Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Дисциплина: Алгоритмы цифровой обработки мультимедиа**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А.Мазницкий

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Крамаренко

**Тема работы:** Методы выделения границ. Алгоритм Канни.

**Ход работы:**

Задание 1. Реализовать метод, который принимает в качестве строки полный адрес файла изображения, читает изображение, переводит его в черно-белый цвет и выводит его на экран применяет размытие по Гауссу и выводит полученное изображение на экран.

Задание 2. Модифицировать построенный метод так, чтобы в результате вычислялось и выводилось на экран две матрицы – матрица значений длин и матрица значений углов градиентов всех пикселей изображения.

Задание 3. Модифицировать метод так, чтобы он выполнял подавление немаксимумов и выводил полученное изображение на экран. Рассмотреть изображение, сделать выводы.

Задание 4. Модифицировать метод так, чтобы он выполнял двойную пороговую фильтрацию и выводил полученное изображение на экран.

Задание 5 (самостоятельно). Провести опыты для различных параметров размытия и различных пороговых значений градиента, определить наилучшие параметры для Вашего изображения. Показать преподавателю значения параметров и результат работы на следующем занятии.

Задание 6 (самостоятельно). Реализовать алгоритм Канни на другом языке программирования.

В лабораторной работе необходимо было реализовать алгоритм Канни средствами языка Python и встроенными методами библиотеки OpenCV.

В первую очередь, согласно заданию 1, необходимо было прочитать полный адрес файла изображения, перевести его в чёрно-белый цвет и вывести на экран, применяя размытие по Гауссу. Использовались стандартные функции, разобранные в предыдущих лабораторных работах. Фрагмент кода, выполняющий это, представлен на рисунке 1.

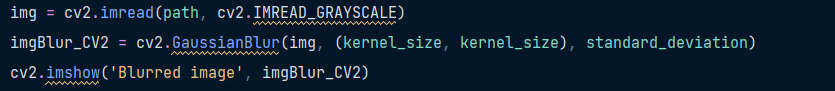


Рисунок 1 – Чтение и размытие изображения.

Далее необходимо было вывести на экран две матрицы – матрицу значений длин и матрицу значений углов градиентов всех пикселей изображения, т. е. необходимо было выполнить второй шаг алгоритма Канни.

Сперва была реализована операция свёртки, модифицированная из прошлой лабораторной работы, с отличием в переопределении матрицы изображения для работы с каждым внутренним пикселем. Функция представлена на рисунке 2.

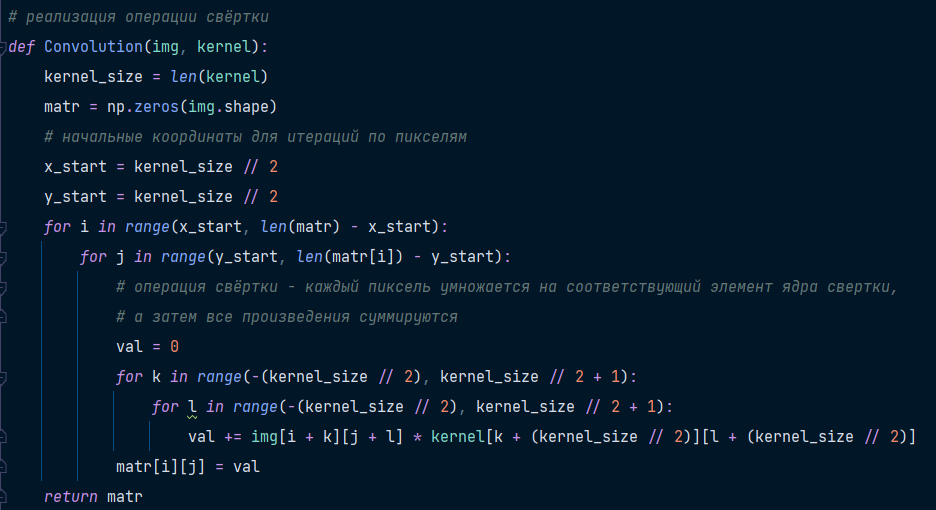


Рисунок 2 – Реализация операции свёртки.

Затем в теле основной функции были определены матрицы оператора Собеля (рисунок 3).

Изображение выглядит как Шрифт, рукописный текст, белый, текст

Автоматически созданное описание

Рисунок 3 – Матрицы оператора Собеля

После этого к изображению и к каждой из выше определенных матриц была применена операция свёртки – img\_Gx = Convolution(img, Gx) и img\_Gy = Convolution(img, Gy). Вновь была переопределена матрица изображения для работы с внутренними пикселями.

Далее с помощью цикла for, проходящему по каждому пикселю изображения, и формулы была найдена матрица длины вектора градиента (рисунок 4).

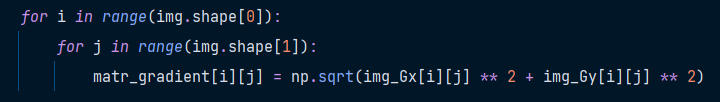


Рисунок 4 – Нахождение матрицы длины вектора градиента.

Для поиска матрицы значений углов градиента предварительно необходимо было определить функцию, которая находит округления угла между вектором градиента и осью Х. На рисунке 5 представлена её реализация, а на рисунке 6 – классическая математическая схема воспользоваться для округления угла.

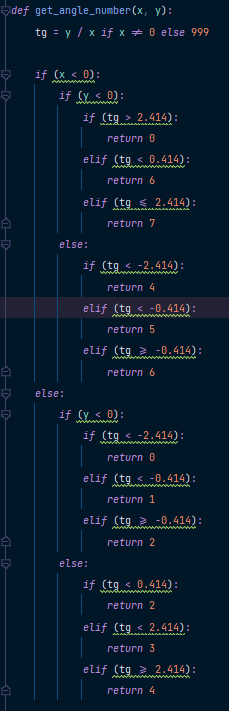


Рисунок 5 – Функция нахождения округления угла между вектором градиента и осью Х.

Округление нужно, так как нет необходимости точно знать, какова величина угла 10 или 15 градусов, например. По факту, необходимо для направления выбрать лишь один из соседних 8 пикселей. То есть необходимо округлить величину угла до 45 градусов.

Изображение выглядит как текст, снимок экрана, Красочность, диаграмма

Автоматически созданное описание

Рисунок 6 – Схема округления угла до 45 градусов.

Вновь используя цикла for, проходящему по каждому пикселю изображения, была найдена матрица углов градиента (img\_angles[i][j] = get\_angle\_number(img\_Gx[i][j], img\_Gy[i][j]). Инициализация была вновь выполнена перед этим.

Для корректного отображения матриц в окне необходимо было привести их значения в нужную форму. Так, для матрицы значений длин градиента сперва было найдено максимальное значение (max\_gradient = np.max(matr\_gradient)), а затем каждый внутренний пиксель изображения был поделен на полученный максимум и умножен на 255, чтобы привести значения длин градиента к диапазону от 0 до 255. Для примера, предположим, что максимальное значение длины градиента равно 100. Если бы мы не выполняли деление на максимальное значение и умножение на 255, то значения длин градиента в матрице могли бы быть любыми числами в диапазоне от 0 до 100. Однако, для корректного отображения матрицы на экране, значения должны быть в диапазоне от 0 до 255. Похожим образом, была изменена матрица значений углов градиента. Её элементы делились на 7 и умножались на 255, , чтобы привести значения углов градиента к диапазону от 0 до 255 (ведь они могут принимать любое значение от 0 до 360). С помощью метода библиотеки OpenCV отображались соответствующие окна (cv2.imshow()). Код, выполняющий это, представлен на рисунке 7.



Рисунок 7 – Вывод полученных матриц.

В задании 3 требуется подавить немаксимумы. Определим что это. В алгоритме Канни немаксимум (non-maximum suppression) — это процесс, который используется для уменьшения ширины границ, обнаруженных на изображении. Он заключается в том, что для каждого пикселя на границе изображения проверяется, является ли он локальным максимумом в направлении градиента (направление градиента перпендикулярно границам). Если пиксель не является локальным максимумом, то его значение устанавливается в 0. Это позволяет сохранить только те пиксели на границе изображения, которые имеют максимальную длину градиента в направлении градиента. Говоря другими словами: границей будет считаться пиксель, градиент которого максимален в сравнении с пикселями по направлению наибольшего роста функции; если значение градиента выше, чем у пикселей слева и справа, то данный пиксель – это граница, иначе – не граница.

Вновь предварительно инициализировав матрицу границ изображения с помощью цикла for для каждого внутреннего пикселя изображения выполнялись следующие действия: сперва значение угла градиента сохранялось в переменной angle, а значение длины градиента - в переменной gradient, затем если пиксель находится на границе изображения, то его значение в матрице img\_border(матрица границ изображения) устанавливалось в 0, в противном случае, для каждого пикселя определяется смещение по осям X и Y, которое зависит от значения угла градиента. Оно определялось по следующим правилам:

1) если угол градиента равен 0 или 4 (то есть градиент направлен вдоль оси X), то смещение по оси X будет равно 0, а смещение по оси Y - также равно 0;

2) если угол градиента равен 2 или 6 (то есть градиент направлен вдоль оси Y), то смещение по оси X будет равно 0, а смещение по оси Y - также равно 0;

3) если угол градиента равен 1 (то есть градиент направлен под углом 45 градусов вправо вверх), то смещение по оси X будет равно 1, а смещение по оси Y - равно -1;

4) если угол градиента равен 3 (то есть градиент направлен под углом 45 градусов вправо вниз), то смещение по оси X будет равно -1, а смещение по оси Y - также равно -1;

5) если угол градиента равен 5 (то есть градиент направлен под углом 45 градусов влево вниз), то смещение по оси X будет равно 1, а смещение по оси Y - равно -1;

6) если угол градиента равен 7 (то есть градиент направлен под углом 45 градусов влево вверх), то смещение по оси X будет равно 1, а смещение по оси Y - также равно 1;

В конце с помощью сравнения определялось, являлся ли пиксель максимальным значением градиента. Функция cv2.imshow() показывала окно с результатом. На рисунке 8 отображён описанный фрагмент кода.

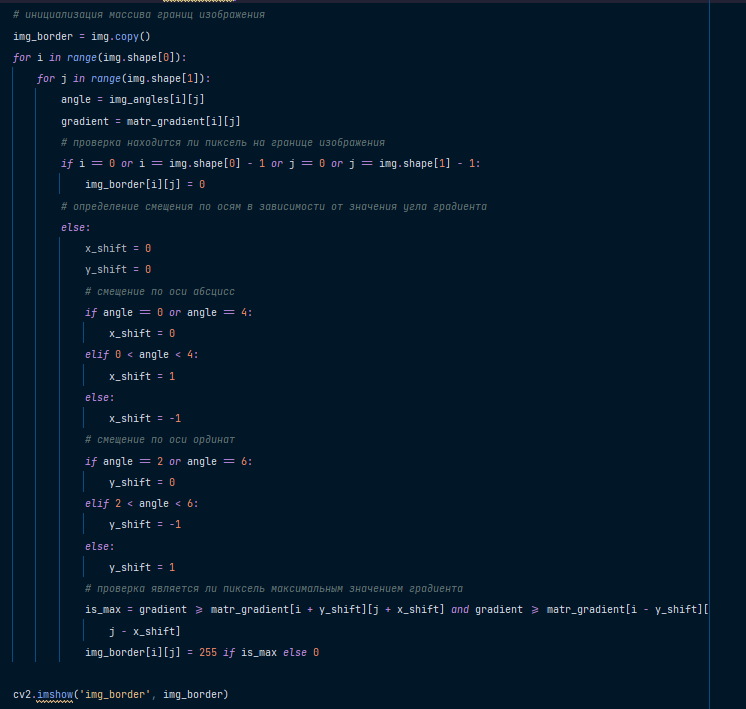


Рисунок 8 – Подавление немаксимумов.

Последним шагом в алгоритме Канни является построение двойной фильтрации, что соответствует заданию 4 лабораторной работы. Она используется в алгоритме для определения границ на изображении. Процесс заключается в том, что для каждого пикселя на границе изображения сравнивается величина его градиента с двумя пороговыми значениями. Это позволяет отфильтровать шумы и сохранить только те границы, которые имеют максимальную длину градиента и являются значимыми.

Сначала были заданы пороговые границ для градиента исходя из максимального градиента по изображению. В данном коде нижняя и верхняя границы для длины градиента определялись так, чтобы отфильтровать границы изображения, которые имеют слишком маленькую или слишком большую длину градиента. Нижняя граница вычисляется как max\_gradient / threshold, а верхняя - как max\_gradient - max\_gradient / threshold. Значение threshold задает коэффициент, который определяет, какую часть от максимального значения длины градиента следует использовать для определения нижней и верхней границ. После, аналогично предыдущим случаям, матрица результата double\_filtration была проинициализирована нулями. Для каждого пикселя изображения с помощью цикла for проверялось, находится ли он на границе изображения (значение пикселя в матрице границ img\_border равно 255). Если это так, то выполнялась проверка, находится ли значение длины градиента в заданном диапазоне. Если это соответствует истине, то для каждого пикселя проверяется, есть ли среди его соседей пиксель с максимальной длиной градиента. Для этого использовался двойной цикл for по переменным k и l, принимающим значения от -1 до 1. Затем проверка значение пикселя в матрице границ img\_border для пикселя, расположенного на расстоянии k по оси X и на расстоянии l по оси Y от текущего пикселя. Если значение пикселя равно 255 и значение длины градиента для этого пикселя больше или равно нижней границе, то устанавливается флаг flag в значение True. Если значение длины градиента больше верхней границы, то значение пикселя в результирующей матрице устанавливается в 255. Наконец, она выводится на экран с помощью функции cv2.imshow(). На рисунке 9 представлен код двойной пороговой фильтрации.



Рисунок 9 – Двойная пороговая фильтрация.

Последним заданием является опыты с изображением. На рисунке 10 представлен результат работы алгоритма Канни со стандартным отклонением равным 3 и размером матрицы свёртки равным 3.

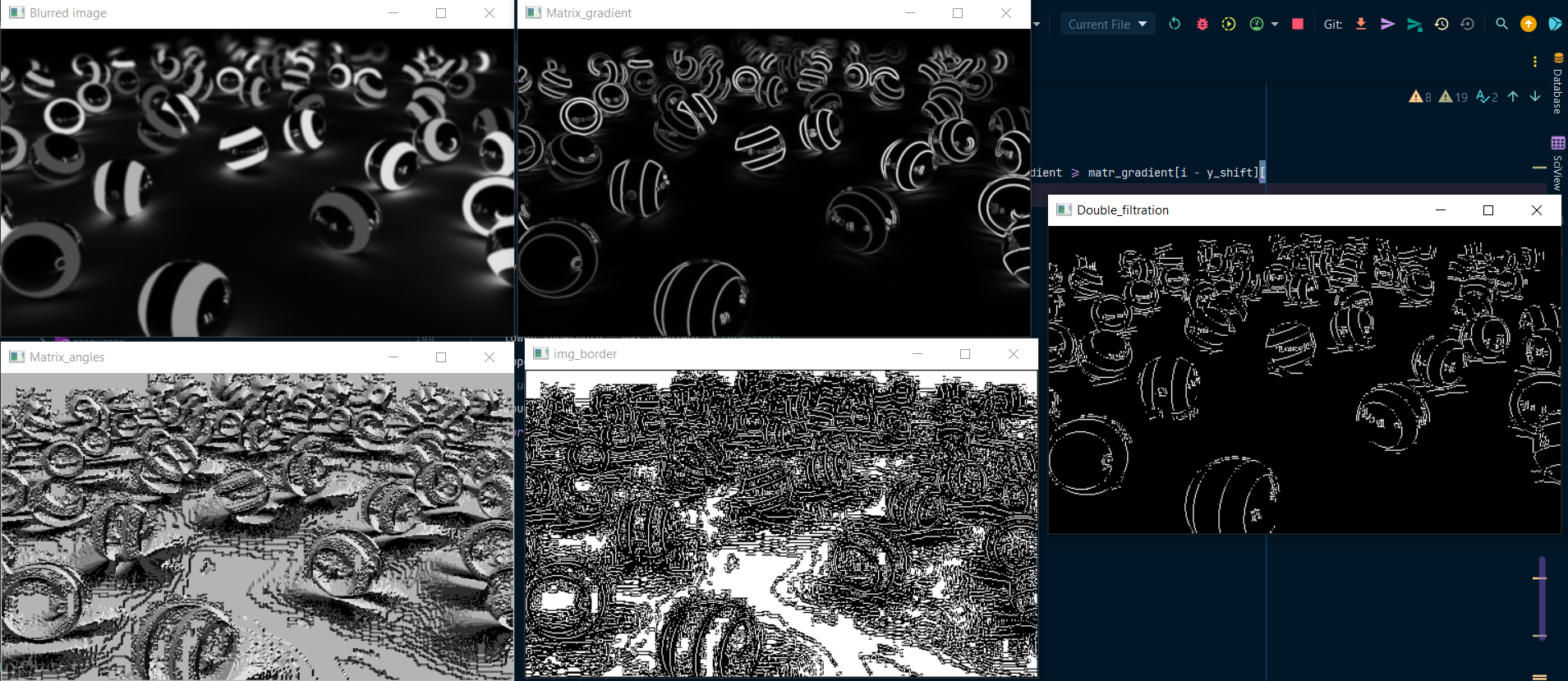


Рисунок 10 – Алгоритм Канни со стандартным отклонением равным 3 и размером матрицы свёртки равным 3.

Изменим параметры на новые (рисунок 11).

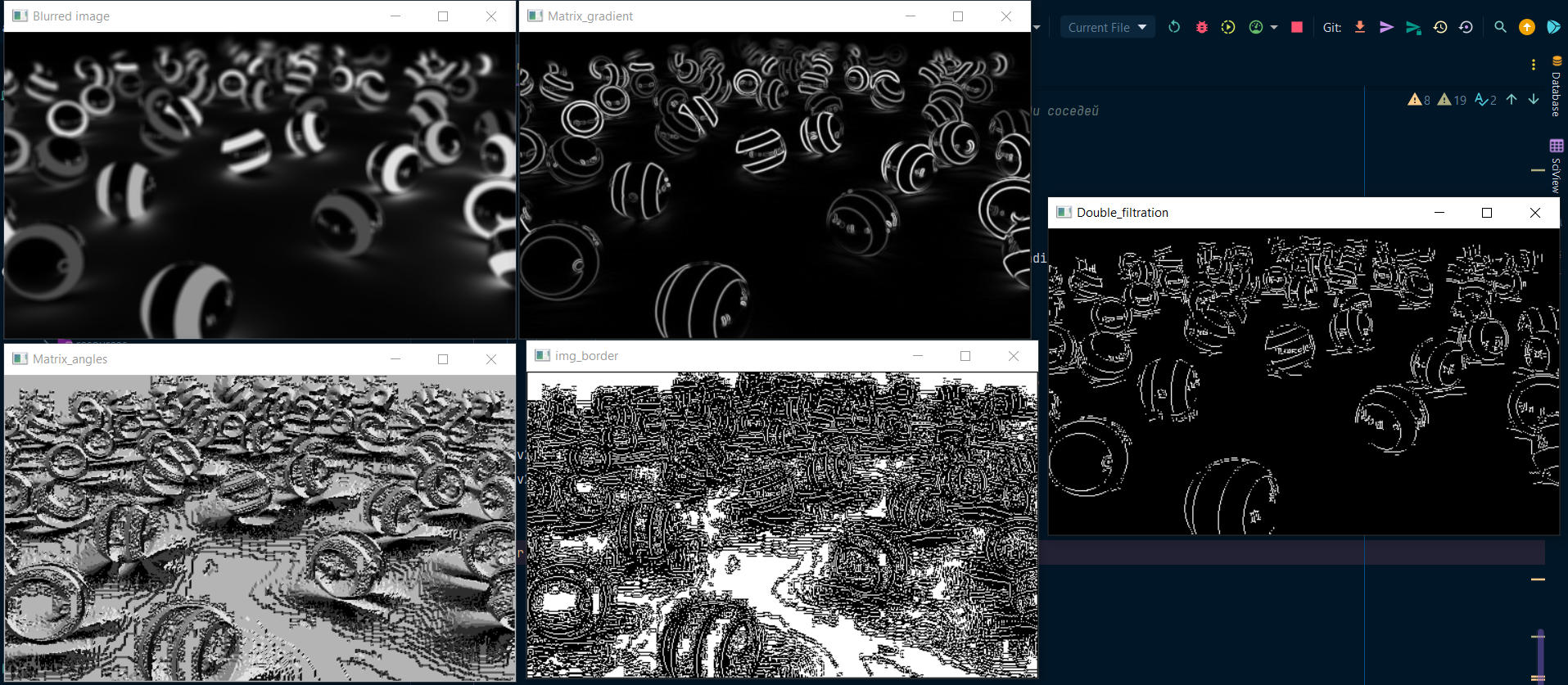


Рисунок 11 – Алгоритм Канни со стандартным отклонением равным 6 и размером матрицы свёртки равным 5.

Повторим изменение (рисунок 12).

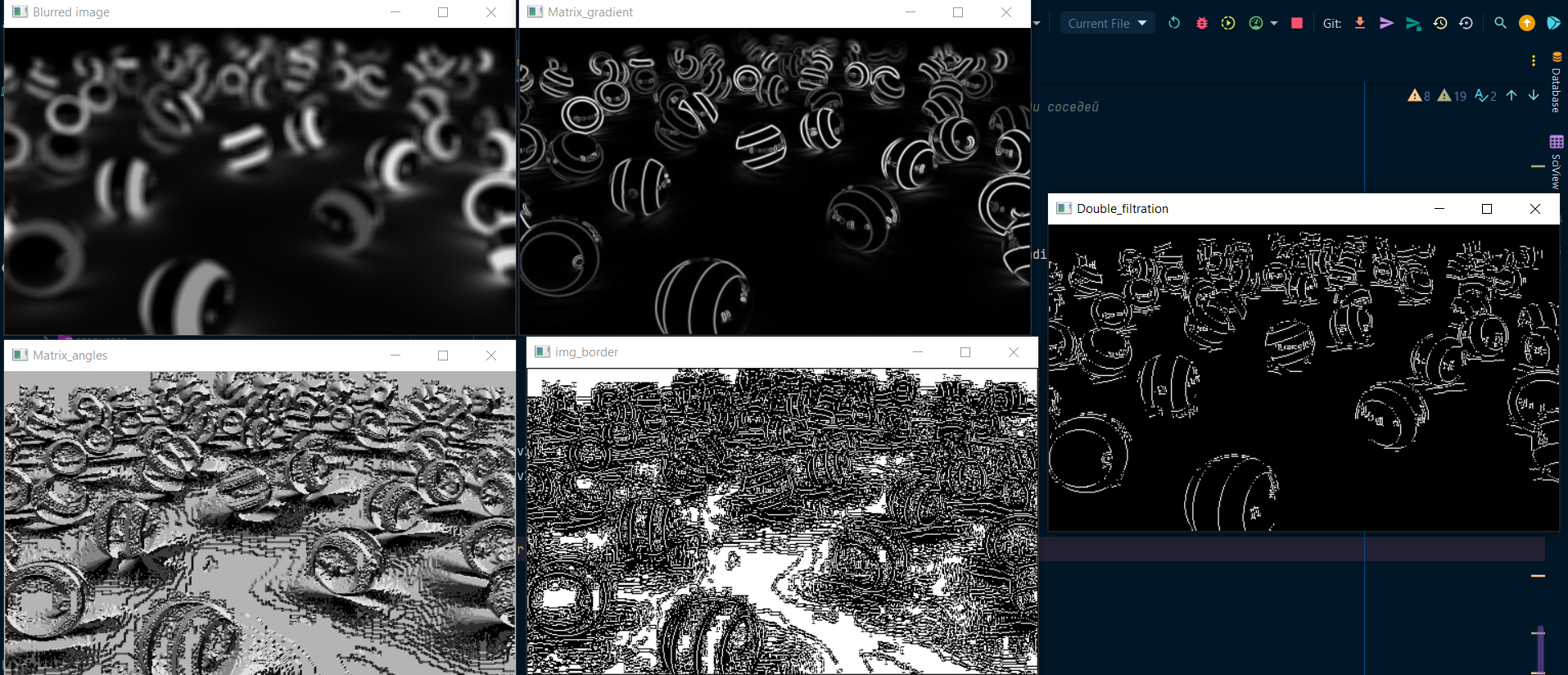
****

Рисунок 12 – Алгоритм Канни со стандартным отклонением равным 100 и размером матрицы свёртки равным 9.

Не трудно заметить, что с увеличением параметров алгоритма, увеличиваются соответствующие матрицы и увеличивается детализированность границ на изображении.

Для выполнения задания 6 лабораторной работы программа была переписана на язык C#. Алгоритм Канни, реализованные на языке Python, были переписаны на синтаксис языка С#. Получившийся код прикреплен в Листинге.

**Листинг программ**

Файл Canny\_Algorithm.py

*import* cv2  
*import* numpy *as* np  
  
  
*# реализация операции свёртки  
def* Convolution(img, kernel):  
 kernel\_size = *len*(kernel)  
 matr = np.zeros(img.shape)  
 *# начальные координаты для итераций по пикселям* x\_start = kernel\_size // 2  
 y\_start = kernel\_size // 2  
 *for* i *in range*(x\_start, *len*(matr) - x\_start):  
 *for* j *in range*(y\_start, *len*(matr[i]) - y\_start):  
 *# операция свёртки - каждый пиксель умножается на соответствующий элемент ядра свертки,  
 # а затем все произведения суммируются* val = 0  
 *for* k *in range*(-(kernel\_size // 2), kernel\_size // 2 + 1):  
 *for* l *in range*(-(kernel\_size // 2), kernel\_size // 2 + 1):  
 val += img[i + k][j + l] \* kernel[k + (kernel\_size // 2)][l + (kernel\_size // 2)]  
 matr[i][j] = val  
 *return* matr  
  
  
*# нахождение округления угла между вектором градиента и осью Х  
def* get\_angle\_number(x, y):  
 tg = y / x *if* x != 0 *else* 999  
  
 *if* (x < 0):  
 *if* (y < 0):  
 *if* (tg > 2.414):  
 *return* 0  
 *elif* (tg < 0.414):  
 *return* 6  
 *elif* (tg <= 2.414):  
 *return* 7  
 *else*:  
 *if* (tg < -2.414):  
 *return* 4  
 *elif* (tg < -0.414):  
 *return* 5  
 *elif* (tg >= -0.414):  
 *return* 6  
 *else*:  
 *if* (y < 0):  
 *if* (tg < -2.414):  
 *return* 0  
 *elif* (tg < -0.414):  
 *return* 1  
 *elif* (tg >= -0.414):  
 *return* 2  
 *else*:  
 *if* (tg < 0.414):  
 *return* 2  
 *elif* (tg < 2.414):  
 *return* 3  
 *elif* (tg >= 2.414):  
 *return* 4  
  
  
*def* main(path, standard\_deviation, kernel\_size, threshold):  
 *# Задание 1 - чтение строки полного адреса изображения и размытие Гаусса* img = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)  
 imgBlur\_CV2 = cv2.GaussianBlur(img, (kernel\_size, kernel\_size), standard\_deviation)  
  
 cv2.imshow('Blurred image', imgBlur\_CV2)  
  
 *# Задание 2 - вычисление и вывод матрицы длин и матрицы углов градиентов с помощью матриц оператора Собеля* Gx = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]  
 Gy = [[-1, -2, -1], [0, 0, 0], [1, 2, 1]]  
  
 *# применение оператора свёртки* img\_Gx = Convolution(img, Gx)  
 img\_Gy = Convolution(img, Gy)  
  
 *# переопределение матрицы изображения для работы с каждым внутренним пикселем* matr\_gradient = np.zeros(img.shape)  
  
 *# нахождение матрицы длины вектора градиента  
 for* i *in range*(img.shape[0]):  
 *for* j *in range*(img.shape[1]):  
 matr\_gradient[i][j] = np.sqrt(img\_Gx[i][j] \*\* 2 + img\_Gy[i][j] \*\* 2)  
  
 *# нахождение округления угла между вектором градиента и осью Х* img\_angles = img.copy()  
 *for* i *in range*(img.shape[0]):  
 *for* j *in range*(img.shape[1]):  
 img\_angles[i][j] = get\_angle\_number(img\_Gx[i][j], img\_Gy[i][j])  
  
 *# вывод матрицы значений длин градиента* img\_gradient\_to\_print = img.copy()  
 max\_gradient = np.max(matr\_gradient)  
 *for* i *in range*(img.shape[0]):  
 *for* j *in range*(img.shape[1]):  
 img\_gradient\_to\_print[i][j] = (*float*(matr\_gradient[i][j]) / max\_gradient) \* 255  
 cv2.imshow('Matrix\_gradient', img\_gradient\_to\_print)  
 *print*('Матрица значений длин градиента:')  
 *print*(img\_gradient\_to\_print)  
  
 *# вывод матрицы значений углов градиента* img\_angles\_to\_print = img.copy()  
 *for* i *in range*(img.shape[0]):  
 *for* j *in range*(img.shape[1]):  
 img\_angles\_to\_print[i][j] = img\_angles[i][j] \* 45  
 cv2.imshow('Matrix\_angles', img\_angles\_to\_print)  
 *print*('Матрица значений углов градиента:')  
 *print*(img\_angles\_to\_print)  
  
 *# Задание 3 - подавление немаксимумов  
 # инициализация массива границ изображения* img\_border = img.copy()  
 *for* i *in range*(img.shape[0]):  
 *for* j *in range*(img.shape[1]):  
 angle = img\_angles[i][j]  
 gradient = matr\_gradient[i][j]  
 *# проверка находится ли пиксель на границе изображения  
 if* i == 0 *or* i == img.shape[0] - 1 *or* j == 0 *or* j == img.shape[1] - 1:  
 img\_border[i][j] = 0  
 *# определение смещения по осям в зависимости от значения угла градиента  
 else*:  
 x\_shift = 0  
 y\_shift = 0  
 *# смещение по оси абсцисс  
 if* angle == 0 *or* angle == 4:  
 x\_shift = 0  
 *elif* 0 < angle < 4:  
 x\_shift = 1  
 *else*:  
 x\_shift = -1  
 *# смещение по оси ординат  
 if* angle == 2 *or* angle == 6:  
 y\_shift = 0  
 *elif* 2 < angle < 6:  
 y\_shift = -1  
 *else*:  
 y\_shift = 1  
 *# проверка является ли пиксель максимальным значением градиента* is\_max = gradient >= matr\_gradient[i + y\_shift][j + x\_shift] *and* gradient >= matr\_gradient[i - y\_shift][  
 j - x\_shift]  
 img\_border[i][j] = 255 *if* is\_max *else* 0  
  
 cv2.imshow('img\_border', img\_border)  
 *# Задание 4 - двойная пороговая фильтрация  
 # задание пороговых границ для градиента* lower\_threshold = max\_gradient / threshold  
 upper\_threshold = max\_gradient - max\_gradient / threshold  
 *# инициализация массива результата* double\_filtration = np.zeros(img.shape)  
 *for* i *in range*(img.shape[0]):  
 *for* j *in range*(img.shape[1]):  
 gradient = matr\_gradient[i][j]  
 *# проверка находится ли пиксель на границы изображения  
 if* img\_border[i][j] == 255:  
 *# проверка градиента в диапазоне  
 if* lower\_threshold <= gradient <= upper\_threshold:  
 flag = *False  
 # проверка пикселя с максимальной длиной градиента среди соседей  
 for* k *in range*(-1, 2):  
 *for* l *in range*(-1, 2):  
 *if* flag:  
 *break  
 # поиск границы  
 if* img\_border[i + k][j + l] == 255 *and* matr\_gradient[i + k][j + l] >= lower\_threshold:  
 flag = *True  
 break  
 if* flag:  
 double\_filtration[i][j] = 255  
 *# если значение градиента выше - верхней границы, то пиксель точно граница  
 elif* gradient > upper\_threshold:  
 double\_filtration[i][j] = 255  
 cv2.imshow('Double\_filtration', double\_filtration)  
 cv2.waitKey(0)  
  
  
main(r'..\resources\balls.jpg', 3, 3, 15)

Файл GaussianBlur.cs

using System;

using System.Collections.Generic;

using System.Linq;

namespace GaussianBlur

{

public class MyGauss

{

private static double MyGaussFunc(int x, int y, int size, double omega)

{

int a = size / 2 + 1;

int b = size / 2 + 1;

return (1 / ((2 \* Math.PI) \* Math.Pow(omega, 2))) \* Math.Exp(-((Math.Pow((x - a), 2) + Math.Pow((y - b), 2)) / (2 \* Math.Pow(omega, 2))));

}

private static void MatrixNorm(List<List<double>> matrix)

{

double sum = 0;

for (int i = 0; i < matrix.Count(); i++)

{

for (int j = 0; j < matrix[i].Count(); j++)

{

sum += matrix[i][j];

}

}

for (int i = 0; i < matrix.Count(); i++)

{

for (int j = 0; j < matrix[i].Count(); j++)

{

matrix[i][j] /= sum;

}

}

}

private static List<List<double>> FillGaussMatrix(int size, double omega)

{

List<List<double>> matrix = new List<List<double>>();

for (int i = 0; i < size; i++)

{

matrix.Add(new List<double>());

for (int j = 0; j < size; j++)

{

matrix[i].Add(MyGaussFunc(i, j, size, omega));

}

}

MatrixNorm(matrix);

return matrix;

}

public static List<List<int>> GaussianBlur(List<List<int>> img, int gaussMatrixSize, double omega)

{

int height = img.Count();

int width = img[0].Count();

int x\_start = gaussMatrixSize / 2;

int x\_end = height - gaussMatrixSize / 2;

int y\_start = gaussMatrixSize / 2;

int y\_end = width - gaussMatrixSize / 2;

List<List<double>> gaussMatrix = FillGaussMatrix(gaussMatrixSize, omega);

List<List<int>> blurMatrix = new List<List<int>>(img);

for (int i = x\_start; i < x\_end; i++)

{

for (int j = y\_start; j < y\_end; j++)

{

double value = 0;

int ii = 0;

for (int k = i - gaussMatrixSize / 2; k < i + gaussMatrixSize / 2; k++)

{

int jj = 0;

for (int c = j - gaussMatrixSize / 2; c < j + gaussMatrixSize / 2; c++)

{

value += img[k][c] \* gaussMatrix[ii][jj];

jj++;

}

ii++;

}

blurMatrix[i][j] = Convert.ToInt32(value);

}

}

return blurMatrix;

}

}

}

Файл MyBorderDetector.Cs

using System.Collections.Generic;

using System;

using System.Linq;

namespace MyGradientBorder

{

public class GradientBorder

{

//private List<List<int>> \_originalGreyImage;

private List<List<(int, int)>> \_gradientVectorMatrix;

private List<List<double>> \_gradientVectorLengths;

private List<List<int>> \_gradientVectorAngles;

private double \_maxLength;

public List<List<int>> BorderMatrix

{

get;

private set;

}

private List<List<int>> \_sobelOperatorX = new List<List<int>>()

{

new List<int>() { -1, 0, 1 },

new List<int>() { -2, 0, 2 },

new List<int>() { -1, 0, 1 }

};

private List<List<int>> \_sobelOperatorY = new List<List<int>>()

{

new List<int>() { -1, -2, -1 },

new List<int>() { 0, 0, 0 },

new List<int>() { 1, 2, 1 }

};

public GradientBorder(List<List<int>> originalImage)

{

GetGradientComponents(originalImage);

}

private int GetAngle(int intx, int y)

{

double x = intx;

if (intx == 0)

x = 0.01;

double tan = y / x;

if ((x > 0 && y < 0 && tan < -2.414) || (x < 0 && y < 0 && tan > 2.414))

{

return 0;

}

else if (x > 0 && y < 0 && tan < -0.414)

{

return 1;

}

else if ((x > 0 && y < 0 && tan > -0.414) || (x > 0 && y > 0 && tan < 0.414))

{

return 2;

}

else if (x > 0 && y > 0 && tan < 2.414)

{

return 3;

}

else if ((x > 0 && y > 0 && tan > 2.414) || (x < 0 && y > 0 && tan < -2.414))

{

return 4;

}

else if (x < 0 && y > 0 && tan < -0.414)

{

return 5;

}

else if ((x < 0 && y > 0 && tan > -0.414) || (x < 0 && y < 0 && tan < 0.414))

{

return 6;

}

else if (x < 0 && y < 0 && tan < 2.414)

{

return 7;

}

else if (x > 0)

{

return 2;

}

else if (x < 0)

{

return 6;

}

else

{

throw new Exception();

}

}

private void GetGradientComponents(List<List<int>> originalImage)

{

int convSize2 = \_sobelOperatorX.Count() / 2;

\_gradientVectorMatrix = new List<List<(int, int)>>();

\_gradientVectorAngles = new List<List<int>>();

\_gradientVectorLengths = new List<List<double>>();

\_maxLength = -100000;

for (int i = convSize2; i < originalImage.Count() - convSize2; i++)

{

\_gradientVectorMatrix.Add(new List<(int, int)>());

\_gradientVectorLengths.Add(new List<double>());

\_gradientVectorAngles.Add(new List<int>());

for (int j = convSize2; j < originalImage[i].Count() - convSize2; j++)

{

(int, int) value = (0, 0);

int ii = 0;

for (int k = i - convSize2; k < i + convSize2 + 1; k++)

{

int jj = 0;

for (int c = j - convSize2; c < j + convSize2 + 1; c++)

{

value.Item1 += originalImage[k][c] \* \_sobelOperatorX[ii][jj];

value.Item2 += originalImage[k][c] \* \_sobelOperatorY[ii][jj];

jj++;

}

ii++;

}

\_gradientVectorMatrix[i - 1].Add(value);

double length = Math.Sqrt(Math.Pow(value.Item1, 2) + Math.Pow(value.Item2, 2));

if (length > \_maxLength)

{

\_maxLength = length;

}

\_gradientVectorLengths[i - 1].Add(length);

\_gradientVectorAngles[i - 1].Add(GetAngle(value.Item1, value.Item2));

}

}

}

private void DestroyNonMax()

{

BorderMatrix = new List<List<int>>();

for (int i = 0; i < \_gradientVectorAngles.Count; i++)

{

BorderMatrix.Add(new List<int>());

for (int j = 0; j < \_gradientVectorAngles[i].Count; j++)

{

int ang = \_gradientVectorAngles[i][j];

if (i == 0 || j == 0 || i == \_gradientVectorAngles.Count() - 1 || j == \_gradientVectorAngles[i].Count() - 1)

{

BorderMatrix[i].Add(0);

}

else if (ang == 0 || ang == 4)

{

if (\_gradientVectorLengths[i][j] > \_gradientVectorLengths[i - 1][j] && \_gradientVectorLengths[i][j] > \_gradientVectorLengths[i + 1][j])

{

BorderMatrix[i].Add(255);

}

else

{

BorderMatrix[i].Add(0);

}

}

else if (ang == 2 || ang == 6)

{

if (\_gradientVectorLengths[i][j] > \_gradientVectorLengths[i][j - 1] && \_gradientVectorLengths[i][j] > \_gradientVectorLengths[i][j + 1])

{

BorderMatrix[i].Add(255);

}

else

{

BorderMatrix[i].Add(0);

}

}

else if (ang == 3 || ang == 7)

{

if (\_gradientVectorLengths[i][j] > \_gradientVectorLengths[i + 1][j + 1] && \_gradientVectorLengths[i][j] > \_gradientVectorLengths[i - 1][j - 1])

{

BorderMatrix[i].Add(255);

}

else

{

BorderMatrix[i].Add(0);

}

}

else if (ang == 1 || ang == 5)

{

if (\_gradientVectorLengths[i][j] > \_gradientVectorLengths[i - 1][j + 1] && \_gradientVectorLengths[i][j] > \_gradientVectorLengths[i + 1][j - 1])

{

BorderMatrix[i].Add(255);

}

else

{

BorderMatrix[i].Add(0);

}

}

}

}

}

private void DoubleFiltering(int low, int high)

{

int low\_bord = Convert.ToInt32(\_maxLength / low);

int high\_bord = Convert.ToInt32(\_maxLength / high);

for (int i = 0; i < BorderMatrix.Count; i++)

{

for (int j = 0; j < BorderMatrix[i].Count; j++)

{

if (BorderMatrix[i][j] == 255 && \_gradientVectorLengths[i][j] >= high\_bord)

{

BorderMatrix[i][j] = 255;

}

else if (BorderMatrix[i][j] == 255 && \_gradientVectorLengths[i][j] <= low\_bord)

{

BorderMatrix[i][j] = 0;

}

else if (BorderMatrix[i][j] == 255)

{

BorderMatrix[i][j] = 100;

}

}

}

for (int i = 1; i < BorderMatrix.Count - 1; i++)

{

for (int j = 1; j < BorderMatrix[i].Count - 1; j++)

{

if (BorderMatrix[i][j] == 100)

{

bool b = false;

for (int k = i - 1; k < i + 2; k++)

{

for (int c = j - 1; c < j + 2; c++)

{

if (BorderMatrix[k][c] == 255)

{

BorderMatrix[i][j] = 255;

b = true;

break;

}

}

}

if (b == false)

{

BorderMatrix[i][j] = 0;

}

}

}

}

}

public List<List<int>> GetBorder(int low, int high, bool filter = false)

{

DestroyNonMax();

if (filter)

{

DoubleFiltering(low, high);

}

return BorderMatrix;

}

}