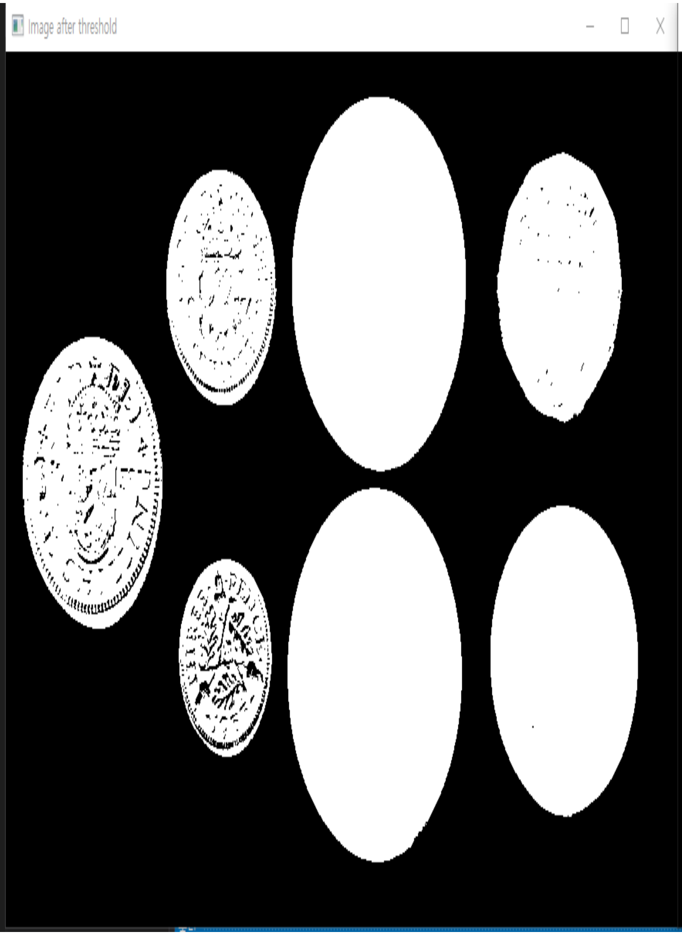


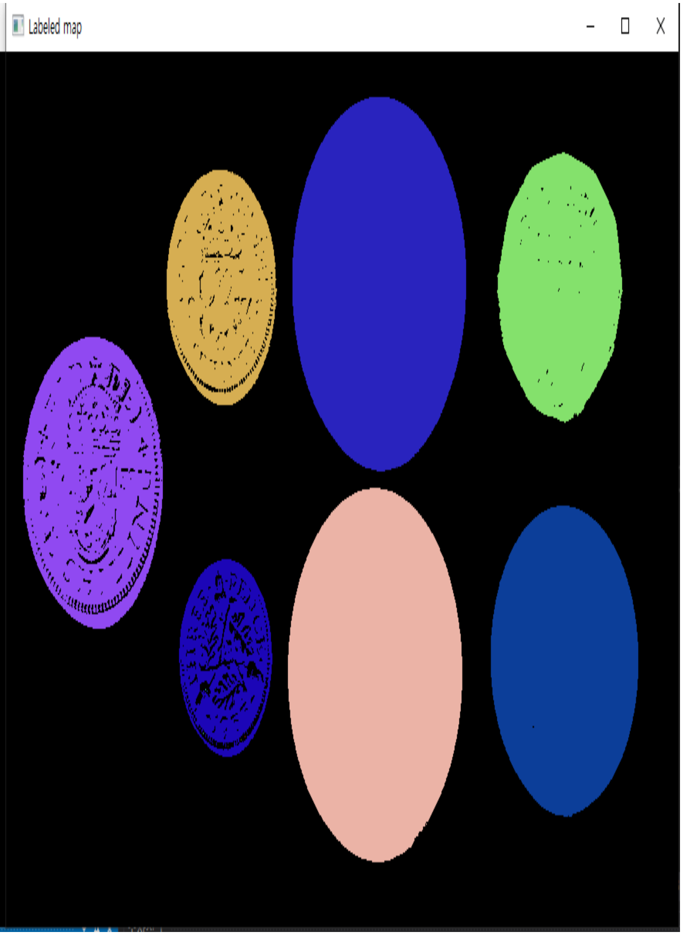


: 15.영상분할 – p.28

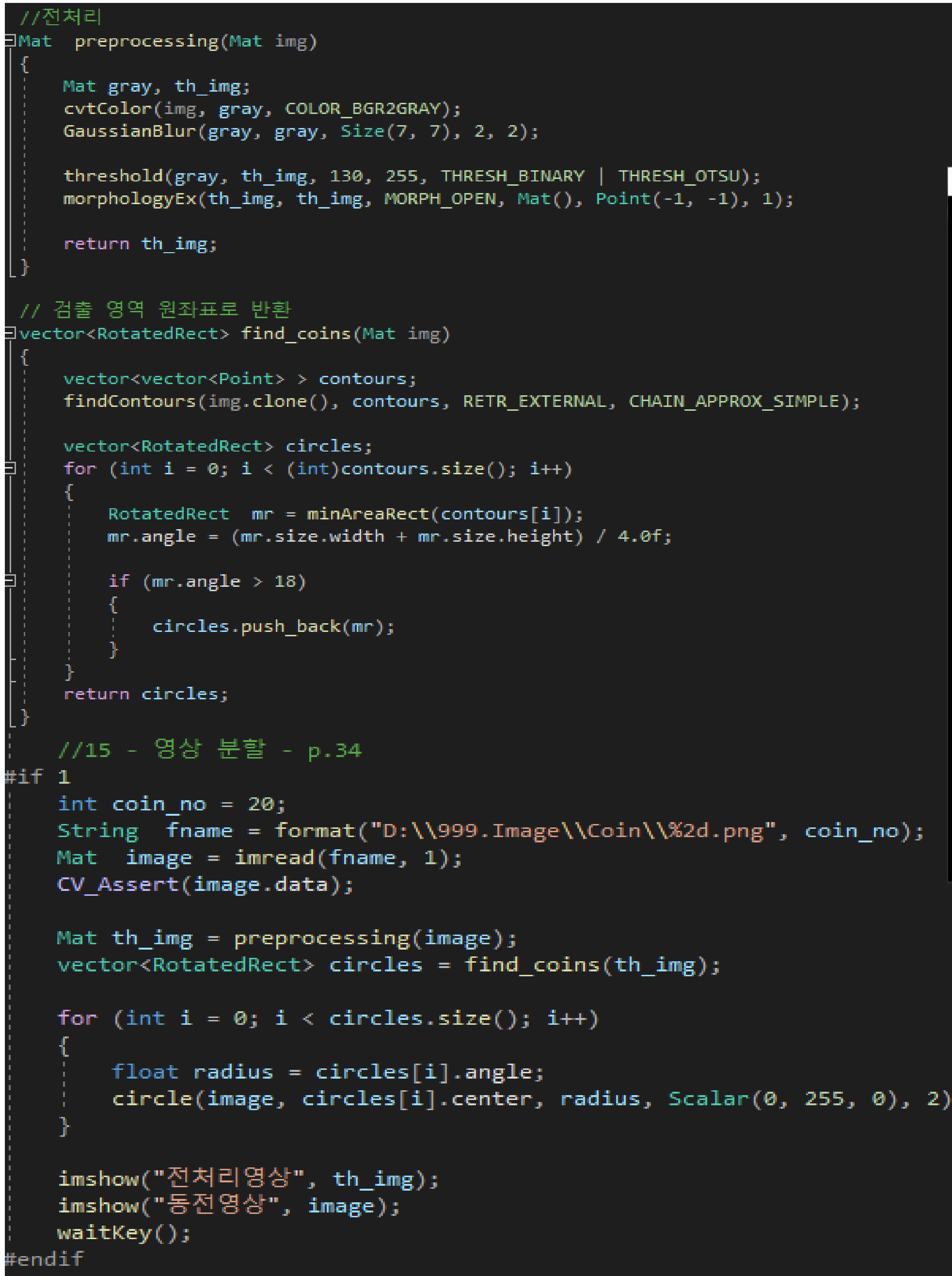


```
    //15 - 영상 분할 - p.28
#if 1
    Mat img, img_edge, labels, centroids, img_color, stats;
    img = imread("D:\\999.Image\\coins.png", IMREAD_GRAYSCALE);

    threshold(img, img_edge, 128, 255, THRESH_BINARY_INV);
    imshow("Image after threshold", img_edge);

    int n = connectedComponentsWithStats(img_edge, labels, stats, centroids);

    vector<Vec3b> colors(n + 1);
    colors[0] = Vec3b(0, 0, 0);
    for (int i = 1; i <= n; i++)
    {
        colors[i] = Vec3b(rand() % 256, rand() % 256, rand() % 256);
    }
    img_color = Mat::zeros(img.size(), CV_8UC3);
    for (int y = 0; y < img_color.rows; y++)
    {
        for (int x = 0; x < img_color.cols; x++)
        {
            int label = labels.at<int>(y, x);
            img_color.at<Vec3b>(y, x) = colors[label];
        }
    }

    imshow("Labeled map", img_color);
    waitKey();
#endif
```

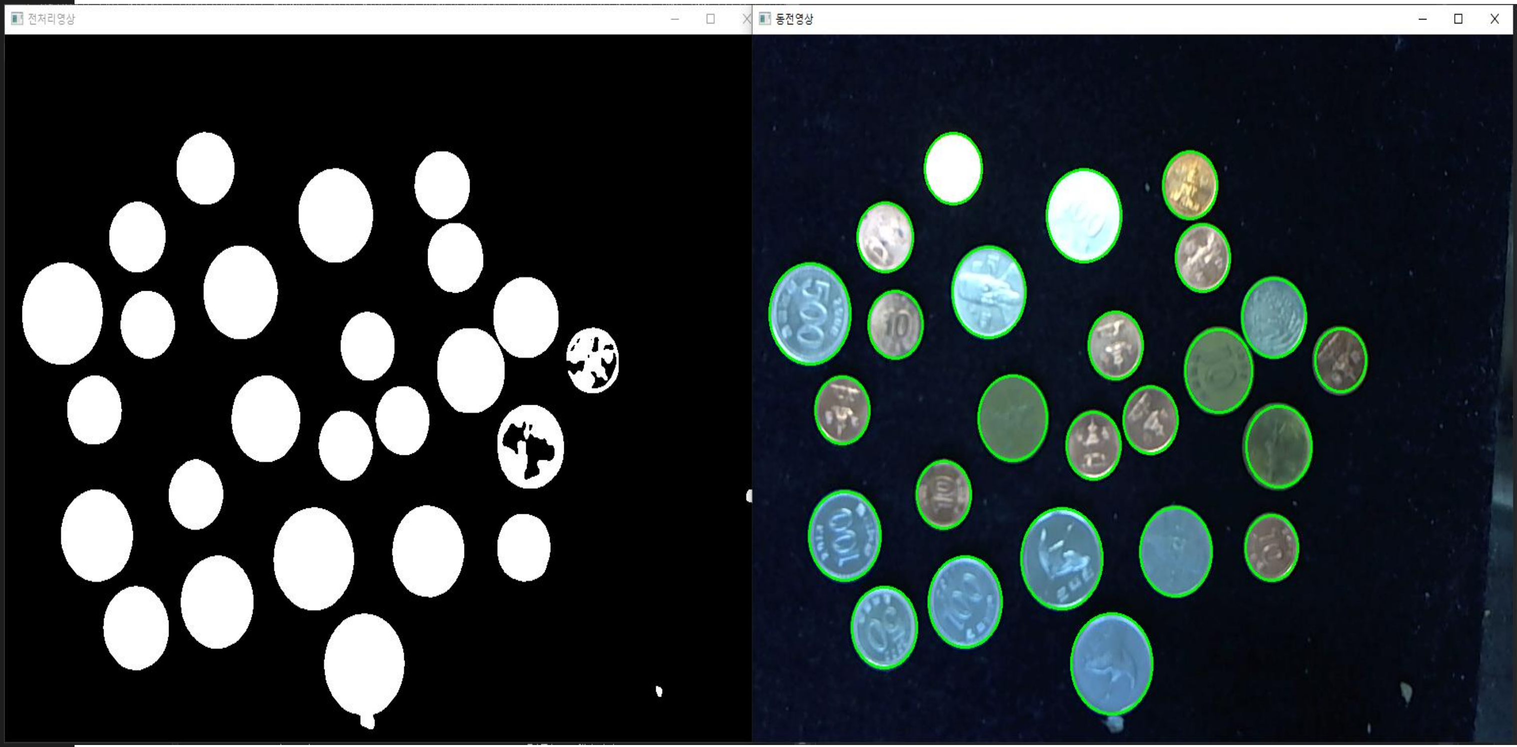




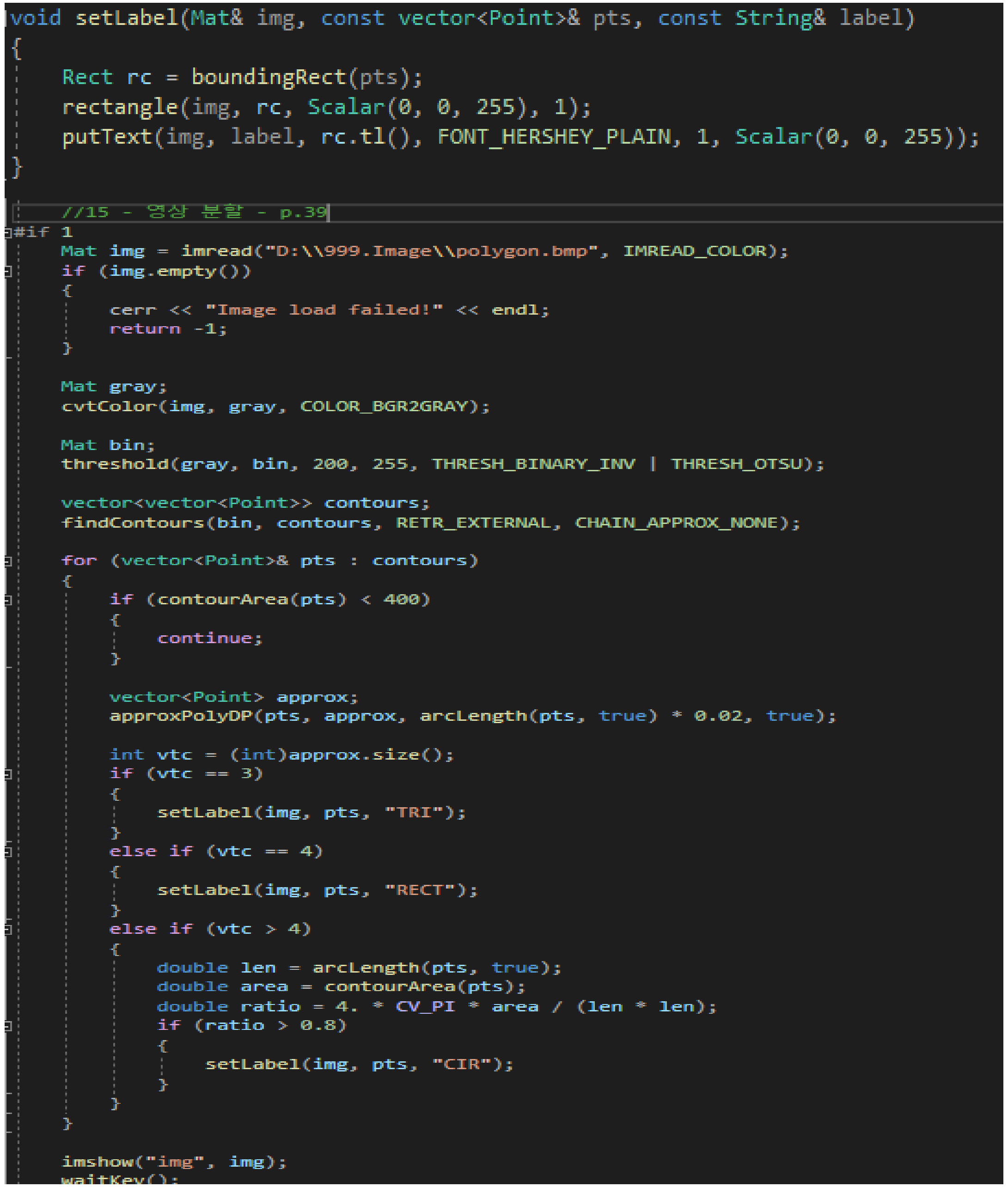
: 15.영상분할 – p.34 ~ 36



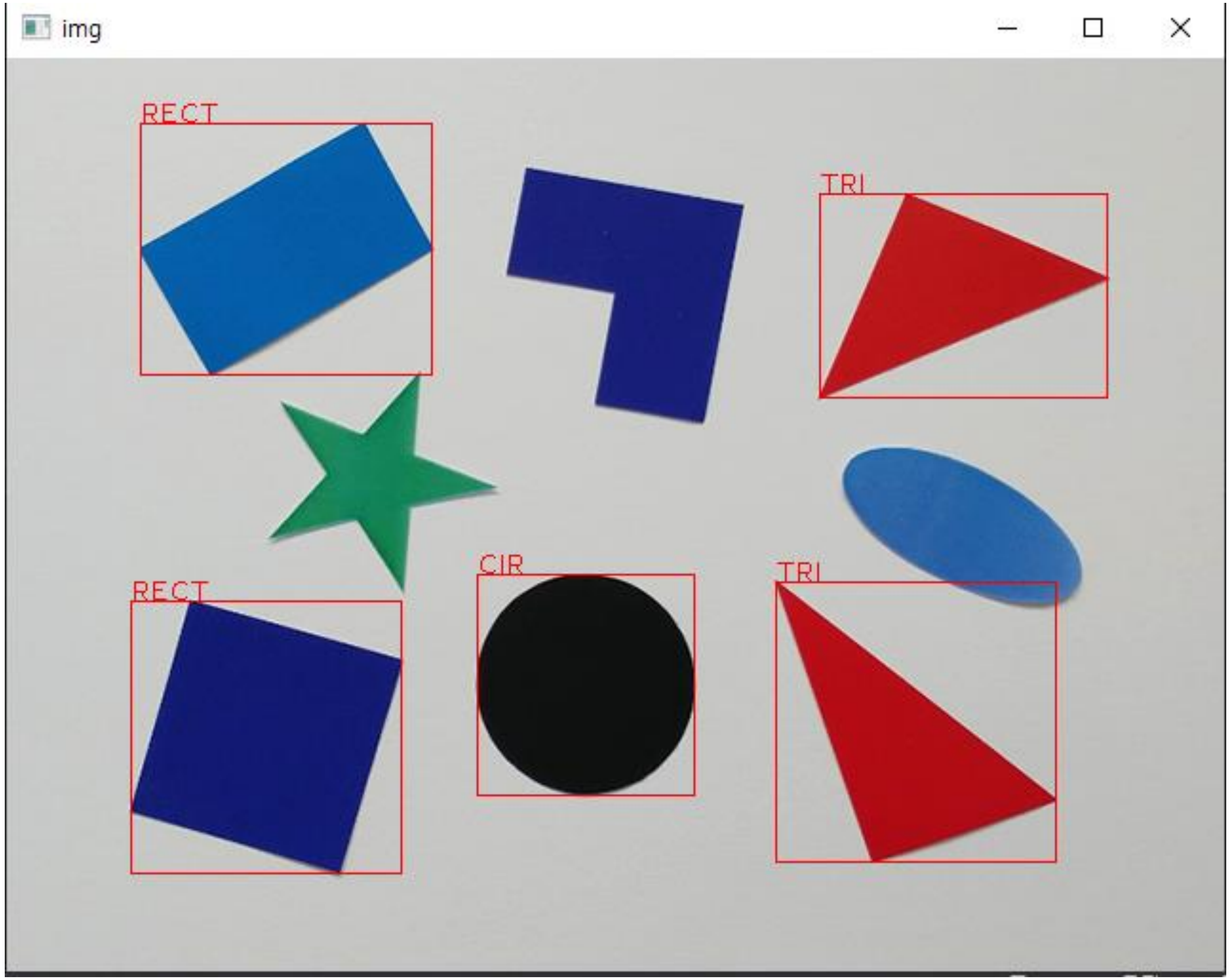
```
//전처리
Mat  preprocessing(Mat img)
{
    Mat gray, th_img;
    cvtColor(img, gray, COLOR_BGR2GRAY);
    GaussianBlur(gray, gray, Size(7, 7), 2, 2);

    threshold(gray, th_img, 130, 255, THRESH_BINARY | THRESH_OTSU);
    morphologyEx(th_img, th_img, MORPH_OPEN, Mat(), Point(-1, -1), 1);

    return th_img;
}

// 검출 영역 원좌표로 반환
vector<RotatedRect> find_coins(Mat img)
{
    vector<vector<Point> > contours;
    findContours(img.clone(), contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

    vector<RotatedRect> circles;
    for (int i = 0; i < (int)contours.size(); i++)
    {
        RotatedRect  mr = minAreaRect(contours[i]);
        mr.angle = (mr.size.width + mr.size.height) / 4.0f;

        if (mr.angle > 18)
        {
            circles.push_back(mr);
        }
    }
    return circles;
}
    //15 - 영상 분할 - p.34
#if 1
    int coin_no = 20;
    String  fname = format("D:\\999.Image\\Coin\\%2d.png", coin_no);
    Mat  image = imread(fname, 1);
    CV_Assert(image.data);

    Mat th_img = preprocessing(image);
    vector<RotatedRect> circles = find_coins(th_img);

    for (int i = 0; i < circles.size(); i++)
    {
        float radius = circles[i].angle;
        circle(image, circles[i].center, radius, Scalar(0, 255, 0), 2);
    }

    imshow("전처리영상", th_img);
    imshow("동전영상", image);
    waitKey();
#endif
```



: 15.영상분할 – p.39



```
void setLabel(Mat& img, const vector<Point>& pts, const String& label)
{
    Rect rc = boundingRect(pts);
    rectangle(img, rc, Scalar(0, 0, 255), 1);
    putText(img, label, rc.tl(), FONT_HERSHEY_PLAIN, 1, Scalar(0, 0, 255));
}

    //15 - 영상 분할 - p.39
#if 1
    Mat img = imread("D:\\999.Image\\polygon.bmp", IMREAD_COLOR);
    if (img.empty())
    {
        cerr << "Image load failed!" << endl;
        return -1;
    }

    Mat gray;
    cvtColor(img, gray, COLOR_BGR2GRAY);

    Mat bin;
    threshold(gray, bin, 200, 255, THRESH_BINARY_INV | THRESH_OTSU);

    vector<vector<Point>> contours;
    findContours(bin, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);

    for (vector<Point>& pts : contours)
    {
        if (contourArea(pts) < 400)
        {
            continue;
        }

        vector<Point> approx;
        approxPolyDP(pts, approx, arcLength(pts, true) * 0.02, true);

        int vtc = (int)approx.size();
        if (vtc == 3)
        {
            setLabel(img, pts, "TRI");
        }
        else if (vtc == 4)
        {
            setLabel(img, pts, "RECT");
        }
        else if (vtc > 4)
        {
            double len = arcLength(pts, true);
            double area = contourArea(pts);
            double ratio = 4. * CV_PI * area / (len * len);
            if (ratio > 0.8)
            {
                setLabel(img, pts, "CIR");
            }
        }
    }

    imshow("img", img);
    waitKey();
```


img
RECT
TRI
RECT
CIR
TRI