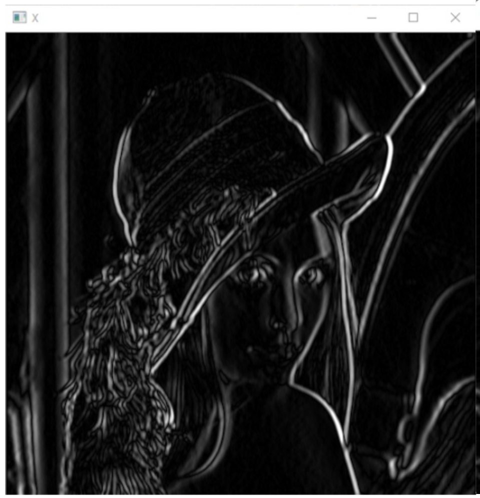
# Sobel_Canny

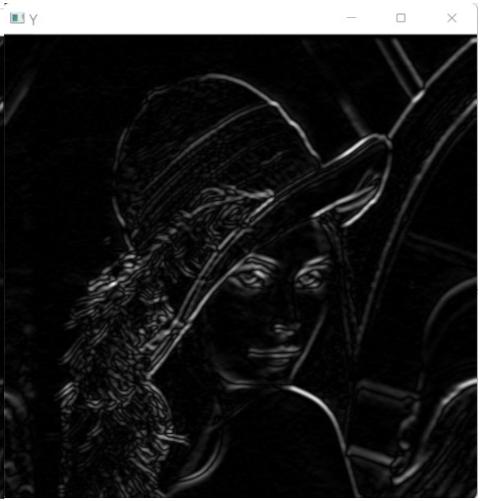
Sung Soo Hwang

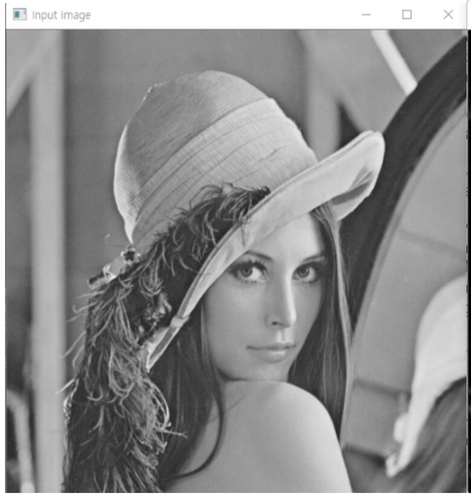
# Example code

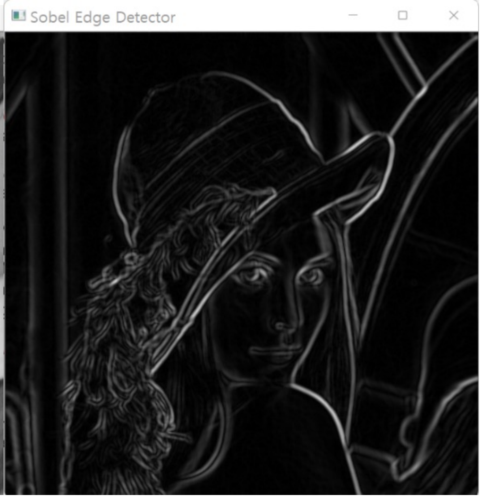
- Sobel edge detector

```
int main() {
        Mat image, blur, grad_x, grad_y, abs_grad_x, abs_grad_y, result;
        image = imread("lena.png", 0);
        GaussianBlur(image, blur, Size(5, 5), 5, 5, BORDER_DEFAULT);

        //performs Sobel operation which is a discrete differentiation
        //blur: input Mat, grad_x: output Mat, CV_16S: depth of the output Mat
        //1: order of derivative in x direction, 0: order of derivative in y direction
        //3: size of the extended Sobel kernel; it must be 1, 3, 5, or 7.
        Sobel(blur, grad_x, CV_16S, 1, 0, 3);
        convertScaleAbs(grad_x, abs_grad_x);

        Sobel(blur, grad_y, CV_16S, 0, 1, 3);
        convertScaleAbs(grad_y, abs_grad_y);

        //abs_grad_x : intput $g_x$ Mat
        //0.5 : weight for abs_grad_x
        //abs_grad_y : intput $g_y$ Mat
        //0.5 : weight for abs_grad_y
        //0 : offset added to weighted sum
        //result : output Mat
        addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, result);

        imshow("X", abs_grad_x);
        imshow("Y", abs_grad_y);
        imshow("Input image", image);
        imshow("Sobel Edge Detector", result);

        waitKey(0);
}
```





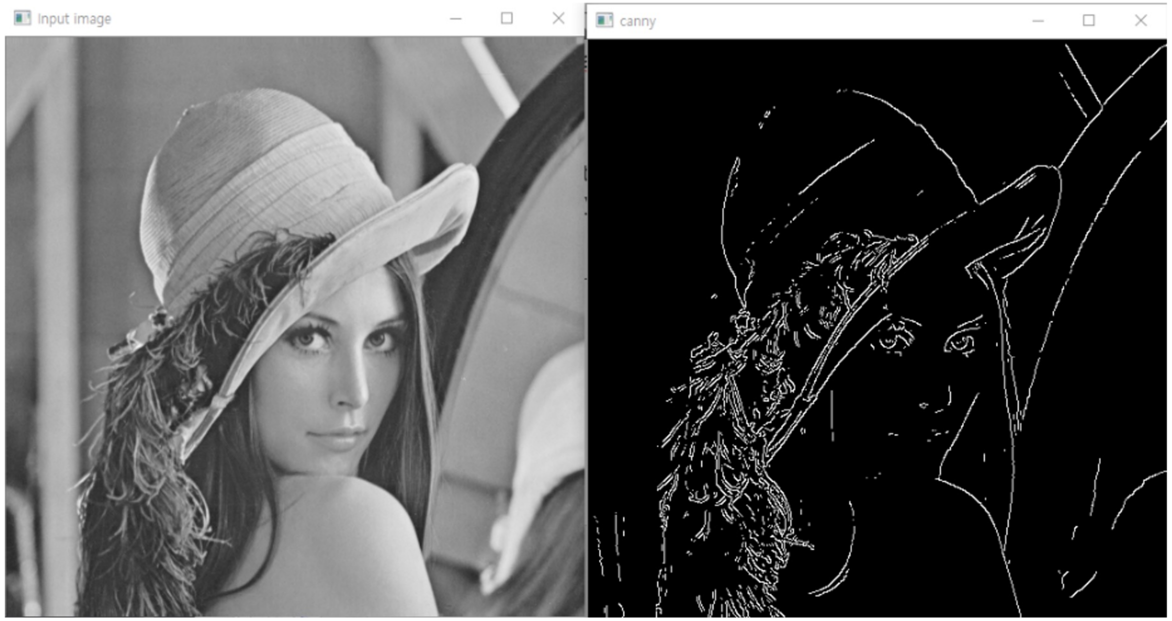




# Example code

- Canny edge operator

```
int main() {
Mat image, canny;
image = imread("lena.png", 0);

//performs canny edge detection
//image: input Mat, canny: output Mat
//190: Thresh_low of double thresholding
//200: Thresh_high of double thresholding
//3: aperture size of the Sobel operation
Canny(image, canny, 190, 200, 3);

imshow("Input image", image);
imshow("canny", canny);

waitKey(0);
}
```



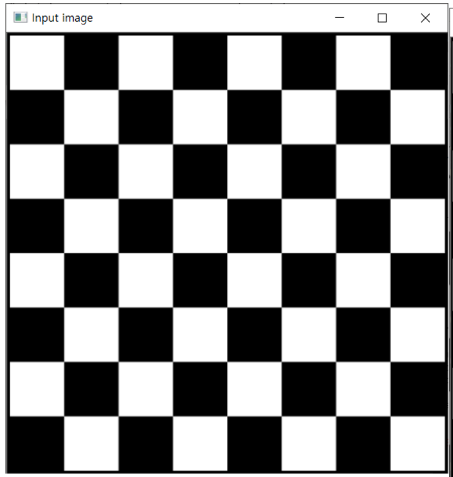
# HoughLines_HoughLinesP

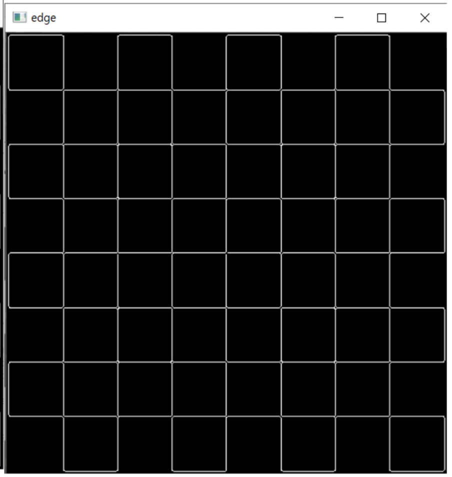
Sung Soo Hwang

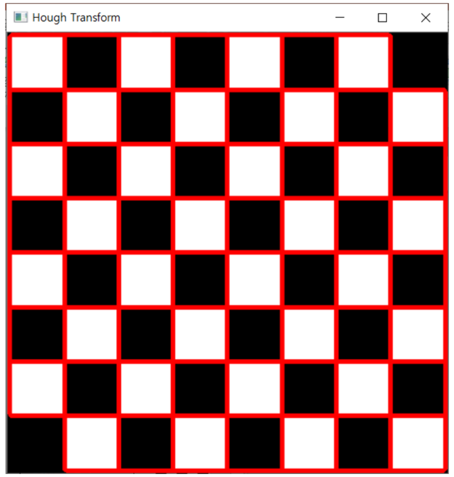
# Example

- HoughLines

```
int main() {
        Mat image, edge, result;
        float rho, theta, a, b, x0, y0;
        Point p1, p2;
        vector<Vec2f> lines;
        image = imread("chess_pattern.png");
        result = image.clone();

        cvtColor(image, image, CV_BGR2GRAY);
        Canny(image, edge, 50, 200, 3);

        //applying Hough Transform to find lines in the image
        //edge: input Mat, lines: output vector of lines
        //1: (rho) distance resolution of the accumulator in pixels
        //CV_PI/180: (theta) angle resolution of the accumulator in radians
        //150: (threshold) accumulator threshold parameter
        //minimum angle to check for lines. Must fall between 0 and max_theta.
        //maximum angle to check for lines. Must fall between min_theta and CV_PI
        HoughLines(edge, lines, 1, CV_PI / 180, 150,0,CV_PI);

        for (int i = 0; i < lines.size(); i++) {
                rho = lines[i][0];
                theta = lines[i][1];
                a = cos(theta);
                b = sin(theta);

                x0 = a * rho;
                y0 = b * rho;

                p1 = Point(cvRound(x0 + 1000 * (-b)), cvRound(y0 + 1000 * a));
                p2 = Point(cvRound(x0 - 1000 * (-b)), cvRound(y0 - 1000 * a));

                line(result, p1, p2, Scalar(0, 0, 255), 3, 8);
        }
        imshow("Input image", image);
        imshow("edge", edge);
        imshow("Hough Transform", result);
        waitKey(0);
}
```



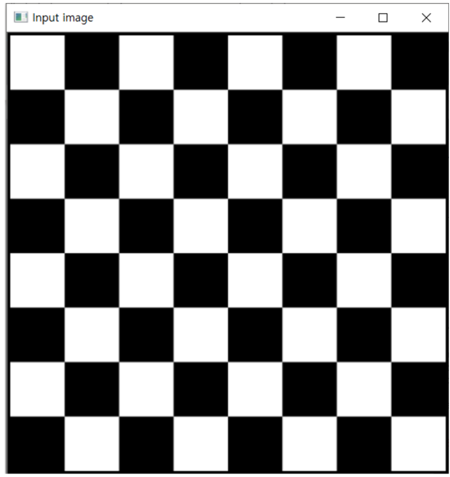


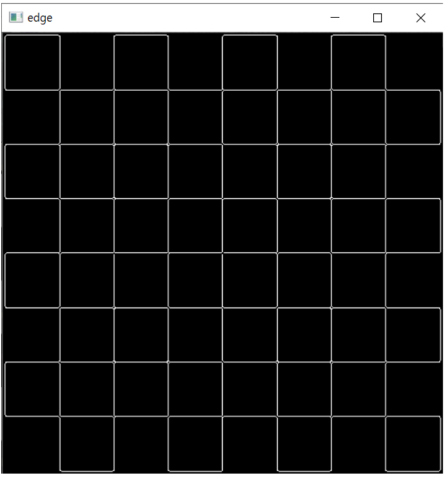


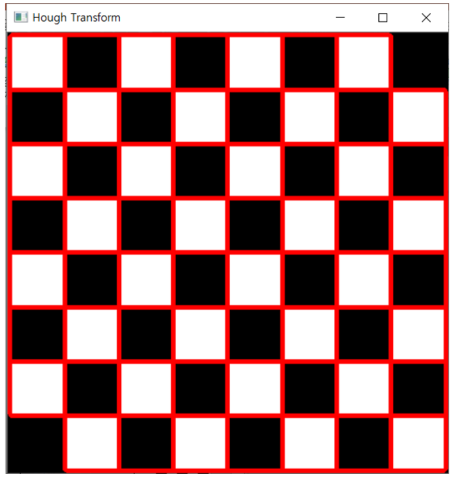
# Example

- HoughLinesP

```
int main() {
        Mat image, edge, result;
        vector<Vec4i> lines;

        image = imread("chess_pattern.png");
        result = image.clone();
        cvtColor(image, image, CV_BGR2GRAY);
        Canny(image, edge, 50, 200, 3);
        //edge: input Mat, lines: output vector of lines
        //1: (rho) distance resolution of the accumulator in pixels
        //CV_PI/180: (theta) angle resolution of the accumulator in radians
        //50: (threshold) accumulator threshold parameter
        //10: (minLineLength) minimum line length.
        //300: (maxLineGap) Maximum allowed gap between points on the same line to link them
        HoughLinesP(edge, lines, 1, CV_PI / 180, 50, 10, 300);

        for (int i = 0; i < lines.size(); i++) {
                Vec4i l = lines[i];
                line(result, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 0, 255), 3, 8);
        }

        imshow("Input image", image);
        imshow("edge", edge);
        imshow("Hough Transform", result);
        waitKey(0);
}
```







# Difference between HoughLines and HoughLinesP

1. Result (check the second parameter)

- HoughLines() computes rho and theta for each line
  →vector<Vec2f> lines
  → i-th value of lines have the rho and theta value of i-th detected line.
- HoughLInesP() computes two points for each line
  → vector<Vec4i> lines
  → i-th value of lines have the (x, y) value for a point and another (x, y) value for a nother point of the i-th detected line.

2. Default Parameters

- HoughLines() have default parameters about rho and theta, such as minimum/ma ximum angle to check for lines.
- HoughLinesP() has default parameters about line segments, such as minimum line length.