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**

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

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Эзри

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

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

**Цель работы:** реализовать алгоритм выделения границ Канни.

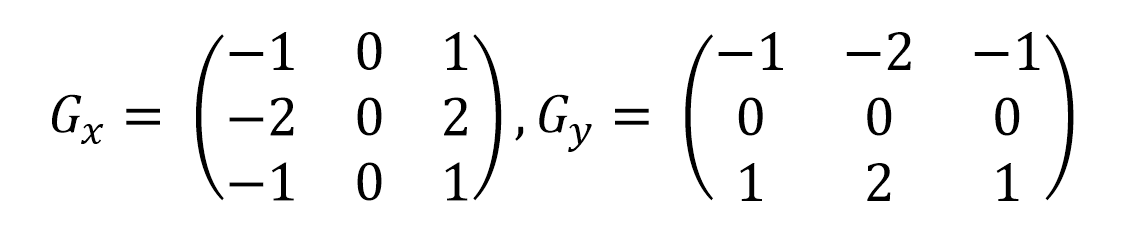
**Ход работы:**

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

Был реализован метод, читающий изображение сразу в градациях серого при помощи флага cv2.IMREAD\_GRAYSCALE функции cv2.imread и возвращающий результат выполнения функции cv2.GaussianBlur с заданным в параметрах размером ядра, а значение сигма задано равным нулю, чтобы библиотека OpenCV сама его задала вместо пользователя.

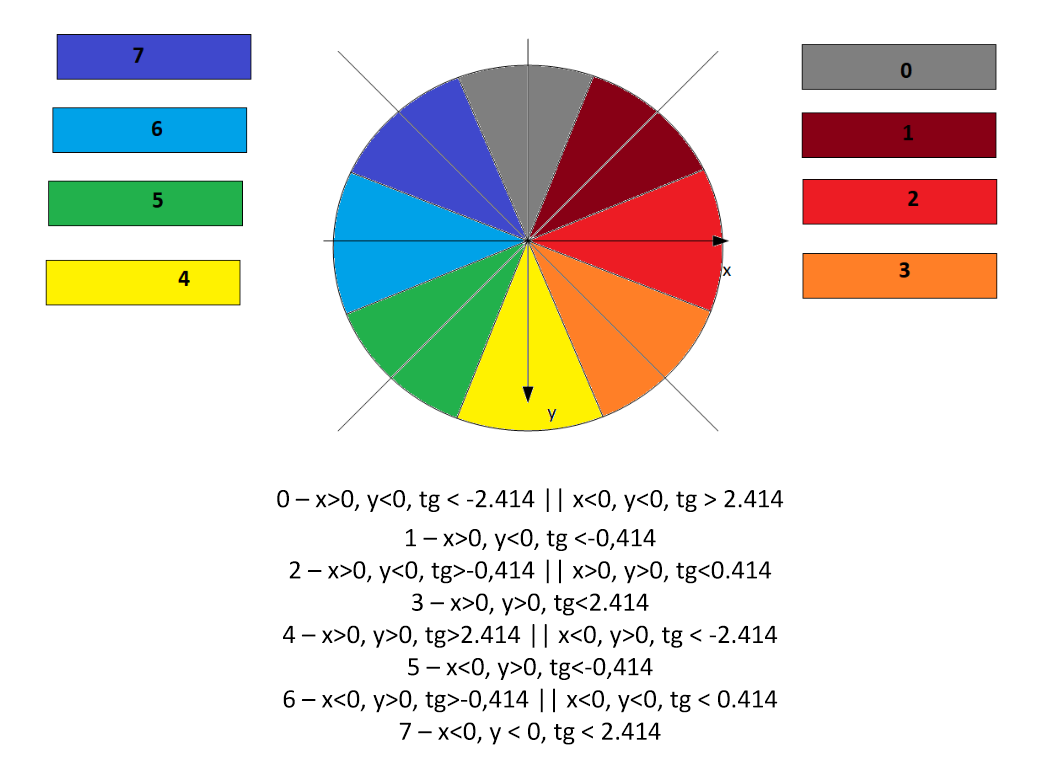
**Задание 2.** Модифицировать построенный метод так, чтобы в результате вычислялось и выводилось на экран две матрицы – матрица значений длин и матрица значений углов градиентов всех пикселей изображения.

Сперва была реализована функция свёртки, взятая из прошлой лабораторной работы. Затем создана функция edge\_detection, которая будет выполнять шаги алгоритма Канни. В начале этой функции были определены матрицы оператора Собеля для вычисления частных производных:



При применении свёртки ядром Gx вычисляется частная производная по x, при Gy – по y. Величина градиента вычисляется как длина вектора: . Для ускорения работы программы, вместо циклов, проходящих по пикселям, используются векторизованные операции NumPy на всей матрице, например: grad\_len = np.sqrt(np.add(np.square(gx), np.square(gy))), где gx и gy – матрицы частных производных, а grad\_len – матрица длин градиентов.

Затем вычисляется тангенс углов градиентов как частное Gy / Gx. После этого переходим к подавлению немаксимумов. Суть в том, что для каждого пикселя вычисляется округлённый угол градиента. Угол округляется, 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. Угол вычисляется в функции angle\_num, которая возвращает номер угла от 0 до 7 согласно следующему набору правил:



Чтобы угол был определён при x = 0 и y = 0, строгие неравенства сравнения x и y с нулём заменены на нестрогие.

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

Исходя из номера угла выбираются два соседних пикселя, находящихся на линии вдоль вектора градиента. Если значение градиента текущего пикселя больше, чем у этих двух соседних, то считается, что он является границей, и, следовательно, закрашивается белым.

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

Процесс двойной пороговой фильтрации заключается в том, что для каждого пикселя, определённого как граница, сравнивается величина его градиента с двумя пороговыми значениями. Это позволяет отфильтровать шумы и сохранить только настоящие границы объектов.

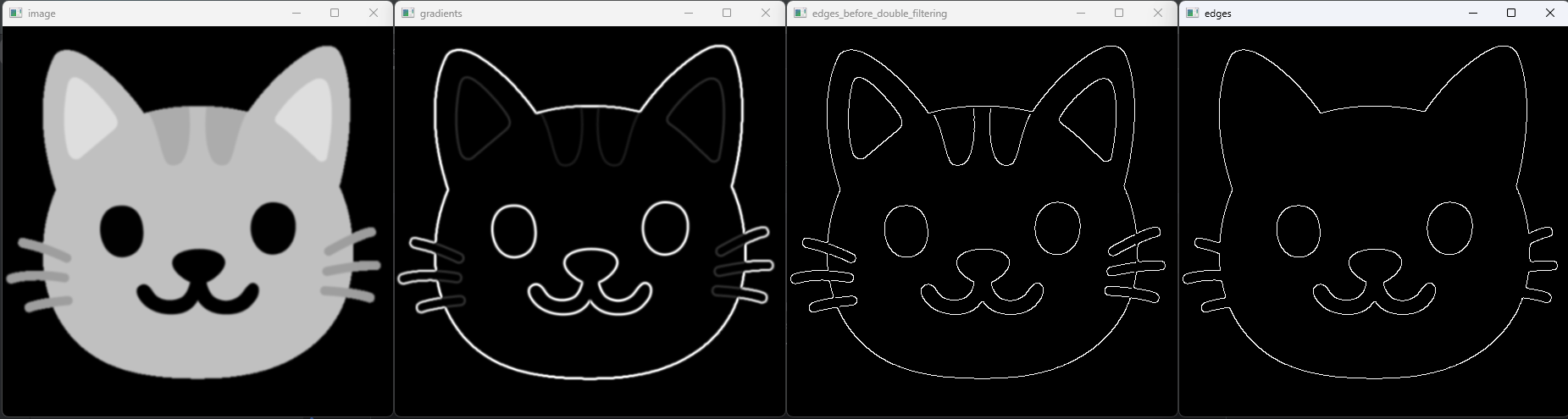
В качестве нижнего и верхнего порогового значения используется максимальное значение градиента на изображении, умноженное на соответствующий параметр, задаваемый пользователем (может принимать значения от 0 до 1). Параметр low должен быть меньше параметра high.

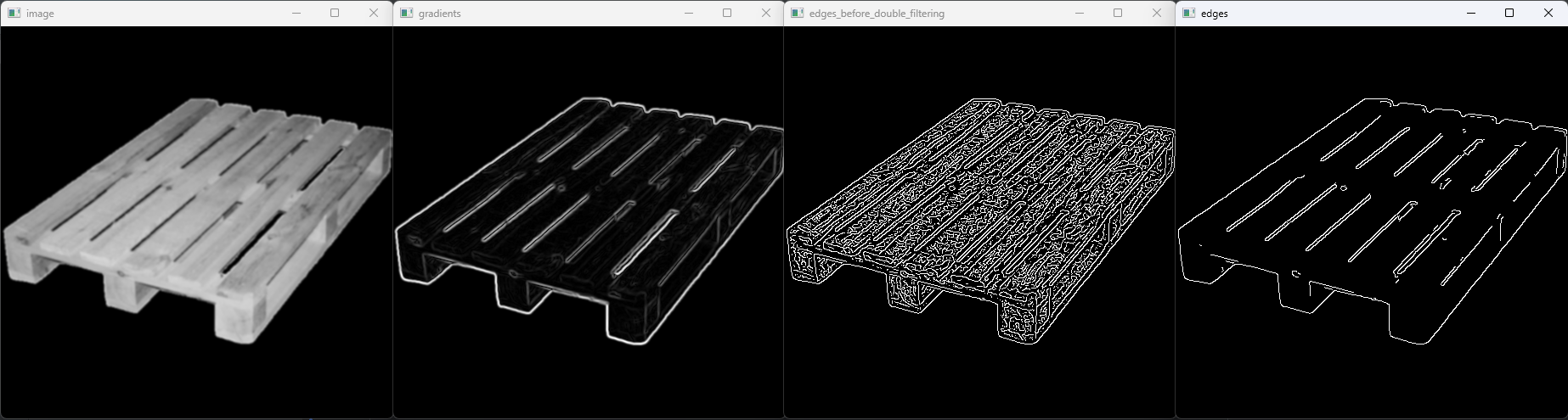
Фильтрация пикселей происходит следующим образом: если значение градиента меньше нижней границы, то пиксель не граница, если значение градиента выше верхней границы, то пиксель точно граница. После такого фильтра останутся пиксели, значение градиента которых заключено между границами. Для них воспользуемся следующим предположением: если пиксель – это граница, то он не может быть отдельной границей, рядом должен быть еще пиксель с границей. Исходя из этого предположения добавим проверку на то, что рядом с границей есть другая граница, для чего необходимо проверить 8 пикселей вокруг заданного.

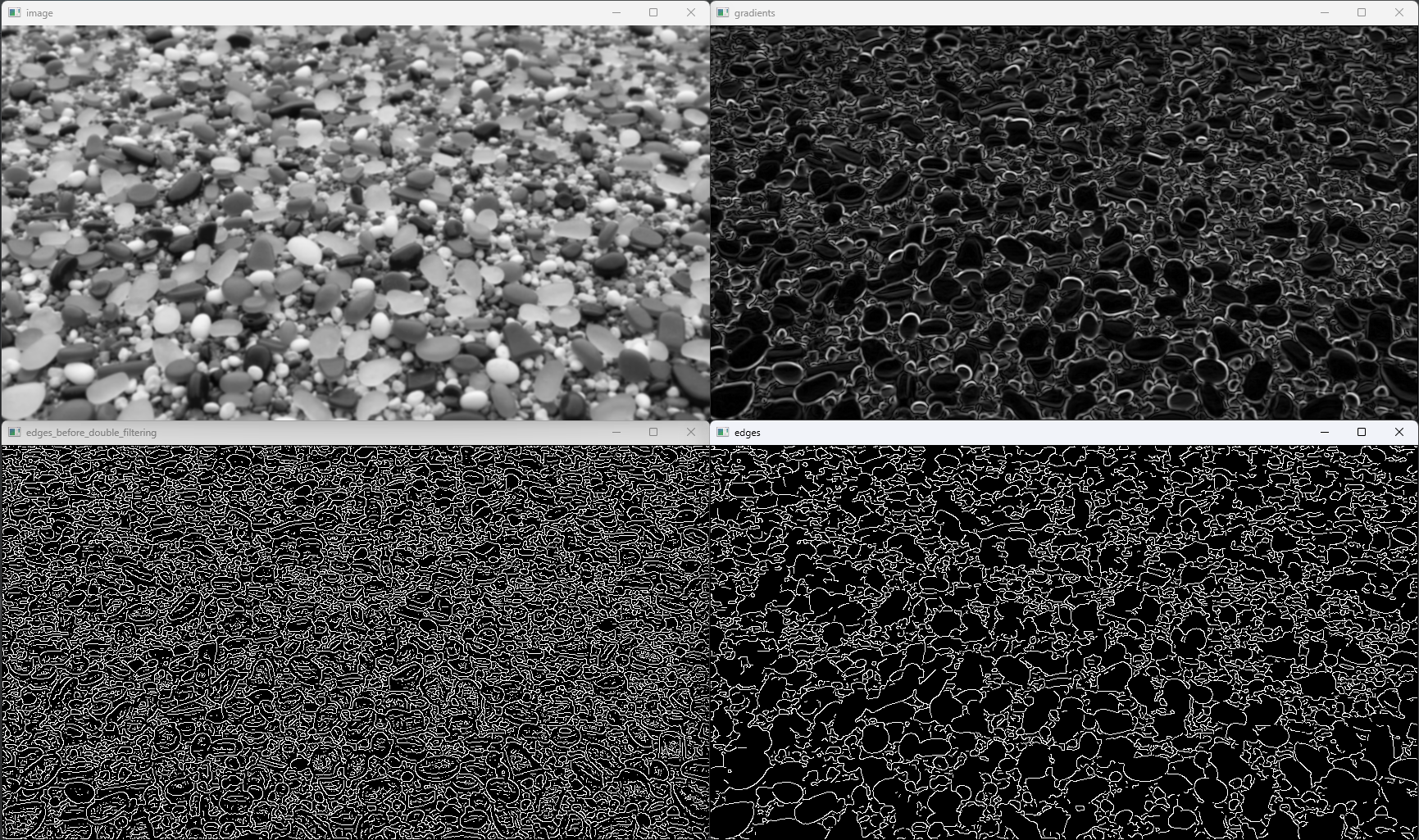
**Задание 5.** Провести опыты для различных параметров размытия и различных пороговых значений градиента, определить наилучшие параметры для Вашего изображения. Показать преподавателю значения параметров и результат работы на следующем занятии.

После тестирования алгоритма на разных значениях размытия и пороговых значений на разных изображениях были выбраны следующие значения: размер ядра размытия = 5, нижний порог = 4% от максимальной величины градиента, верхний порог = 20% от максимальной величины градиента. Ниже представлены результаты работы алгоритма Канни с указанными входными параметрами. По порядку: чёрно-белое изображение после размытия, градиенты, результат подавления немаксимумов, результат двойной фильтрации (итоговый результат).

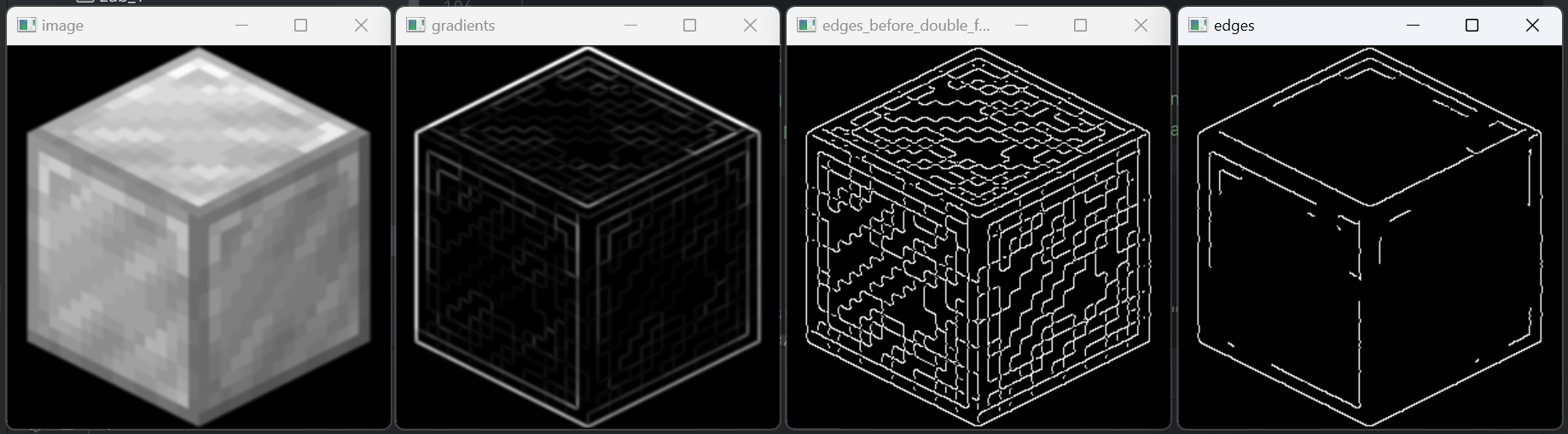












**Задание 6.** Реализовать алгоритм Канни на другом языке программирования.

Алгоритм Канни был реализован на C++. Для работы с изображением была использована библиотека OpenCV для C++, для сборки использована система CMake. Созданная программа принимает следующие аргументы: canny.exe <Путь к изображению> <Размер ядра размытия> <Нижний пороговый процент> <Верхний пороговый процент> <Масштаб изображения>.

**Листинг программы**

Файл Lab\_4.py

import cv2

import numpy as np

def preprocess(filepath: str, blur\_kernel\_size: int, scale: float = 1):

    img = cv2.imread(filepath, cv2.IMREAD\_GRAYSCALE)

    if scale != 1:

        img = cv2.resize(img, None, fx=scale, fy=scale)

    return cv2.GaussianBlur(img, (blur\_kernel\_size, blur\_kernel\_size), 0)

def conv(grayscale\_image: np.ndarray, kernel: np.ndarray):

    result = np.zeros\_like(grayscale\_image, np.int32)

    h, w = grayscale\_image.shape[:2]

    kernel\_size = kernel.shape[0]

    half\_kernel\_size = int(kernel\_size // 2)

    for y in range(half\_kernel\_size, h - half\_kernel\_size):

        for x in range(half\_kernel\_size, w - half\_kernel\_size):

            val = 0

            for k in range(-half\_kernel\_size, half\_kernel\_size + 1):

                for l in range(-half\_kernel\_size, half\_kernel\_size + 1):

                    val += grayscale\_image[y + k, x + l] \* kernel[half\_kernel\_size + k, half\_kernel\_size + l]

            result[y, x] = val

    return result

def angle\_num(x, y, tg):

    if (x >= 0 and y <= 0 and tg < -2.414) or (x <= 0 and y <= 0 and tg > 2.414):

        return 0

    elif x >= 0 and y <= 0 and tg < -0.414:

        return 1

    elif (x >= 0 and y <= 0 and tg > -0.414) or (x >= 0 and y >= 0 and tg < 0.414):

        return 2

    elif x >= 0 and y >= 0 and tg < 2.414:

        return 3

    elif (x >= 0 and y >= 0 and tg > 2.414) or (x <= 0 and y >= 0 and tg < -2.414):

        return 4

    elif x <= 0 and y >= 0 and tg < -0.414:

        return 5

    elif (x <= 0 and y >= 0 and tg > -0.414) or (x <= 0 and y <= 0 and tg < 0.414):

        return 6

    elif x <= 0 and y <= 0 and tg < 2.414:

        return 7

def edge\_detection(grayscale\_image: np.ndarray, low\_percent: float = None, high\_percent: float = None, show\_grad: bool = False, show\_nms: bool = False):

    # Sobel

    ker\_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])

    ker\_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    gx = conv(grayscale\_image, ker\_x)

    gy = conv(grayscale\_image, ker\_y)

    # Calculate gradient lengths and tangents

    grad\_len = np.sqrt(np.add(np.square(gx), np.square(gy)))

    max\_grad\_len = grad\_len.max()

    if show\_grad:

        cv2.imshow('gradients', (grad\_len / max\_grad\_len \* 255).astype(np.uint8))

    tg = np.divide(gy, gx)

    tg = np.nan\_to\_num(tg)

    print(grad\_len)

    print(tg)

    # Non-maximum suppression

    edges = np.zeros\_like(grayscale\_image)

    for y in range(1, edges.shape[0] - 1):

        for x in range(1, edges.shape[1] - 1):

            angle = angle\_num(gx[y, x], gy[y, x], tg[y, x])

            if angle == 0 or angle == 4:

                neighbor1 = [y - 1, x]

                neighbor2 = [y + 1, x]

            elif angle == 1 or angle == 5:

                neighbor1 = [y - 1, x + 1]

                neighbor2 = [y + 1, x - 1]

            elif angle == 2 or angle == 6:

                neighbor1 = [y, x + 1]

                neighbor2 = [y, x - 1]

            elif angle == 3 or angle == 7:

                neighbor1 = [y + 1, x + 1]

                neighbor2 = [y - 1, x - 1]

            else:

                raise Exception('Угол не определён')

            if grad\_len[y, x] >= grad\_len[neighbor1[0], neighbor1[1]] and grad\_len[y, x] > grad\_len[neighbor2[0], neighbor2[1]]:

                edges[y, x] = 255

    if show\_nms:

        cv2.imshow('edges\_before\_double\_filtering', edges)

    # Double threshold filtering

    max\_grad\_len = grad\_len.max()

    low\_level = int(max\_grad\_len \* low\_percent)

    high\_level = int(max\_grad\_len \* high\_percent)

    for y in range(edges.shape[0]):

        for x in range(edges.shape[1]):

            if edges[y, x] > 0:

                if grad\_len[y, x] < low\_level:

                    edges[y, x] = 0

                elif grad\_len[y, x] < high\_level:

                    keep = False

                    for neighbor\_y in (y - 1, y, y + 1):

                        for neighbor\_x in (x - 1, x, x + 1):

                            if neighbor\_y != y or neighbor\_x != x:

                                if edges[neighbor\_y, neighbor\_x] > 0 and grad\_len[neighbor\_y, neighbor\_x] >= high\_level:

                                    keep = True

                    if not keep:

                        edges[y, x] = 0

    return edges

if \_\_name\_\_ == '\_\_main\_\_':

    folder = '../images/'

    images = ['block.png', 'cat.jpg', 'desert.bmp', 'emoji.png',

              'enderman.png', 'error.png', 'poddon.png', 'seaglass.webp']

    select = 7

    scale = 1

    blur = 5

    low\_percent = 0.04

    high\_percent = 0.2

    # image = preprocess(r"cpp\Release\Static-assets-upload1811341065938287602.png", blur, scale)

    image = preprocess(folder + images[select - 1], blur, scale)

    edges = edge\_detection(image, low\_percent, high\_percent, show\_grad=True, show\_nms=True)

    cv2.imshow('image', image)

    cv2.imshow('edges', edges)

    # edges\_library = cv2.Canny(image, 100, 200)

    # cv2.imshow('edges\_library', edges\_library)

    cv2.waitKey(0)

    cv2.destroyAllWindows()

    # cv2.imwrite("C:/Users/ARTEZON/Desktop/out.jpg", edges)

Файл canny.cpp

#include <cstdio>

#include <cmath>

#include <opencv2/opencv.hpp>

cv::Mat conv(const cv::Mat &img, const cv::Mat &kernel) {

    cv::Mat result = cv::Mat::zeros(2, img.size, CV\_32SC1);

    int h = img.size[0], w = img.size[1];

    int kernelSize = kernel.size[0];

    int halfKernelSize = kernelSize / 2;

    for (int y = halfKernelSize; y < h - halfKernelSize; y++) {

        for (int x = halfKernelSize; x < w - halfKernelSize; x++) {

            double val = 0;

            for (int k = -halfKernelSize; k <= halfKernelSize; k++) {

                for (int l = -halfKernelSize; l <= halfKernelSize; l++) {

                    val += img.at<uint8\_t>(y + k, x + l) \* kernel.at<int8\_t>(halfKernelSize + k, halfKernelSize + l);

                }

            }

            result.at<int32\_t>(y, x) = (int32\_t)val;

        }

    }

    return result;

}

uint8\_t angle\_num(int32\_t x, int32\_t y) {

    double tan = (double)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

        return -1;

}

void cannyAlgorithm(cv::Mat &img, cv::Mat &grad, cv::Mat &nms, cv::Mat &edges, float lowerThresholdPercent, float upperThresholdPercent) {

    // Sobel

    const cv::Mat kerX = (cv::Mat\_<int8\_t>(3, 3) << -1, 0, 1, -2, 0, 2, -1, 0, 1);

    const cv::Mat kerY = (cv::Mat\_<int8\_t>(3, 3) << -1, -2, -1, 0, 0, 0, 1, 2, 1);

    cv::Mat gX = conv(img, kerX);

    cv::Mat gY = conv(img, kerY);

    // Calculate gradient lengths and tangents

    cv::Mat gradLen = cv::Mat(2, img.size, CV\_32FC1);

    for (int y = 0; y < gradLen.size[0]; y++) {

        for (int x = 0; x < gradLen.size[1]; x++) {

            gradLen.at<float>(y, x) = (float)sqrt(pow(gX.at<int32\_t>(y, x), 2) + pow(gY.at<int32\_t>(y, x), 2));

        }

    }

    double maxGradLen;

    cv::minMaxIdx(gradLen, nullptr, &maxGradLen, nullptr, nullptr);

    grad = gradLen \* 255 / maxGradLen;

    grad.convertTo(grad, CV\_8UC1);

    // Non-maximum suppression

    edges = cv::Mat::zeros(2, img.size, img.type());

    for (int y = 1; y < edges.size[0] - 1; y++) {

        for (int x = 1; x < edges.size[1] - 1; x++) {

            uint8\_t angle = angle\_num(gX.at<int32\_t>(y, x), gY.at<int32\_t>(y, x));

            int neighbor1[2] = {y, x};

            int neighbor2[2] = {y, x};

            if (angle == 0 || angle == 4) {

                neighbor1[0] = y - 1;

                neighbor1[1] = x;

                neighbor2[0] = y + 1;

                neighbor2[1] = x;

            }

            else if (angle == 1 || angle == 5) {

                neighbor1[0] = y - 1;

                neighbor1[1] = x + 1;

                neighbor2[0] = y + 1;

                neighbor2[1] = x - 1;

            }

            else if (angle == 2 || angle == 6) {

                neighbor1[0] = y;

                neighbor1[1] = x + 1;

                neighbor2[0] = y;

                neighbor2[1] = x - 1;

            }

            else if (angle == 3 || angle == 7) {

                neighbor1[0] = y + 1;

                neighbor1[1] = x + 1;

                neighbor2[0] = y - 1;

                neighbor2[1] = x - 1;

            }

            if (gradLen.at<float>(y, x) >= gradLen.at<float>(neighbor1[0], neighbor1[1]) && gradLen.at<float>(y, x) > gradLen.at<float>(neighbor2[0], neighbor2[1])) {

                edges.at<uint8\_t>(y, x) = 255;

            }

        }

    }

    edges.copyTo(nms);

    // Double threshold filtering

    int lowLevel = int(maxGradLen \* lowerThresholdPercent);

    int highLevel = int(maxGradLen \* upperThresholdPercent);

    for (int y = 1; y < edges.size[0] - 1; y++) {

        for (int x = 1; x < edges.size[1] - 1; x++) {

            if (edges.at<uint8\_t>(y, x) > 0) {

                if (gradLen.at<float>(y, x) < lowLevel) {

                    edges.at<uint8\_t>(y, x) = 0;

                }

                else if (gradLen.at<float>(y, x) < highLevel) {

                    bool keep = false;

                    for (int offsetY = -1; offsetY <= 1; offsetY++) {

                        for (int offsetX = -1; offsetX <= 1; offsetX++) {

                            if (offsetY != 0 || offsetX != 0) {

                                if (edges.at<uint8\_t>(y + offsetY, x + offsetX) > 0 && gradLen.at<float>(y + offsetY, x + offsetX) >= highLevel) {

                                    keep = true;

                                }

                            }

                        }

                    }

                    if (!keep) edges.at<uint8\_t>(y, x) = 0;

                }

            }

        }

    }

}

int main(int argc, char \*argv[]) {

    char \*path;

    int blurKernelSize;

    float lowerThresholdPercent;

    float upperThresholdPercent;

    float scale;

    if (argc == 6) {

        path = argv[1];

        blurKernelSize = atoi(argv[2]);

        lowerThresholdPercent = (float)atof(argv[3]);

        upperThresholdPercent = (float)atof(argv[4]);

        scale = (float)atof(argv[5]);

    }

    else {

        char \*prog\_name = strrchr(argv[0], '/') + 1;

        if (prog\_name - 1 == NULL)

            prog\_name = strrchr(argv[0], '\\') + 1;

        if (prog\_name - 1 == NULL)

            prog\_name = argv[0];

        printf("Usage:\n   %s <Path to image> <Kernel size of Gaussian blur> <Lower threshold percentage> <Upper threshold percentage> <Image scale>\n\nFor example:\n   %s image.jpg 5 0.04 0.2 1\n\nTo disable Gaussian blur, use 1 as kernel size.\n\n", prog\_name, prog\_name);

        return 0;

    }

    if (blurKernelSize < 1 || blurKernelSize % 2 == 0) {

        printf("Error: blur kernel size must be a positive odd integer (1 means no blur)\n");

        exit(1);

    }

    if (lowerThresholdPercent < 0 || lowerThresholdPercent > 1) {

        printf("Error: lower threshold percentage must be between 0 and 1\n");

        exit(1);

    }

    if (upperThresholdPercent < 0 || upperThresholdPercent > 1) {

        printf("Error: upper threshold percentage must be between 0 and 1\n");

        exit(1);

    }

    if (scale < 0.05 || scale > 5) {

        printf("Error: image scale must be between 0.05 and 5\n");

        exit(1);

    }

    cv::Mat img;

    img = cv::imread(path, cv::IMREAD\_GRAYSCALE);

    if (img.empty()) {

        printf("Error: image not loaded\n");

        exit(1);

    }

    if (blurKernelSize > 1) {

        GaussianBlur(img, img, cv::Size(blurKernelSize, blurKernelSize), 0.0);

    }

    if (scale != 1) {

        resize(img, img, cv::Size(), scale, scale);

    }

    cv::Mat grad, nms, edges;

    cannyAlgorithm(img, grad, nms, edges, lowerThresholdPercent, upperThresholdPercent);

    cv::imshow("Preprocessed Image", img);

    cv::imshow("Gradients", grad);

    cv::imshow("NMS result", nms);

    char \*canny\_title = new char[128];

    std::snprintf(canny\_title, 128, "Edges (image size: %dx%d, blur kernel size: %d, lower threshold: %g%%, upper threshold: %g%%", img.size[1], img.size[0], blurKernelSize, lowerThresholdPercent \* 100, upperThresholdPercent \* 100);

    cv::imshow(canny\_title, edges);

    delete[] canny\_title;

    cv::waitKey(0);

}

**Ответы на вопросы**

1. Опишите, в чём заключается задача выявления контуров, и области применения этой задачи.

Задача выявления контуров заключается в обнаружении границ объектов на изображении. Эти границы представляют собой совокупности пикселей, в которых сильно меняется яркость. Область применения включает в себя компьютерное зрение, обработку изображений и машинное обучение. Выявление контуров помогает выделить объекты на изображении, обнаружить их формы и структуры, что необходимо для решения задач идентификации объекта, распознавания текста на фрагментах изображения, детекции движения в видеопотоке, обнаружения объекта в видеопотоке, трекинга объекта и других задач.

2. На чём основываются градиентные методы выявления контуров?

Градиентные методы выявления контуров основаны на вычислении градиента функции яркости в каждой точке изображения. Градиент показывает, как сильно меняется интенсивность пикселей и в каких направлениях. На границе объекта происходит сильный скачок яркости изображения, поэтому высокие значения градиента обычно соответствуют контурам объектов.

3. Опишите основные этапы алгоритма Канни.

Алгоритм Канни состоит из следующих шагов:

1) построение чёрно-белого изображения;

2) применение размытия по Гауссу для подавления шумов;

3) вычисление градиентов функции яркости, используя, например, оператор Собеля;

4) подавление немаксимумов для выделения локальных максимумов в градиенте (если значение градиента пикселя больше соседних, то пиксель определяется как граничный, иначе значение пикселя подавляется);

5) двойная пороговая фильтрация (сравнение величин градиентов с двумя пороговыми значениями).

4. Что такое градиент пикселя изображения и какие могут возникнуть проблемы с его вычислением? Объясните почему они возникают?

Градиент пикселя изображения представляет собой вектор, показывающий направление наибольшего изменения интенсивности и скорость этого изменения. Что касается проблем: в случае, если рядом стоят пиксели одинакового цвета, то длина градиента равна нулю и его угол, соответственно, не определён. Если x=0 и y=0, то при делении y/x будет NaN. Если только x=0, то будет бесконечность. Впрочем, всё и так хорошо работает. Другая проблема, это что на граничных пикселях изображения невозможно вычислить градиент без дополнения изображения, потому что матрицы свёртки имеют размерность 3x3. Также на значения градиентов влияет шум, поэтому на начальном этапе используется фильтрация для его подавления.

5. Опишите принцип работы оператора Собеля и особенности его использования в алгоритме Канни.

Оператор Собеля – это оператор, используемый для вычисления градиента яркости пикселей. Он основан на свёртке изображения с двумя ядрами (одним для вычисления горизонтальной частной производной и другим для вертикальной).

6. Какие операторы возможно использовать вместо оператора Собеля, найдите самостоятельно и опишите, в чем их отличие от оператора Собеля.

Кроме оператора Собеля, для вычисления градиента можно использовать операторы Робертса, Прюитта, Кирша.

7. Каким образом и для чего осуществляется округление угла градиента? Опишите на примере матрицы изображения, зачем хранить угол и для чего его округлять. Поясните на чертеже, как происходит округление.

Округление угла градиента необходимо для определения направления градиента в одном из восьми предопределенных направлений (0, 45, 90, 135 градусов и их обратные направления), так как у каждого внутреннего пикселя имеется 8 соседних. Например, если угол градиента составляет 30 градусов, он округляется до ближайшего угла из предопределенных направлений, например, 45 градусов.

8. Опишите, в чём суть этапа подавление немаксимумов, покажите роль угла градиента в данном этапе.

Этап подавления немаксимумов заключается в сравнении значений градиента пикселя с его соседями в направлении градиента. Только локальные максимумы градиента сохраняются, а все остальные пиксели подавляются. Угол градиента играет роль в определении направления проверки для каждого пикселя.

9. Опишите, в чём принцип двойной пороговой фильтрации.

Двойная пороговая фильтрация – это метод, который использует два порога (нижний и верхний) для классификации пикселей на пиксели границ и не-границ. Пиксели с градиентом выше верхнего порога считаются пикселями границ, пиксели с градиентом ниже нижнего порога считаются не-границами, а пиксели с градиентом между нижним и верхним порогами являются «пикселями-кандидатами». Далее выполняется процесс связывания границ (отсеивание пикселей-кандидатов, которые не соединены с пикселями границ)

для окончательного выделения контуров на изображении.