Homework requirement:

Implement active contour algorithm(snake algorithm)

1. Set initial points
2. Find contour

Each test image requires one result image and one result video.

Development environment:

Windows 10

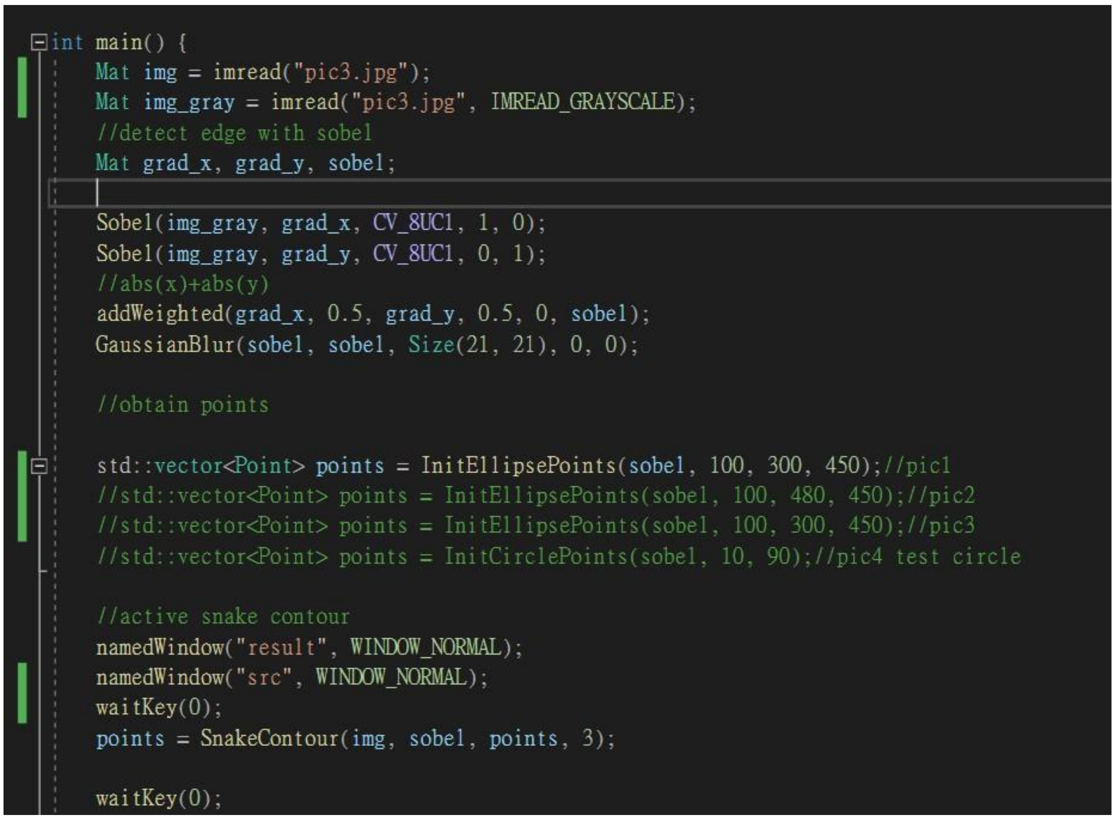
Visual Studio 2022 community

OpenCV 4.6.0

Implementation:

Main function:

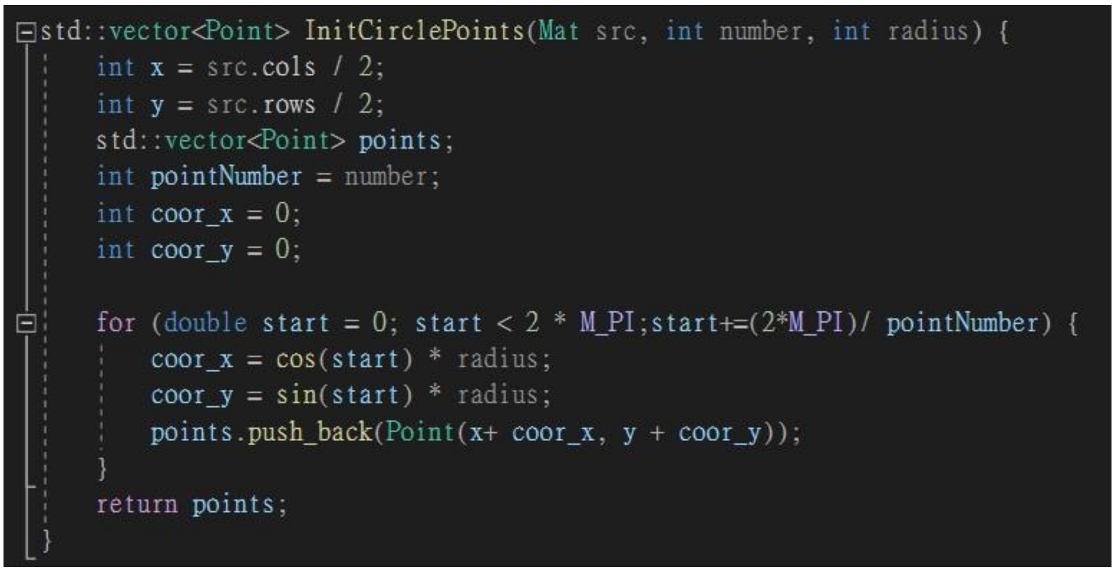
1. Generate a gradient image with Sobel(), and blur the gradient image to increase the area for the active contour algorithm.
2. Drop initial points on the image.
3. Call SnakeContour() to get the result of active contour.



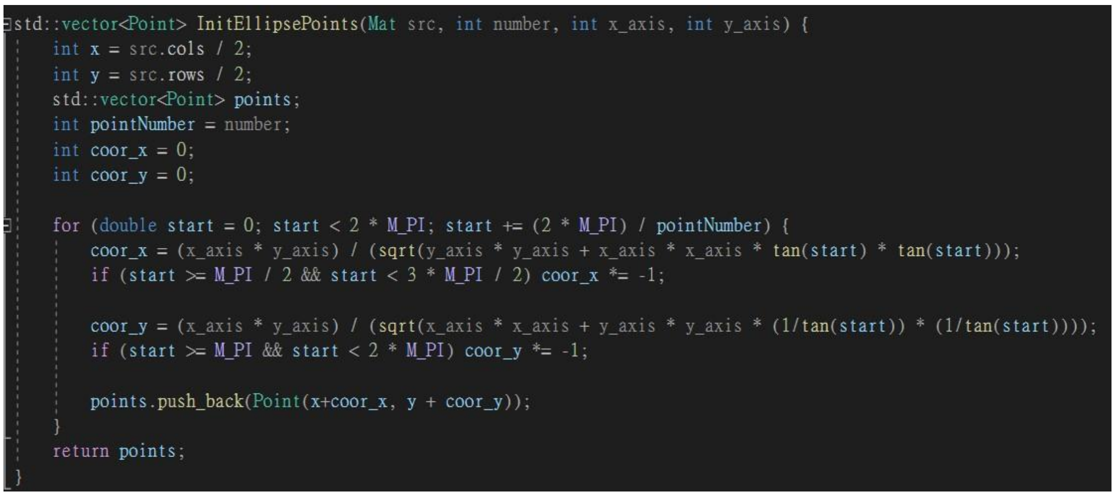
```
int main() {
    Mat img = imread("pic3.jpg");
    Mat img_gray = imread("pic3.jpg", IMREAD_GRAYSCALE);
    //detect edge with sobel
    Mat grad_x, grad_y, sobel;

    Sobel(img_gray, grad_x, CV_8UC1, 1, 0);
    Sobel(img_gray, grad_y, CV_8UC1, 0, 1);
    //abs(x)+abs(y)
    addWeighted(grad_x, 0.5, grad_y, 0.5, 0, sobel);
    GaussianBlur(sobel, sobel, Size(21, 21), 0, 0);

    //obtain points

    std::vector<Point> points = InitEllipsePoints(sobel, 100, 300, 450);//pic1
    //std::vector<Point> points = InitEllipsePoints(sobel, 100, 480, 450);//pic2
    //std::vector<Point> points = InitEllipsePoints(sobel, 100, 300, 450);//pic3
    //std::vector<Point> points = InitCirclePoints(sobel, 10, 90);//pic4 test circle

    //active snake contour
    namedWindow("result", WINDOW_NORMAL);
    namedWindow("src", WINDOW_NORMAL);
    waitKey(0);
    points = SnakeContour(img, sobel, points, 3);

    waitKey(0);
```

InitCirclePoints() & InitEllipsePoints():

Use the equation of circle and ellipse to generate points evenly regarding to angle.



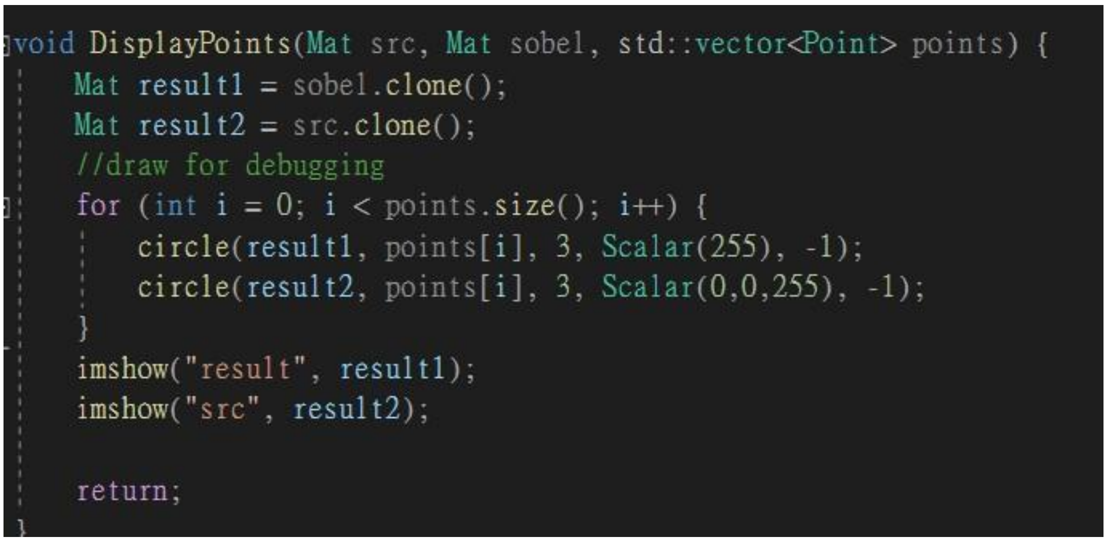
```cpp
std::vector<Point> InitCirclePoints(Mat src, int number, int radius) {
    int x = src.cols / 2;
    int y = src.rows / 2;
    std::vector<Point> points;
    int pointNumber = number;
    int coor_x = 0;
    int coor_y = 0;

    for (double start = 0; start < 2 * M_PI;start+=(2*M_PI)/ pointNumber) {
        coor_x = cos(start) * radius;
        coor_y = sin(start) * radius;
        points.push_back(Point(x+ coor_x, y + coor_y));
    }
    return points;
}
```



```cpp
std::vector<Point> InitEllipsePoints(Mat src, int number, int x_axis, int y_axis) {
    int x = src.cols / 2;
    int y = src.rows / 2;
    std::vector<Point> points;
    int pointNumber = number;
    int coor_x = 0;
    int coor_y = 0;

    for (double start = 0; start < 2 * M_PI; start += (2 * M_PI) / pointNumber) {
        coor_x = (x_axis * y_axis) / (sqrt(y_axis * y_axis + x_axis * x_axis * tan(start) * tan(start)));
        if (start >= M_PI / 2 && start < 3 * M_PI / 2) coor_x *= -1;

        coor_y = (x_axis * y_axis) / (sqrt(x_axis * x_axis + y_axis * y_axis * (1/tan(start)) * (1/tan(start))));
        if (start >= M_PI && start < 2 * M_PI) coor_y *= -1;

        points.push_back(Point(x+coor_x, y + coor_y));
    }
    return points;
}
```

DisplayPoints():

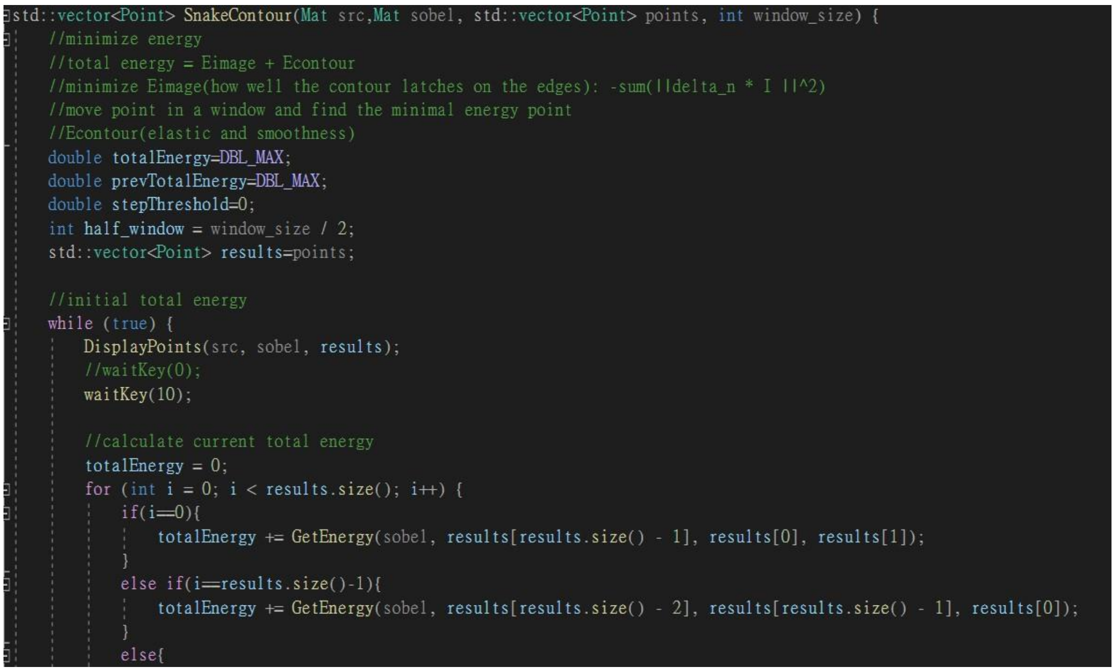
Show the result of the active contour.



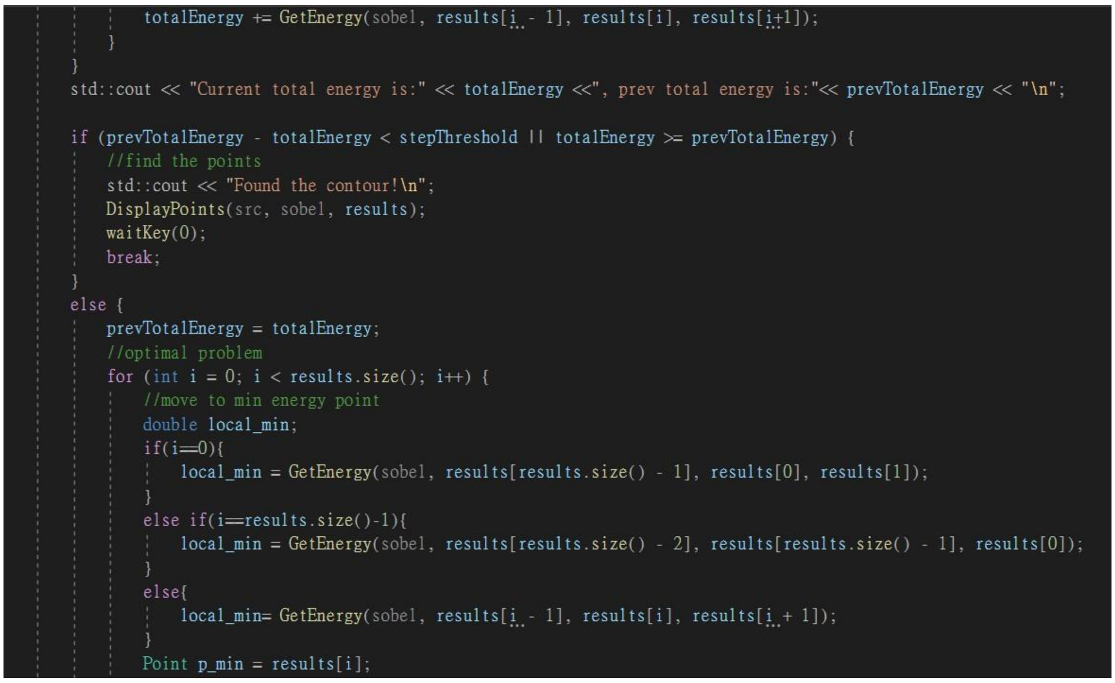
```cpp
void DisplayPoints(Mat src, Mat sobel, std::vector<Point> points) {
    Mat result1 = sobel.clone();
    Mat result2 = src.clone();
    //draw for debugging
    for (int i = 0; i < points.size(); i++) {
        circle(result1, points[i], 3, Scalar(255), -1);
        circle(result2, points[i], 3, Scalar(0,0,255), -1);
    }
    imshow("result", result1);
    imshow("src", result2);

    return;
}
```

SnakeContour():

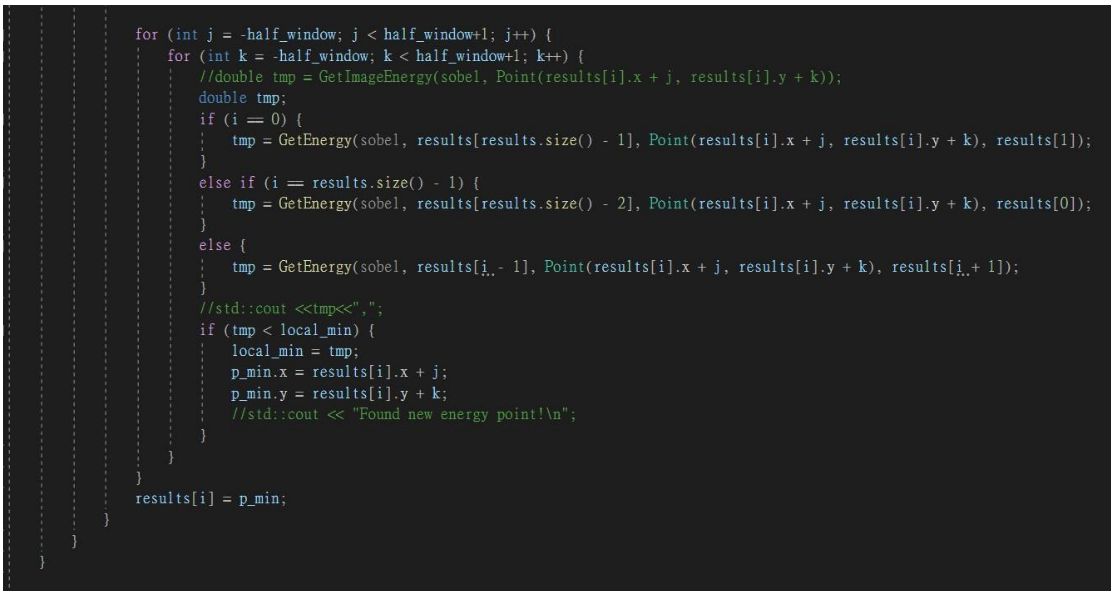
Sequentially iterate through each point. For each point, altering the position of previous point, next point in the window. Save the minimum energy. Compare the current total energy with previous total energy. If the total energy stop decresing, quit the while loop and output the result. Otherwise, calculate the next total minimum energy.



```cpp
std::vector<Point> SnakeContour(Mat src,Mat sobel, std::vector<Point> points, int window_size) {
    //minimize energy
    //total energy = Eimage + Econtour
    //minimize Eimage(how well the contour latches on the edges): -sum(||delta_n * I ||^2)
    //move point in a window and find the minimal energy point
    //Econtour(elastic and smoothness)
    double totalEnergy=DBL_MAX;
    double prevTotalEnergy=DBL_MAX;
    double stepThreshold=0;
    int half_window = window_size / 2;
    std::vector<Point> results=points;

    //initial total energy
    while (true) {
        DisplayPoints(src, sobel, results);
        //waitKey(0);
        waitKey(10);

        //calculate current total energy
        totalEnergy = 0;
        for (int i = 0; i < results.size(); i++) {
            if(i==0){
                totalEnergy += GetEnergy(sobel, results[results.size() - 1], results[0], results[1]);
            }
            else if(i==results.size()-1){
                totalEnergy += GetEnergy(sobel, results[results.size() - 2], results[results.size() - 1], results[0]);
            }
            else{
```

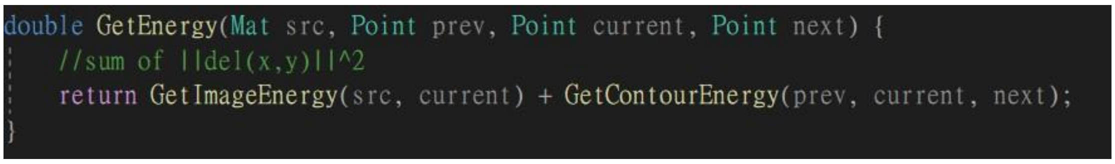


```
                totalEnergy += GetEnergy(sobel, results[i - 1], results[i], results[i+1]);
            }
        }
        std::cout << "Current total energy is:" << totalEnergy <<", prev total energy is:"<< prevTotalEnergy << "\n";

        if (prevTotalEnergy - totalEnergy < stepThreshold || totalEnergy >= prevTotalEnergy) {
            //find the points
            std::cout << "Found the contour!\n";
            DisplayPoints(src, sobel, results);
            waitKey(0);
            break;
        }
        else {
            prevTotalEnergy = totalEnergy;
            //optimal problem
            for (int i = 0; i < results.size(); i++) {
                //move to min energy point
                double local_min;
                if(i==0){
                    local_min = GetEnergy(sobel, results[results.size() - 1], results[0], results[1]);
                }
                else if(i==results.size()-1){
                    local_min = GetEnergy(sobel, results[results.size() - 2], results[results.size() - 1], results[0]);
                }
                else{
                    local_min= GetEnergy(sobel, results[i - 1], results[i], results[i + 1]);
                }
                Point p_min = results[i];
```



```
                for (int j = -half_window; j < half_window+1; j++) {
                    for (int k = -half_window; k < half_window+1; k++) {
                        //double tmp = GetImageEnergy(sobel, Point(results[i].x + j, results[i].y + k));
                        double tmp;
                        if (i == 0) {
                            tmp = GetEnergy(sobel, results[results.size() - 1], Point(results[i].x + j, results[i].y + k), results[1]);
                        }
                        else if (i == results.size() - 1) {
                            tmp = GetEnergy(sobel, results[results.size() - 2], Point(results[i].x + j, results[i].y + k), results[0]);
                        }
                        else {
                            tmp = GetEnergy(sobel, results[i - 1], Point(results[i].x + j, results[i].y + k), results[i + 1]);
                        }
                        //std::cout <<tmp<<",";
                        if (tmp < local_min) {
                            local_min = tmp;
                            p_min.x = results[i].x + j;
                            p_min.y = results[i].y + k;
                            //std::cout << "Found new energy point!\n";
                        }
                    }
                }
                results[i] = p_min;
            }
        }
    }
```

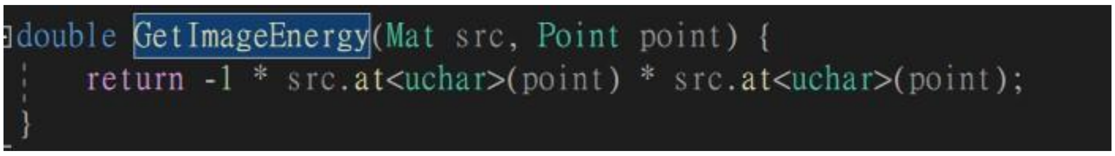
GetEnergy():



```
double GetEnergy(Mat src, Point prev, Point current, Point next) {
    //sum of ||del(x,y)||^2
    return GetImageEnergy(src, current) + GetContourEnergy(prev, current, next);
}
```

GetImageEnergy():

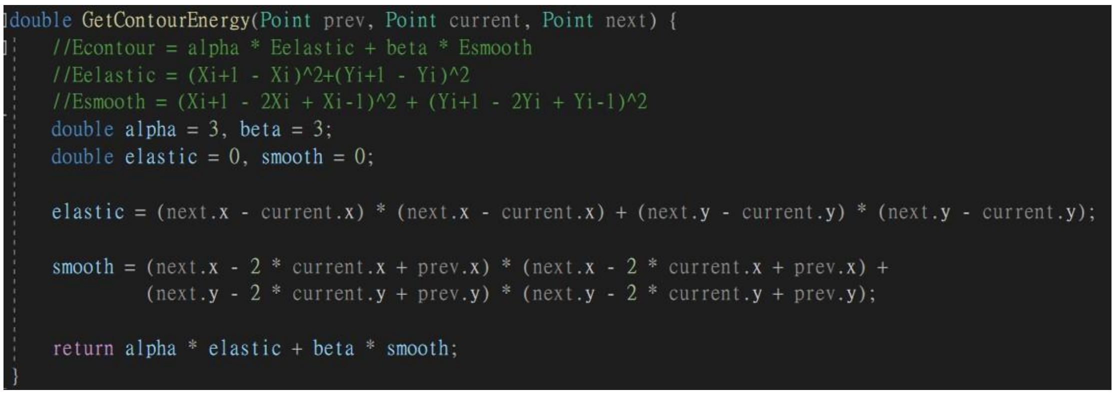
The external energy is gradient magnitude square.



```
double GetImageEnergy(Mat src, Point point) {
    return -1 * src.at<uchar>(point) * src.at<uchar>(point);
}
```

GetContourEnergy():

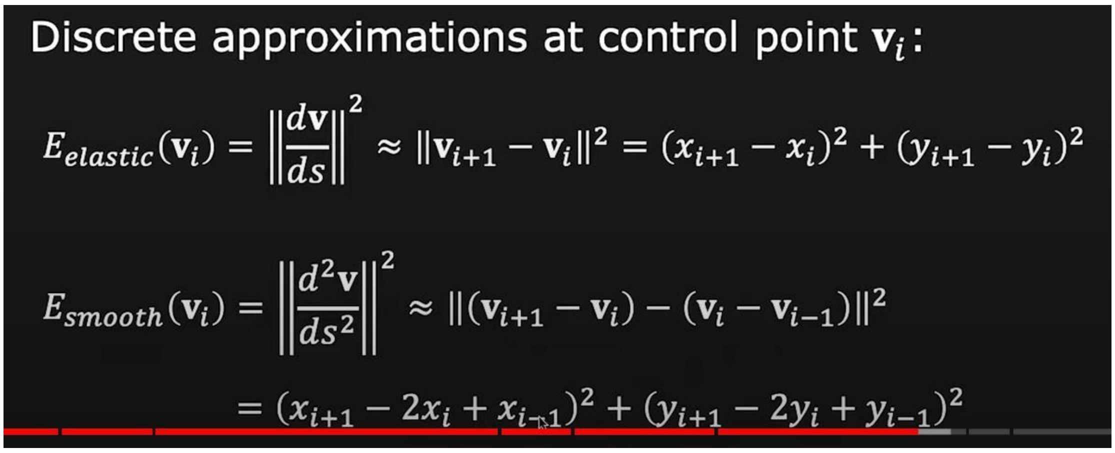
The internal energy is alpha * elastic + beta * smooth.



```
double GetContourEnergy(Point prev, Point current, Point next) {
    //Econtour = alpha * Eelastic + beta * Esmooth
    //Eelastic = (Xi+1 - Xi)^2+(Yi+1 - Yi)^2
    //Esmooth = (Xi+1 - 2Xi + Xi-1)^2 + (Yi+1 - 2Yi + Yi-1)^2
    double alpha = 3, beta = 3;
    double elastic = 0, smooth = 0;

    elastic = (next.x - current.x) * (next.x - current.x) + (next.y - current.y) * (next.y - current.y);

    smooth = (next.x - 2 * current.x + prev.x) * (next.x - 2 * current.x + prev.x) +
            (next.y - 2 * current.y + prev.y) * (next.y - 2 * current.y + prev.y);

    return alpha * elastic + beta * smooth;
}
```

Elastic energy is (the first derivative of distance between current point and next point)^2

Smooth energy is (the second derivative of distance between current point and next point, previous point)^2



Discrete approximations at control point $\mathbf{v}_i$:

$$E_{elastic}(\mathbf{v}_i) = \left\|\frac{d\mathbf{v}}{ds}\right\|^2 \approx \|\mathbf{v}_{i+1} - \mathbf{v}_i\|^2 = (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$$

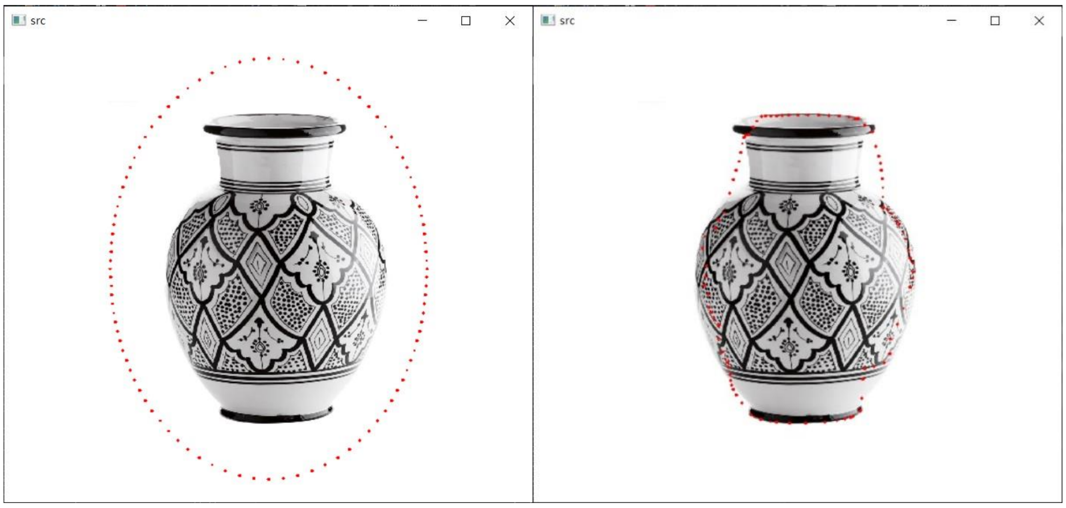
$$E_{smooth}(\mathbf{v}_i) = \left\|\frac{d^2\mathbf{v}}{ds^2}\right\|^2 \approx \|(\mathbf{v}_{i+1} - \mathbf{v}_i) - (\mathbf{v}_i - \mathbf{v}_{i-1})\|^2$$

$$= (x_{i+1} - 2x_i + x_{i-1})^2 + (y_{i+1} - 2y_i + y_{i-1})^2$$

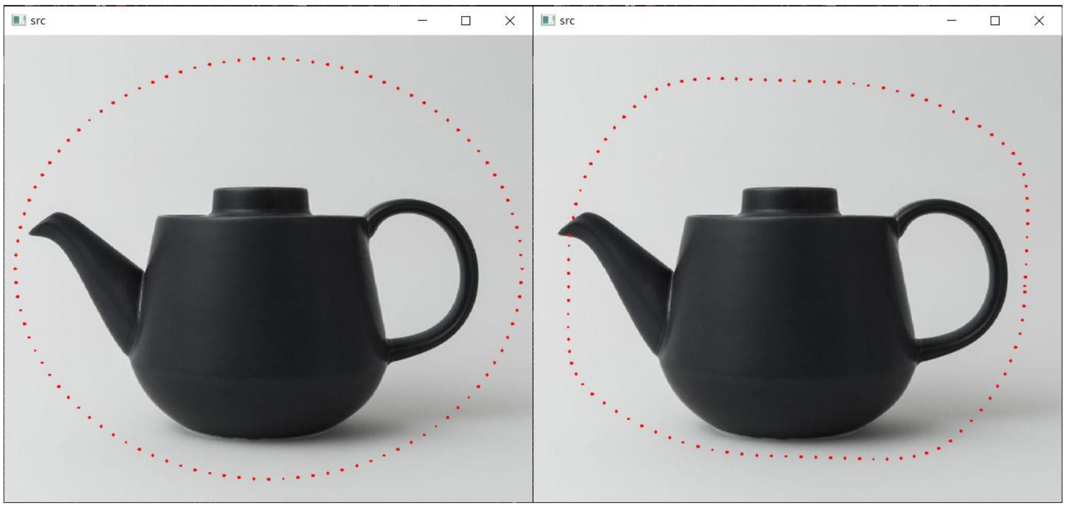
Reference: https://youtu.be/FROJUMk9P3Y

Result:

Pic1



Pic2



Pic3