

ГУАП

КАФЕДРА № 44

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

доц., канд.техн.наук

должность, уч. степень, звание

подпись, дата

А.М.Сергеев

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

РАЗРАБОТКА WINDOWS-ПРИЛОЖЕНИЯ

по курсу: ЦИФРОВАЯ ОБРАБОТКА ИЗОБРАЖЕНИЙ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

4143

подпись, дата

Е.Д.Тегай

инициалы, фамилия

Санкт-Петербург 2025

Индивидуальное задание

Содержимое индивидуального задания продемонстрировано на рисунке 1.

16. Выделение контура фильтрами Собеля, Превитта, Кирша и Уоллеса.

Рисунок 1 – Индивидуальное задание

Теоретические положения, используемые при выполнении работы

1. Гистограмма. Гистограмма – график распределения яркости пикселей на изображении, где по оси абсциссы отложены значения яркости (от 0 до 255), а по оси ординат – количество пикселей для каждого уровня яркости. Данный график полезен для анализа контраста и освещённости и выявления пере- или недоэкспонированных областей. При построении оси ординат в данном приложении также используется логарифмическая шкала для наглядности, так как распределение пикселей может быть неравномерным (например, при большом количестве темных или светлых участков на исходном изображении).

2. Фильтры выделения контуров. Разработанное приложение оперирует различными 4-мя фильтрами: Собеля, Превитта, Кирша и Уоллеса. Все они помогают находить границы объектов на изображении. Рассмотрим первый фильтр. Принцип его работы таков, что он вычисляет градиент яркости с помощью свёртки изображения с матрицами 3×3 или 5×5 (горизонтальной и вертикальной). Чем сильнее перепад, тем ярче контур. Особенность этого фильтра такова, что он даёт плавные, слегка размытые границы, но устойчив к шуму.

Следующим фильтром является фильтр Превитта. Данный фильтр похож на предыдущий, но использует более простые веса в матрице свёртки. Он менее точный, чем фильтр Собеля, но быстрее вычисляется. Поэтому он лучше подходит для предварительного анализа.

Далее рассмотрим фильтр Кирша. Он проверяет 8 возможных направлений градиента и выбирает самое сильное изменение. Особенность его

заключается в том, что он хорошо выделяет диагональные и сложные контуры, однако он чувствителен к шуму.

Последним используемым фильтром является фильтр Уоллеса. Он не просто ищет контуры, а усиливает локальный контраст. То есть он подстраивается под разные участки изображения – затемнённые области делает ярче, а пересвеченные – мягче. Данный фильтр полезен для улучшения детализации, но требует настройки параметров (например, силы усиления).

Разница между рассматриваемыми фильтрами заключается в точности, скорости и устойчивости к шумам.

3. Методы оценки результатов. В реализованном приложении доступна возможность сравнивать гистограммы для загруженного и обработанного изображений. Так, например, можно заметить сдвиги пиков яркости. Если пик сместился влево, то изображение стало темнее, а если вправо – светлее. Также можно увидеть изменение дисперсии (разброса значений). Если гистограмма широкая, то это означает, что существует высокий контраст. А если она узкая, то контраст низкий, то есть пиксели близки по яркости. При тестировании можно заметить, что фильтры контуров часто увеличивают дисперсию, так как они усиливают перепады яркости.

Гистограммы дают числовые данные, но итоговую оценку можно и даже лучше проводить на глаз. Хороший фильтр оставляет границы объектов чёткими, без размытия. Например, фильтр Собеля даёт мягкие контуры, а фильтр Кирша – более резкие, но с шумами.

4. Математическая база. В основе алгоритмов обработки изображений лежит несколько математических концепций. Основным инструментом для работы с фильтрами является операция свёртки. Она выполняется путём поэлементного умножения матрицы изображения на ядро фильтра (маску) с последующим суммированием результатов. Например, в фильтре Собеля, как было уже упомянуто выше, используются две матрицы 3×3 (или 5×5), где каждый пиксель результирующего изображения вычисляется как взвешенная

сумма соседних пикселей. Такая операция и позволяет выделять особенности изображения (границы, углы и тому подобное).

Помимо линейной алгебры используются также и статистические методы: расчёт среднего значения яркости, который показывает общий уровень освещённости; вычисление дисперсии, которое характеризует контрастность изображения; логарифмирование гистограммы, которое используется для визуализации распределения яркостей, когда значения имеют большой разброс.

Эти математические основы позволяют эффективно обрабатывать изображения, анализировать их статистические характеристики, преобразовывать между разными цветовыми пространствами и строить наглядные гистограммы распределения яркостей.

Описание процесса выполнения работы

В качестве среды разработки искомого Windows-приложения была выбрана Visual Studio 2019. Разработка началась с реализации интерфейса. Структура интерфейса такова: большую часть области окна занимают два графических объекта-поля, куда выводится исходное и обработанное изображения. Сопутствуют также соответствующие кнопки для сохранения результата, загрузки изображения, создания тестового образца и применения выбранного фильтра. Для выбора фильтра используются радиокнопки (переключатели). В качестве дополнительной возможности приложения в рамках индивидуального задания была выбрана демонстрация по желанию пользователя гистограммы. Графическим объектом, через который пользователь может дать понять программе о необходимости в показе гистограммы, является чекбокс. Последними элементами интерфейса являются текстовые поля с сопутствующими подписями, которые нужны для задания параметров в рамках определённых фильтров пользователем. Если пользователь ничего не вводит в эти поля, фильтр применяется с заданными значениями по умолчанию. Итоговый внешний вид интерфейса продемонстрирован на рисунке 2.

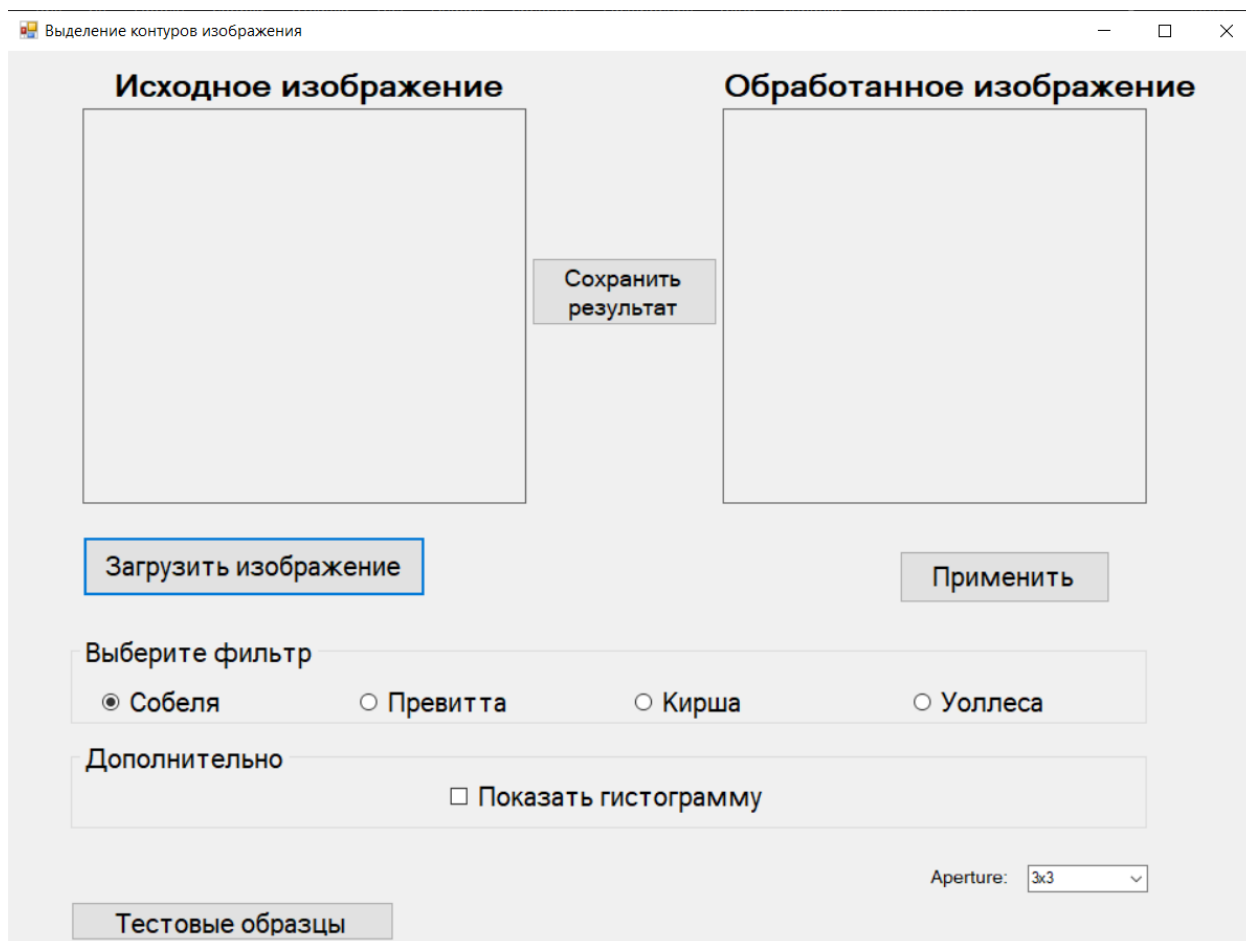


Рисунок 2 – Внешний вид приложения

Следует отметить, что итоговый проект состоит из двух форм. Первая форма – основная – описана выше, а вторая форма необходима для вывода гистограмм. Внутри этой формы не расположены никакие графические объекты, она является лишь окном для рисования гистограмм и вывода пользователю получившихся результатов.

Приступим к рассмотрению разработанной программы. Основным файлом, где прописывалась логика, является **Form1.cs**. Эта программа и представляет собой искомое графическое приложение для обработки изображений с использованием различных фильтров выделения контуров и анализа результатов. Основной функционал включает загрузку изображений, применение фильтров, отображение результатов и построение гистограмм.

Программу можно разделить на несколько блоков. Одним из первых является блок с описанием загрузки изображений. Внутри него, собственно, реализована функция открытия файлов изображений стандартных форматов.

При загрузке также выполняется проверка корректности файла. Загруженное изображение отображается в специальном элементе просмотра. Далее идёт описание системы фильтрации. Как уже упоминалось ранее, приложение поддерживает 4 основных типа фильтров для выделения контуров. Для первых трёх фильтров доступен выбор размера области обработки. Последний фильтр (Уоллеса) имеет дополнительные настраиваемые параметры. Следующим блоком является преобразование изображений. Внутри него реализована функция перевода цветного изображения в оттенки серого. Для каждого фильтра предусмотрена своя алгоритмическая реализация. Обработка выполняется попиксельно с учётом соседних областей. Далее идёт описание визуализации результатов. Для этого реализовано окно для построения и сравнения гистограмм яркости обоих изображений. Эти гистограммы строятся в логарифмическом масштабе для наглядности. Также на них отображается статистическая информация о распределении яркостей. Завершающим блоком является блок с описанием дополнительных функций. Например, генерация тестовых изображений с различными характеристиками, возможность сохранения обработанных изображений, контекстное меню для быстрого доступа к операциям и отображение информации о пикселях при наведении курсора.

Также есть файл **Form1.Designer.cs**, внутри которого код представляет собой сгенерированный файл дизайнера для формы приложения. Форма содержит элементы управления, позволяющие загружать изображения, применять к ним различные фильтры для выделения контуров, а также настраивать параметры обработки.

Код, описанный в файле **HistogramForm.cs**, представляет собой форму для отображения гистограмм яркости изображений. Форма позволяет визуализировать распределение яркости пикселей как для исходного, так и для обработанного изображений, если оно предоставлено. Для каждого изображений вычисляются: максимальная яркость, общее количество пикселей и средняя яркость. Статистика отображается рядом с

соответствующей гистограммой. Гистограммы рисуются в виде столбцов, где высота столбца соответствует количеству пикселей определённой яркости. Оси также подписаны: ось ординат - количество пикселей в логарифмическом масштабе, а ось абсцисс - значение яркости (0-255). Для удобства сравнения исходное изображение выделено синим цветом, тогда как обработанное – оранжевым.

Последним файлом является **HistogramForm.Designer.cs**, который содержит в себе автоматически сгенерированный код дизайнера для предыдущей формы. Здесь определяется базовая структура формы и её начальные настройки. Листинги всех рассмотренных файлов будут приведены ниже.

Рассмотрим более подробно программную реализацию алгоритмов для каждого фильтра. Фильтр Собеля вычисляет градиент яркости изображений в 2 направлениях – горизонтальном и вертикальном) с помощью специальных матриц свёртки. Горизонтальный и вертикальный градиенты вычисляются как сумма произведений значений пикселей окрестности на соответствующие элементы матриц. Результирующее изображение показывает резкие изменения яркости, которые соответствуют границам объектов. Фильтр Кирша использует 8 направленных матриц (по одной на каждое из основных направлений: север, юг, запад, восток и диагонали). Результирующим значением является максимальный отклик среди всех направлений. Фильтр Превитта аналогичен фильтру Собеля, но использует другие матрицы свёртки, которые менее чувствительны к шумам, но хуже выделяют плавные границы. Особенностью также является меньший вес центральных пикселей по сравнению с фильтром Собеля, что делает его менее чувствительным к мелким деталям. Фильтр Уоллеса является локально адаптивным. Он усиливает контраст в областях с низкой дисперсией, сохраняя детали в областях с высокой дисперсией. Основан на статистике (среднее и дисперсия яркости в окрестности). Фильтры успешно выделяют контуры на изображении, но их эффективность зависит от выбранных параметров и характера исходного

изображения. Фильтр Уоллеса особенно полезен для изображений с неравномерным освещением.

Листинг основного файла Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using System.IO; // Для работы с файлами
using System.Drawing.Imaging; // Для BitmapData и ImageLockMode
using System.Runtime.InteropServices; // Для Marshal.Copy

namespace lab1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            InitializeContextMenu();

            // Инициализация ComboBox
            comboApertureSize.Items.AddRange(new string[] { "3×3", "5×5" });
            comboApertureSize.SelectedIndex = 0;

            // Подключаем обработчики событий
            radioButtonSobel.CheckedChanged += RadioButton_CheckedChanged;
            radioButtonPrewitt.CheckedChanged += RadioButton_CheckedChanged;
            radioButtonKirsch.CheckedChanged += RadioButton_CheckedChanged;
            radioButtonWallis.CheckedChanged += RadioButton_CheckedChanged;

            // Начальная настройка видимости
            UpdateFilterSettingsVisibility();

            pictureBoxOriginal.MouseMove += pictureBoxOriginal_MouseMove;
            pictureBoxProcessed.MouseMove += pictureBoxOriginal_MouseMove; // Тот же обработчик
        }

        private void InitializeContextMenu()
        {
            ContextMenuStrip menu = new ContextMenuStrip();

            ToolStripMenuItem saveItem = new ToolStripMenuItem("Сохранить изображение");
            saveItem.Click += (s, e) => btnSaveResult_Click(s, e);

            ToolStripMenuItem copyItem = new ToolStripMenuItem("Копировать в буфер");
            copyItem.Click += (s, e) =>
            {
                if (pictureBoxProcessed.Image != null)
                    Clipboard.SetImage(pictureBoxProcessed.Image);
            };
        }
    }
}
```



```

        menu.Items.Add(saveItem);
        menu.Items.Add(copyItem);

        pictureBoxProcessed.ContextMenuStrip = menu;
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        // Инициализация по умолчанию
        radioSobel.Checked = true;
    }

    private void btnLoadImage_Click(object sender, EventArgs e)
    {
        OpenFileDialog openFileDialog = new OpenFileDialog();
        openFileDialog.Filter = "Изображения|*.jpg;*.png;*.bmp";

        if (openFileDialog.ShowDialog() == DialogResult.OK)
        {
            try
            {
                pictureBoxOriginal.Image = new Bitmap(openFileDialog.FileName);
                labelStatus.Text = "Изображение загружено";
            }
            catch
            {
                MessageBox.Show("Невозможно открыть файл", "Ошибка",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }

    //----- РЕАЛИЗАЦИЯ ФИЛЬТРОВ -----
    private Bitmap ApplySobelFilter(Bitmap original, int apertureSize)
    {
        Bitmap result = new Bitmap(original.Width, original.Height);
        Bitmap grayscale = ToGrayscale(original);
        int radius = apertureSize / 2; // Для 3×3 radius=1, для 5×5 radius=2

        for (int y = radius; y < grayscale.Height - radius; y++)
        {
            for (int x = radius; x < grayscale.Width - radius; x++)
            {
                // Получаем окрестность (размер зависит от apertureSize)
                int[,] pixels = new int[apertureSize, apertureSize];
                for (int ky = -radius; ky <= radius; ky++)
                {
                    for (int kx = -radius; kx <= radius; kx++)
                    {
                        Color c = grayscale.GetPixel(x + kx, y + ky);
                        pixels[ky + radius, kx + radius] = c.R;
                    }
                }

                // Применяем оператор Собеля
                int gx = 0, gy = 0;

                if (apertureSize == 3)
                {
                    // Классический оператор Собеля 3×3
                    gx = (pixels[0, 2] + 2 * pixels[1, 2] + pixels[2, 2]) -
                        (pixels[0, 0] + 2 * pixels[1, 0] + pixels[2, 0]);

```

```

        gy = (pixels[2, 0] + 2 * pixels[2, 1] + pixels[2, 2]) -
              (pixels[0, 0] + 2 * pixels[0, 1] + pixels[0, 2]);
    }
    else if (apertureSize == 5)
    {
        // Оператор Собеля для 5×5
        gx =
            (1 * pixels[0, 4] + 2 * pixels[1, 4] + 0 * pixels[2, 4] - 2 * pixels[3, 4] - 1 * pixels[4, 4]) +
            (4 * pixels[0, 3] + 8 * pixels[1, 3] + 0 * pixels[2, 3] - 8 * pixels[3, 3] - 4 * pixels[4, 3]) +
            (6 * pixels[0, 2] + 12 * pixels[1, 2] + 0 * pixels[2, 2] - 12 * pixels[3, 2] - 6 * pixels[4, 2]) +
            (4 * pixels[0, 1] + 8 * pixels[1, 1] + 0 * pixels[2, 1] - 8 * pixels[3, 1] - 4 * pixels[4, 1]) +
            (1 * pixels[0, 0] + 2 * pixels[1, 0] + 0 * pixels[2, 0] - 2 * pixels[3, 0] - 1 * pixels[4, 0]);

        gy =
            (1 * pixels[4, 0] + 4 * pixels[4, 1] + 6 * pixels[4, 2] + 4 * pixels[4, 3] + 1 * pixels[4, 4]) -
            (1 * pixels[0, 0] + 4 * pixels[0, 1] + 6 * pixels[0, 2] + 4 * pixels[0, 3] + 1 * pixels[0, 4]);
    }

    int magnitude = (int)Math.Sqrt(gx * gx + gy * gy);
    magnitude = Math.Min(255, Math.Max(0, magnitude));
    result.SetPixel(x, y, Color.FromArgb(magnitude, magnitude, magnitude));
}
}
return result;
}

```

```

private Bitmap ToGrayscale(Bitmap original)
{
    Bitmap grayscale = new Bitmap(original.Width, original.Height);

    for (int y = 0; y < original.Height; y++)
    {
        for (int x = 0; x < original.Width; x++)
        {
            Color c = original.GetPixel(x, y);
            int gray = (int)(c.R * 0.3 + c.G * 0.59 + c.B * 0.11);
            grayscale.SetPixel(x, y, Color.FromArgb(gray, gray, gray));
        }
    }

    return grayscale;
}

```

```

private Bitmap ApplyPrewittFilter(Bitmap original, int apertureSize)
{
    Bitmap result = new Bitmap(original.Width, original.Height);
    Bitmap grayscale = ToGrayscale(original);

    int radius = apertureSize / 2; // Для 3×3 radius=1, для 5×5 radius=2

    for (int y = radius; y < grayscale.Height - radius; y++)
    {
        for (int x = radius; x < grayscale.Width - radius; x++)
        {
            // Получаем окрестность (размер зависит от apertureSize)
            int[,] pixels = new int[apertureSize, apertureSize];
            for (int ky = -radius; ky <= radius; ky++)
            {
                for (int kx = -radius; kx <= radius; kx++)
                {
                    Color c = grayscale.GetPixel(x + kx, y + ky);

```

```

        pixels[ky + radius, kx + radius] = c.R;
    }
}

// Оператор Превитта
int gx = 0, gy = 0;
if (apertureSize == 3)
{
    // Классический оператор Превитта 3×3
    gx = (pixels[0, 2] + pixels[1, 2] + pixels[2, 2]) -
        (pixels[0, 0] + pixels[1, 0] + pixels[2, 0]);

    gy = (pixels[2, 0] + pixels[2, 1] + pixels[2, 2]) -
        (pixels[0, 0] + pixels[0, 1] + pixels[0, 2]);
}
else if (apertureSize == 5)
{
    // Оператор Превитта для 5×5 (расширенная версия)
    gx =
        (1 * pixels[0, 4] + 1 * pixels[1, 4] + 0 * pixels[2, 4] - 1 * pixels[3, 4] - 1 * pixels[4, 4]) +
        (2 * pixels[0, 3] + 2 * pixels[1, 3] + 0 * pixels[2, 3] - 2 * pixels[3, 3] - 2 * pixels[4, 3]) +
        (2 * pixels[0, 2] + 2 * pixels[1, 2] + 0 * pixels[2, 2] - 2 * pixels[3, 2] - 2 * pixels[4, 2]) +
        (2 * pixels[0, 1] + 2 * pixels[1, 1] + 0 * pixels[2, 1] - 2 * pixels[3, 1] - 2 * pixels[4, 1]) +
        (1 * pixels[0, 0] + 1 * pixels[1, 0] + 0 * pixels[2, 0] - 1 * pixels[3, 0] - 1 * pixels[4, 0]);

    gy =
        (1 * pixels[4, 0] + 2 * pixels[4, 1] + 2 * pixels[4, 2] + 2 * pixels[4, 3] + 1 * pixels[4, 4]) -
        (1 * pixels[0, 0] + 2 * pixels[0, 1] + 2 * pixels[0, 2] + 2 * pixels[0, 3] + 1 * pixels[0, 4]);
}

int magnitude = (int)Math.Sqrt(gx * gx + gy * gy);
magnitude = Math.Min(255, Math.Max(0, magnitude));

result.SetPixel(x, y, Color.FromArgb(magnitude, magnitude, magnitude));
}
}
return result;
}

```

```

private Bitmap ApplyKirschFilter(Bitmap original, int apertureSize)
{
    Bitmap result = new Bitmap(original.Width, original.Height);
    Bitmap grayscale = ToGrayscale(original);
    int radius = apertureSize / 2;

    // Маски Кирша для 3×3
    int[,] kirschMasks3x3 = new int[8][,] {
        new int[3,3] { { 5, 5, 5 }, { -3, 0, -3 }, { -3, -3, -3 } }, // Север
        new int[3,3] { { -3, 5, 5 }, { -3, 0, 5 }, { -3, -3, -3 } }, // Северо-Восток
        new int[3,3] { { -3, -3, 5 }, { -3, 0, 5 }, { -3, -3, 5 } }, // Восток
        new int[3,3] { { -3, -3, -3 }, { -3, 0, 5 }, { -3, 5, 5 } }, // Юго-Восток
        new int[3,3] { { -3, -3, -3 }, { -3, 0, -3 }, { 5, 5, 5 } }, // Юг
        new int[3,3] { { -3, -3, -3 }, { 5, 0, -3 }, { 5, 5, -3 } }, // Юго-Запад
        new int[3,3] { { 5, -3, -3 }, { 5, 0, -3 }, { 5, -3, -3 } }, // Запад
        new int[3,3] { { 5, 5, -3 }, { 5, 0, -3 }, { -3, -3, -3 } } // Северо-Запад
    };
}

```

```

    // Маски Кирша для 5×5 (расширенные версии)
    int[,] kirschMasks5x5 = new int[8][,] {
        // Север
        new int[5,5] {
            { 5, 5, 5, 5, 5 },
            { 5, 5, 5, 5, 5 },

```

```

        {-3,-3, 0,-3,-3},
        {-3,-3,-3,-3,-3},
        {-3,-3,-3,-3,-3} },
    // Северо-Восток
    new int[5,5] {
        {-3,-3, 5, 5, 5},
        {-3,-3, 5, 5, 5},
        {-3,-3, 0, 5, 5},
        {-3,-3,-3,-3,-3},
        {-3,-3,-3,-3,-3} },
    // Восток
    new int[5,5] {
        {-3,-3,-3, 5, 5},
        {-3,-3,-3, 5, 5},
        {-3,-3, 0, 5, 5},
        {-3,-3,-3, 5, 5},
        {-3,-3,-3, 5, 5} },
    // Юго-Восток
    new int[5,5] {
        {-3,-3,-3,-3,-3},
        {-3,-3,-3,-3,-3},
        {-3,-3, 0, 5, 5},
        {-3,-3, 5, 5, 5},
        {-3,-3, 5, 5, 5} },
    // Юг
    new int[5,5] {
        {-3,-3,-3,-3,-3},
        {-3,-3,-3,-3,-3},
        {-3,-3, 0,-3,-3},
        { 5, 5, 5, 5, 5},
        { 5, 5, 5, 5, 5} },
    // Юго-Запад
    new int[5,5] {
        {-3,-3,-3,-3,-3},
        {-3,-3,-3,-3,-3},
        { 5, 5, 0,-3,-3},
        { 5, 5, 5,-3,-3},
        { 5, 5, 5,-3,-3} },
    // Запад
    new int[5,5] {
        { 5, 5,-3,-3,-3},
        { 5, 5,-3,-3,-3},
        { 5, 5, 0,-3,-3},
        { 5, 5,-3,-3,-3},
        { 5, 5,-3,-3,-3} },
    // Северо-Запад
    new int[5,5] {
        { 5, 5, 5,-3,-3},
        { 5, 5, 5,-3,-3},
        { 5, 5, 0,-3,-3},
        {-3,-3,-3,-3,-3},
        {-3,-3,-3,-3,-3} }
};

for (int y = radius; y < grayscale.Height - radius; y++)
{
    for (int x = radius; x < grayscale.Width - radius; x++)
    {
        // Получаем окрестность
        int[,] pixels = new int[apertureSize, apertureSize];
        for (int ky = -radius; ky <= radius; ky++)
        {
            for (int kx = -radius; kx <= radius; kx++)

```

```

        {
            Color c = grayscale.GetPixel(x + kx, y + ky);
            pixels[ky + radius, kx + radius] = c.R;
        }
    }

    // Применяем все маски Кирша и находим максимум
    int maxMagnitude = 0;
    int[,] masks = (apertureSize == 3) ? kirschMasks3x3 : kirschMasks5x5;

    for (int i = 0; i < 8; i++)
    {
        int sum = 0;
        for (int ky = 0; ky < apertureSize; ky++)
        {
            for (int kx = 0; kx < apertureSize; kx++)
            {
                sum += pixels[ky, kx] * masks[i][ky, kx];
            }
        }
        maxMagnitude = Math.Max(maxMagnitude, Math.Abs(sum));
    }

    // Нормализуем значение
    maxMagnitude = Math.Min(255, Math.Max(0, maxMagnitude));
    result.SetPixel(x, y, Color.FromArgb(maxMagnitude, maxMagnitude, maxMagnitude));
}
}
return result;
}

```

```

private Bitmap ApplyWallisFilter(Bitmap original)
{
    Bitmap result = new Bitmap(original.Width, original.Height);
    Bitmap grayscale = ToGrayscale(original);

    // Получаем параметры из интерфейса
    double mean = (double)numericMean.Value;
    double variance = (double)numericVariance.Value;
    double k = (double)numericGain.Value;
    double a = 1.0 - k;

    // Предварительно вычисляем локальные средние и дисперсии
    int[,] localMeans = new int[original.Width, original.Height];
    int[,] localVars = new int[original.Width, original.Height];
    int radius = 3; // Радиус окрестности для вычисления статистики

    // Вычисляем локальные статистики
    for (int y = radius; y < grayscale.Height - radius; y++)
    {
        for (int x = radius; x < grayscale.Width - radius; x++)
        {
            // Вычисляем среднее в окрестности
            int sum = 0;
            for (int ky = -radius; ky <= radius; ky++)
            {
                for (int kx = -radius; kx <= radius; kx++)
                {
                    sum += grayscale.GetPixel(x + kx, y + ky).R;
                }
            }
            localMeans[x, y] = sum / ((2 * radius + 1) * (2 * radius + 1));
        }
    }
}

```

```

        // Вычисляем дисперсию в окрестности
        int sumSq = 0;
        for (int ky = -radius; ky <= radius; ky++)
        {
            for (int kx = -radius; kx <= radius; kx++)
            {
                int diff = grayscale.GetPixel(x + kx, y + ky).R - localMeans[x, y];
                sumSq += diff * diff;
            }
        }
        localVars[x, y] = sumSq / ((2 * radius + 1) * (2 * radius + 1));
    }
}

// Применяем фильтр Уоллеса
for (int y = radius; y < grayscale.Height - radius; y++)
{
    for (int x = radius; x < grayscale.Width - radius; x++)
    {
        int pixelValue = grayscale.GetPixel(x, y).R;
        double localMean = localMeans[x, y];
        double localVar = Math.Max(1, localVars[x, y]); // Избегаем деления на 0

        // Формула Уоллеса
        double newValue = mean + k * (pixelValue - localMean) *
            (variance / Math.Sqrt(variance * variance + a * a * localVar));

        // Ограничиваем значения
        int finalValue = (int)Math.Max(0, Math.Min(255, newValue));
        result.SetPixel(x, y, Color.FromArgb(finalValue, finalValue, finalValue));
    }
}

return result;
}
//-----

private void файлToolStripMenuItem_Click(object sender, EventArgs e) { }
private void открытьToolStripMenuItem_Click(object sender, EventArgs e) { }
private void label1_Click(object sender, EventArgs e) { }
private void labelStatus_Click(object sender, EventArgs e) { }
private void textBox1_TextChanged(object sender, EventArgs e) { }

private void checkBox2_CheckedChanged(object sender, EventArgs e)
{
}

private void checkBoxShowHistogram_CheckedChanged(object sender, EventArgs e)
{
    if (checkBoxShowHistogram.Checked && pictureBoxOriginal.Image != null)
    {
        try
        {
            Bitmap original = new Bitmap(pictureBoxOriginal.Image);
            Bitmap processed = pictureBoxProcessed.Image != null
                ? new Bitmap(pictureBoxProcessed.Image)
                : null;

            using (var histForm = new HistogramForm(original, processed))
            {
                histForm.ShowDialog();
            }
        }
    }
}

```

```

    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка: {ex.Message}");
    }
    finally
    {
        checkBoxShowHistogram.Checked = false;
    }
}

protected override void OnFormClosing(FormClosingEventArgs e)
{
    Console.WriteLine($"Состояние перед закрытием: ProcessedImage={pictureBoxProcessed.Image !=
null}");
    base.OnFormClosing(e);
}
// ----- КНОПКА ПРИМЕНИТЬ -----
private void btnApply_Click(object sender, EventArgs e)
{
    if (pictureBoxOriginal.Image == null)
    {
        MessageBox.Show("Сначала загрузите изображение!", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    try
    {
        Bitmap original = new Bitmap(pictureBoxOriginal.Image);
        Bitmap result = ApplySelectedFilter(original);

        // Отображаем результат
        pictureBoxProcessed.Image = result;

        // Обновляем статус
        labelStatus.Text = $"Фильтр применён. Размер: {result.Width}x{result.Height}";
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка обработки: {ex.Message}", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void btnShowHistogram_Click(object sender, EventArgs e)
{
    if (pictureBoxOriginal.Image == null)
    {
        MessageBox.Show("Сначала загрузите изображение!", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    try
    {
        Bitmap original = new Bitmap(pictureBoxOriginal.Image);
        Bitmap processed = pictureBoxProcessed.Image != null
            ? new Bitmap(pictureBoxProcessed.Image)
            : null;

        HistogramForm histForm = new HistogramForm(original);
    }
}

```

```

        histForm.Size = new Size(800, 600); // Устанавливаем размер формы
        histForm.Show();
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при создании гистограммы: {ex.Message}", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private Bitmap GenerateTestImage(int width, int height)
{
    Bitmap bmp = new Bitmap(width, height, PixelFormat.Format24bppRgb);
    Rectangle rect = new Rectangle(0, 0, width, height);

    BitmapData bmpData = bmp.LockBits(rect, ImageLockMode.WriteOnly, bmp.PixelFormat);

    try
    {
        int bytesPerPixel = 3; // Для Format24bppRgb
        byte[] pixelValues = new byte[bmpData.Stride * height];
        Random rnd = new Random();

        for (int y = 0; y < height; y++)
        {
            int currentLine = y * bmpData.Stride;

            for (int x = 0; x < width; x++)
            {
                int currentPixel = currentLine + x * bytesPerPixel;
                byte grayValue;

                // Яркие и контрастные тестовые зоны
                if (y < height / 4) // Градиент (0-255)
                {
                    grayValue = (byte)(255 * x / width);
                }
                else if (y < height / 2) // Белая линия (255) на чёрном фоне (0)
                {
                    grayValue = (byte)(Math.Abs(x - width / 2) <= 1 ? 255 : 0);
                }
                else if (y < 3 * height / 4) // Шум (случайные белые точки)
                {
                    grayValue = (byte)(rnd.Next(100) < 5 ? 255 : 0);
                }
                else // Резкий перепад (100/200)
                {
                    grayValue = (byte)(x < width / 2 ? 100 : 200);
                }

                // Записываем BGR (Format24bppRgb)
                pixelValues[currentPixel] = grayValue; // B
                pixelValues[currentPixel + 1] = grayValue; // G
                pixelValues[currentPixel + 2] = grayValue; // R
            }
        }

        Marshal.Copy(pixelValues, 0, bmpData.Scan0, pixelValues.Length);
    }
    finally
    {
        bmp.UnlockBits(bmpData);
    }
}

```



```

        return bmp;
    }

    // Обработчик кнопки "Тестовые образцы"
    private void btnTestPatterns_Click(object sender, EventArgs e)
    {
        try
        {
            Bitmap testImage = GenerateTestImage(400, 400);
            pictureBoxOriginal.Image = testImage;

            if (radioSobel.Checked || radioPrewitt.Checked ||
                radioKirsch.Checked || radioWallis.Checked)
            {
                Bitmap processed = ApplySelectedFilter(testImage);
                pictureBoxProcessed.Image = processed;

                // Анализ разных областей
                var regions = new Dictionary<string, Rectangle>
                {
                    { "Градиент", new Rectangle(0, 0, 400, 100) },
                    { "Резкая граница", new Rectangle(0, 100, 400, 100) },
                    { "Шум", new Rectangle(0, 200, 400, 100) },
                    { "Перепад", new Rectangle(0, 300, 400, 100) }
                };

                StringBuilder report = new StringBuilder();
                report.AppendLine("Результаты анализа:");

                foreach (var region in regions)
                {
                    var analysis = ImageAnalyzer.AnalyzeFragment(
                        testImage, processed, region.Value);

                    report.AppendLine($"n--- {region.Key} ---");
                    foreach (var metric in analysis)
                    {
                        report.AppendLine($"{metric.Key}: {metric.Value:F2}");
                    }
                }

                // Вывод результатов
                string resultFile = "analysis_results.txt";
                File.WriteAllText(resultFile, report.ToString());
                labelStatus.Text = $"Анализ завершен. Результаты в {resultFile}";

                // Автоматическое открытие файла с результатами
                System.Diagnostics.Process.Start("notepad.exe", resultFile);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка: {ex.Message}", "Ошибка",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private Bitmap ApplySelectedFilter(Bitmap original)
    {
        // Получаем выбранный размер апертуры
        int apertureSize = comboApertureSize.SelectedIndex == 0 ? 3 : 5;
    }

```

```

        if (radioSobel.Checked) return ApplySobelFilter(original, apertureSize);
        if (radioPrewitt.Checked) return ApplyPrewittFilter(original, apertureSize);
        if (radioKirsch.Checked) return ApplyKirschFilter(original, apertureSize);
        if (radioWallis.Checked) return ApplyWallisFilter(original);
        return original;
    }

    private void panel1_Paint(object sender, PaintEventArgs e)
    {

    }

    private void UpdateFilterSettingsVisibility()
    {
        bool isWallis = radioWallis.Checked;

        // Настройки Уоллеса
        numericMean.Visible = isWallis;
        numericVariance.Visible = isWallis;
        numericGain.Visible = isWallis;
        labelMean.Visible = isWallis;
        labelVariance.Visible = isWallis;
        labelGain.Visible = isWallis;

        // Настройки апертуры (для всех кроме Уоллеса)
        comboApertureSize.Visible = !isWallis;
        labelAperture.Visible = !isWallis;
    }

    private void radioSobel_CheckedChanged(object sender, EventArgs e) => UpdateFilterSettingsVisibility();
    private void RadioButton_CheckedChanged(object sender, EventArgs e)
    {
        // Проверяем, что переключение действительно произошло
        RadioButton rb = sender as RadioButton;
        if (rb != null && rb.Checked)
        {
            UpdateFilterSettingsVisibility();
        }
    }

    private void btnSaveResult_Click(object sender, EventArgs e)
    {
        if (pictureBoxProcessed.Image == null)
        {
            MessageBox.Show("Нет обработанного изображения для сохранения!", "Ошибка",
                MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        SaveFileDialog saveDialog = new SaveFileDialog();
        saveDialog.Filter = "JPEG Image|*.jpg|PNG Image|*.png|BMP Image|*.bmp";
        saveDialog.Title = "Сохранить обработанное изображение";
        saveDialog.FileName = "processed_image.jpg";

        if (saveDialog.ShowDialog() == DialogResult.OK)
        {
            try
            {
                // Определяем формат на основе расширения файла
                ImageFormat format = ImageFormat.Jpeg;
                switch (Path.GetExtension(saveDialog.FileName).ToLower())
                {
                    case ".png": format = ImageFormat.Png; break;
                }
            }
        }
    }

```

```

        case ".bmp": format = ImageFormat.Bmp; break;
    }

    // Сохраняем изображение
    pictureBoxProcessed.Image.Save(saveDialog.FileName, format);

    labelStatus.Text = $"Изображение сохранено: {saveDialog.FileName}";
}
catch (Exception ex)
{
    MessageBox.Show($"Ошибка при сохранении: {ex.Message}", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

// Добавляем обработчики событий для pictureBox'ов
private void pictureBoxOriginal_MouseMove(object sender, MouseEventArgs e)
{
    if (pictureBoxOriginal.Image == null) return;

    Bitmap bmp = (Bitmap)pictureBoxOriginal.Image;
    if (e.X >= 0 && e.X < bmp.Width && e.Y >= 0 && e.Y < bmp.Height)
    {
        Color pixel = bmp.GetPixel(e.X, e.Y);
        int brightness = (int)(pixel.R * 0.299 + pixel.G * 0.587 + pixel.B * 0.114);

        // Если есть обработанное изображение, показываем и его значения
        if (pictureBoxProcessed.Image != null)
        {
            Bitmap processedBmp = (Bitmap)pictureBoxProcessed.Image;
            Color processedPixel = processedBmp.GetPixel(e.X, e.Y);
            int processedBrightness = (int)(processedPixel.R * 0.299 + processedPixel.G * 0.587 + processedPixel.B
* 0.114);

            labelStatus.Text = $"Исходное: X={e.X}, Y={e.Y}, R={pixel.R}, G={pixel.G}, B={pixel.B},
Яркость={brightness} | " +
                $"Обработанное: R={processedPixel.R}, G={processedPixel.G}, B={processedPixel.B},
Яркость={processedBrightness}";
        }
        else
        {
            labelStatus.Text = $"X={e.X}, Y={e.Y}, R={pixel.R}, G={pixel.G}, B={pixel.B},
Яркость={brightness}";
        }
    }
}

private void PictureBox_MouseMove(object sender, MouseEventArgs e)
{
    PictureBox pb = (PictureBox)sender;
    if (pb.Image == null) return;

    // Масштабирование координат, если изображение увеличено/уменьшено
    int x = (int)(e.X * (float)pb.Image.Width / pb.Width);
    int y = (int)(e.Y * (float)pb.Image.Height / pb.Height);

    Bitmap bmp = (Bitmap)pb.Image;
    Color pixel = bmp.GetPixel(x, y);
    int brightness = (int)(pixel.R * 0.299 + pixel.G * 0.587 + pixel.B * 0.114);

    // Сравнение с оригиналом (если это обработанное изображение)
    if (pb == pictureBoxProcessed && pictureBoxOriginal.Image != null)

```

```

    {
        Bitmap originalBmp = (Bitmap)pictureBoxOriginal.Image;
        Color originalPixel = originalBmp.GetPixel(x, y);
        int originalBrightness = (int)(originalPixel.R * 0.299 + originalPixel.G * 0.587 + originalPixel.B * 0.114);

        labelStatus.Text = $"Координаты: ({x}, {y}) | " +
            $"Оригинал: {originalBrightness} → Обработанное: {brightness}";
    }
    else
    {
        labelStatus.Text = $"Координаты: ({x}, {y}) | Яркость: {brightness}";
    }
}
//-----
}
}

```

Листинг вспомогательного файла Form1.Designer.cs

```

namespace lab1
{
    partial class Form1
    {
        /// <summary>
        /// Обязательная переменная конструктора.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Освободить все используемые ресурсы.
        /// </summary>
        /// <param name="disposing">истинно, если управляемый ресурс должен быть удален; иначе
        ложно.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Код, автоматически созданный конструктором форм Windows

        /// <summary>
        /// Требуемый метод для поддержки конструктора — не изменяйте
        /// содержимое этого метода с помощью редактора кода.
        /// </summary>
        private void InitializeComponent()
        {
            this.pictureBoxOriginal = new System.Windows.Forms.PictureBox();
            this.pictureBoxProcessed = new System.Windows.Forms.PictureBox();
            this.labelOriginal = new System.Windows.Forms.Label();
            this.labelProcessed = new System.Windows.Forms.Label();
            this.btnLoadImage = new System.Windows.Forms.Button();
            this.labelStatus = new System.Windows.Forms.Label();
            this.radioSobel = new System.Windows.Forms.RadioButton();
            this.radioPrewitt = new System.Windows.Forms.RadioButton();
            this.radioKirsch = new System.Windows.Forms.RadioButton();
            this.radioWallis = new System.Windows.Forms.RadioButton();
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.groupBoxOptions = new System.Windows.Forms.GroupBox();
            this.checkBoxShowHistogram = new System.Windows.Forms.CheckBox();

```

```

this.btnApply = new System.Windows.Forms.Button();
this.btnTestPatterns = new System.Windows.Forms.Button();
this.panel1 = new System.Windows.Forms.Panel();
this.numericMean = new System.Windows.Forms.NumericUpDown();
this.numericVariance = new System.Windows.Forms.NumericUpDown();
this.numericGain = new System.Windows.Forms.NumericUpDown();
this.labelMean = new System.Windows.Forms.Label();
this.labelVariance = new System.Windows.Forms.Label();
this.labelGain = new System.Windows.Forms.Label();
this.comboApertureSize = new System.Windows.Forms.ComboBox();
this.labelAperture = new System.Windows.Forms.Label();
this.btnSaveResult = new System.Windows.Forms.Button();
((System.ComponentModel.ISupportInitialize)(this.pictureBoxOriginal)).BeginInit();
((System.ComponentModel.ISupportInitialize)(this.pictureBoxProcessed)).BeginInit();
this.groupBox1.SuspendLayout();
this.groupBoxOptions.SuspendLayout();
((System.ComponentModel.ISupportInitialize)(this.numericMean)).BeginInit();
((System.ComponentModel.ISupportInitialize)(this.numericVariance)).BeginInit();
((System.ComponentModel.ISupportInitialize)(this.numericGain)).BeginInit();
this.SuspendLayout();
//
// pictureBoxOriginal
//
this.pictureBoxOriginal.Anchor
((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.Top
System.Windows.Forms.AnchorStyles.Bottom)
| System.Windows.Forms.AnchorStyles.Left)
| System.Windows.Forms.AnchorStyles.Right)));
this.pictureBoxOriginal.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
this.pictureBoxOriginal.Location = new System.Drawing.Point(76, 54);
this.pictureBoxOriginal.Name = "pictureBoxOriginal";
this.pictureBoxOriginal.Size = new System.Drawing.Size(451, 371);
this.pictureBoxOriginal.TabIndex = 1;
this.pictureBoxOriginal.TabStop = false;
//
// pictureBoxProcessed
//
this.pictureBoxProcessed.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
this.pictureBoxProcessed.Location = new System.Drawing.Point(728, 54);
this.pictureBoxProcessed.Name = "pictureBoxProcessed";
this.pictureBoxProcessed.Size = new System.Drawing.Size(432, 371);
this.pictureBoxProcessed.TabIndex = 1;
this.pictureBoxProcessed.TabStop = false;
//
// labelOriginal
//
this.labelOriginal.AutoSize = true;
this.labelOriginal.Font = new System.Drawing.Font("Microsoft Sans Serif", 18F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.labelOriginal.Location = new System.Drawing.Point(102, 15);
this.labelOriginal.Name = "labelOriginal";
this.labelOriginal.Size = new System.Drawing.Size(379, 36);
this.labelOriginal.TabIndex = 2;
this.labelOriginal.Text = "Исходное изображение";
this.labelOriginal.Click += new System.EventHandler(this.label1_Click);
//
// labelProcessed
//
this.labelProcessed.AutoSize = true;
this.labelProcessed.Font = new System.Drawing.Font("Microsoft Sans Serif", 18F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.labelProcessed.Location = new System.Drawing.Point(722, 15);
this.labelProcessed.Name = "labelProcessed";

```

```

this.labelProcessed.Size = new System.Drawing.Size(450, 36);
this.labelProcessed.TabIndex = 2;
this.labelProcessed.Text = "Обработанное изображение";
this.labelProcessed.Click += new System.EventHandler(this.label1_Click);
//
// btnLoadImage
//
this.btnLoadImage.Font = new System.Drawing.Font("Microsoft Sans Serif", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.btnLoadImage.Location = new System.Drawing.Point(76, 457);
this.btnLoadImage.Name = "btnLoadImage";
this.btnLoadImage.Size = new System.Drawing.Size(349, 56);
this.btnLoadImage.TabIndex = 3;
this.btnLoadImage.Text = "Загрузить изображение";
this.btnLoadImage.UseVisualStyleBackColor = true;
this.btnLoadImage.Click += new System.EventHandler(this.btnLoadImage_Click);
//
// labelStatus
//
this.labelStatus.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.labelStatus.Location = new System.Drawing.Point(431, 432);
this.labelStatus.Name = "labelStatus";
this.labelStatus.Size = new System.Drawing.Size(447, 133);
this.labelStatus.TabIndex = 4;
this.labelStatus.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
this.labelStatus.Click += new System.EventHandler(this.labelStatus_Click);
//
// radioSobel
//
this.radioSobel.AutoSize = true;
this.radioSobel.Checked = true;
this.radioSobel.Font = new System.Drawing.Font("Microsoft Sans Serif", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.radioSobel.Location = new System.Drawing.Point(32, 44);
this.radioSobel.Name = "radioSobel";
this.radioSobel.Size = new System.Drawing.Size(122, 33);
this.radioSobel.TabIndex = 5;
this.radioSobel.TabStop = true;
this.radioSobel.Text = "Собеля";
this.radioSobel.UseVisualStyleBackColor = true;
this.radioSobel.CheckedChanged += new System.EventHandler(this.radioSobel_CheckedChanged);
//
// radioPrewitt
//
this.radioPrewitt.AutoSize = true;
this.radioPrewitt.Font = new System.Drawing.Font("Microsoft Sans Serif", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.radioPrewitt.Location = new System.Drawing.Point(295, 44);
this.radioPrewitt.Name = "radioPrewitt";
this.radioPrewitt.Size = new System.Drawing.Size(143, 33);
this.radioPrewitt.TabIndex = 5;
this.radioPrewitt.Text = "Превитта";
this.radioPrewitt.UseVisualStyleBackColor = true;
//
// radioKirsch
//
this.radioKirsch.AutoSize = true;
this.radioKirsch.Font = new System.Drawing.Font("Microsoft Sans Serif", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.radioKirsch.Location = new System.Drawing.Point(575, 44);
this.radioKirsch.Name = "radioKirsch";
this.radioKirsch.Size = new System.Drawing.Size(113, 33);

```

```

this.radioKirsch.TabIndex = 5;
this.radioKirsch.Text = "Кирша";
this.radioKirsch.UseVisualStyleBackColor = true;
//
// radioWallis
//
this.radioWallis.AutoSize = true;
this.radioWallis.Font = new System.Drawing.Font("Microsoft Sans Serif", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.radioWallis.Location = new System.Drawing.Point(858, 44);
this.radioWallis.Name = "radioWallis";
this.radioWallis.Size = new System.Drawing.Size(133, 33);
this.radioWallis.TabIndex = 5;
this.radioWallis.Text = "Уоллеса";
this.radioWallis.UseVisualStyleBackColor = true;
//
// groupBox1
//
this.groupBox1.Controls.Add(this.radioSobel);
this.groupBox1.Controls.Add(this.radioPrewitt);
this.groupBox1.Controls.Add(this.radioWallis);
this.groupBox1.Controls.Add(this.radioKirsch);
this.groupBox1.Font = new System.Drawing.Font("Microsoft Sans Serif", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.groupBox1.Location = new System.Drawing.Point(64, 550);
this.groupBox1.Name = "groupBox1";
this.groupBox1.Size = new System.Drawing.Size(1096, 84);
this.groupBox1.TabIndex = 7;
this.groupBox1.TabStop = false;
this.groupBox1.Text = "Выберите фильтр";
//
// groupBoxOptions
//
this.groupBoxOptions.Controls.Add(this.checkBoxShowHistogram);
this.groupBoxOptions.Font = new System.Drawing.Font("Microsoft Sans Serif", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.groupBoxOptions.Location = new System.Drawing.Point(64, 650);
this.groupBoxOptions.Name = "groupBoxOptions";
this.groupBoxOptions.Size = new System.Drawing.Size(1096, 82);
this.groupBoxOptions.TabIndex = 8;
this.groupBoxOptions.TabStop = false;
this.groupBoxOptions.Text = "Дополнительно";
//
// checkBoxShowHistogram
//
this.checkBoxShowHistogram.AutoSize = true;
this.checkBoxShowHistogram.Location = new System.Drawing.Point(387, 35);
this.checkBoxShowHistogram.Name = "checkBoxShowHistogram";
this.checkBoxShowHistogram.Size = new System.Drawing.Size(301, 33);
this.checkBoxShowHistogram.TabIndex = 0;
this.checkBoxShowHistogram.Text = "Показать гистограмму";
this.checkBoxShowHistogram.UseVisualStyleBackColor = true;
this.checkBoxShowHistogram.CheckedChanged += new
System.EventHandler(this.checkBoxShowHistogram_CheckedChanged);
//
// btnApply
//
this.btnApply.Font = new System.Drawing.Font("Microsoft Sans Serif", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.btnApply.Location = new System.Drawing.Point(908, 470);
this.btnApply.Name = "btnApply";
this.btnApply.Size = new System.Drawing.Size(215, 49);
this.btnApply.TabIndex = 9;

```

```

this.btnApply.Text = "Применить";
this.btnApply.UseVisualStyleBackColor = true;
this.btnApply.Click += new System.EventHandler(this.btnApply_Click);
//
// btnTestPatterns
//
this.btnTestPatterns.Font = new System.Drawing.Font("Microsoft Sans Serif", 15F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.btnTestPatterns.Location = new System.Drawing.Point(64, 800);
this.btnTestPatterns.Name = "btnTestPatterns";
this.btnTestPatterns.Size = new System.Drawing.Size(329, 37);
this.btnTestPatterns.TabIndex = 10;
this.btnTestPatterns.Text = "Тестовые образцы";
this.btnTestPatterns.UseVisualStyleBackColor = true;
this.btnTestPatterns.Click += new System.EventHandler(this.btnTestPatterns_Click);
//
// panel1
//
this.panel1.Location = new System.Drawing.Point(76, 54);
this.panel1.Name = "panel1";
this.panel1.Size = new System.Drawing.Size(451, 375);
this.panel1.TabIndex = 11;
this.panel1.Paint += new System.Windows.Forms.PaintEventHandler(this.panel1_Paint);
//
// numericMean
//
this.numericMean.Location = new System.Drawing.Point(147, 763);
this.numericMean.Maximum = new decimal(new int[] {
255,
0,
0,
0});
this.numericMean.Name = "numericMean";
this.numericMean.Size = new System.Drawing.Size(120, 22);
this.numericMean.TabIndex = 12;
this.numericMean.Value = new decimal(new int[] {
128,
0,
0,
0});
//
// numericVariance
//
this.numericVariance.Location = new System.Drawing.Point(451, 764);
this.numericVariance.Maximum = new decimal(new int[] {
1000,
0,
0,
0});
this.numericVariance.Name = "numericVariance";
this.numericVariance.Size = new System.Drawing.Size(120, 22);
this.numericVariance.TabIndex = 12;
this.numericVariance.Value = new decimal(new int[] {
50,
0,
0,
0});
//
// numericGain
//
this.numericGain.Location = new System.Drawing.Point(758, 765);
this.numericGain.Maximum = new decimal(new int[] {
1,

```



```

0,
0,
0});
this.numericGain.Minimum = new decimal(new int[] {
1,
0,
0,
65536});
this.numericGain.Name = "numericGain";
this.numericGain.Size = new System.Drawing.Size(120, 22);
this.numericGain.TabIndex = 12;
this.numericGain.Value = new decimal(new int[] {
8,
0,
0,
65536});
//
// labelMean
//
this.labelMean.AutoSize = true;
this.labelMean.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.labelMean.Location = new System.Drawing.Point(73, 763);
this.labelMean.Name = "labelMean";
this.labelMean.Size = new System.Drawing.Size(55, 20);
this.labelMean.TabIndex = 13;
this.labelMean.Text = "Mean:";
//
// labelVariance
//
this.labelVariance.AutoSize = true;
this.labelVariance.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.labelVariance.Location = new System.Drawing.Point(345, 764);
this.labelVariance.Name = "labelVariance";
this.labelVariance.Size = new System.Drawing.Size(80, 20);
this.labelVariance.TabIndex = 13;
this.labelVariance.Text = "Variance:";
//
// labelGain
//
this.labelGain.AutoSize = true;
this.labelGain.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.labelGain.Location = new System.Drawing.Point(649, 764);
this.labelGain.Name = "labelGain";
this.labelGain.Size = new System.Drawing.Size(49, 20);
this.labelGain.TabIndex = 13;
this.labelGain.Text = "Gain:";
//
// comboApertureSize
//
this.comboApertureSize.FormattingEnabled = true;
this.comboApertureSize.Items.AddRange(new object[] {
"3x3",
"5x5"});
this.comboApertureSize.Location = new System.Drawing.Point(1039, 765);
this.comboApertureSize.Name = "comboApertureSize";
this.comboApertureSize.Size = new System.Drawing.Size(121, 24);
this.comboApertureSize.TabIndex = 14;
//
// labelAperture
//

```

```

        this.labelAperture.AutoSize = true;
        this.labelAperture.Font = new System.Drawing.Font("Microsoft Sans Serif", 10F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
        this.labelAperture.Location = new System.Drawing.Point(936, 765);
        this.labelAperture.Name = "labelAperture";
        this.labelAperture.Size = new System.Drawing.Size(78, 20);
        this.labelAperture.TabIndex = 15;
        this.labelAperture.Text = "Aperture:";
        //
        // btnSaveResult
        //
        this.btnSaveResult.Font = new System.Drawing.Font("Microsoft Sans Serif", 13F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(204)));
        this.btnSaveResult.Location = new System.Drawing.Point(533, 195);
        this.btnSaveResult.Name = "btnSaveResult";
        this.btnSaveResult.Size = new System.Drawing.Size(189, 64);
        this.btnSaveResult.TabIndex = 16;
        this.btnSaveResult.Text = "Сохранить результат";
        this.btnSaveResult.UseVisualStyleBackColor = true;
        this.btnSaveResult.Click += new System.EventHandler(this.btnSaveResult_Click);
        //
        // Form1
        //
        this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(1273, 1055);
        this.Controls.Add(this.btnSaveResult);
        this.Controls.Add(this.labelAperture);
        this.Controls.Add(this.comboApertureSize);
        this.Controls.Add(this.labelGain);
        this.Controls.Add(this.labelVariance);
        this.Controls.Add(this.labelMean);
        this.Controls.Add(this.numericGain);
        this.Controls.Add(this.numericVariance);
        this.Controls.Add(this.numericMean);
        this.Controls.Add(this.btnTestPatterns);
        this.Controls.Add(this.btnApply);
        this.Controls.Add(this.groupBoxOptions);
        this.Controls.Add(this.groupBox1);
        this.Controls.Add(this.labelStatus);
        this.Controls.Add(this.btnLoadImage);
        this.Controls.Add(this.labelProcessed);
        this.Controls.Add(this.labelOriginal);
        this.Controls.Add(this.pictureBoxProcessed);
        this.Controls.Add(this.pictureBoxOriginal);
        this.Controls.Add(this.panel1);
        this.Name = "Form1";
        this.Text = "Выделение контуров изображения";
        this.Load += new System.EventHandler(this.Form1_Load);
        ((System.ComponentModel.ISupportInitialize)(this.pictureBoxOriginal)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.pictureBoxProcessed)).EndInit();
        this.groupBox1.ResumeLayout(false);
        this.groupBox1.PerformLayout();
        this.groupBoxOptions.ResumeLayout(false);
        this.groupBoxOptions.PerformLayout();
        ((System.ComponentModel.ISupportInitialize)(this.numericMean)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericVariance)).EndInit();
        ((System.ComponentModel.ISupportInitialize)(this.numericGain)).EndInit();
        this.ResumeLayout(false);
        this.PerformLayout();

    }

```

```

#endregion
private System.Windows.Forms.PictureBox pictureBoxOriginal;
private System.Windows.Forms.PictureBox pictureBoxProcessed;
private System.Windows.Forms.Label labelOriginal;
private System.Windows.Forms.Label labelProcessed;
private System.Windows.Forms.Button btnLoadImage;
private System.Windows.Forms.Label labelStatus;
private System.Windows.Forms.RadioButton radioSobel;
private System.Windows.Forms.RadioButton radioPrewitt;
private System.Windows.Forms.RadioButton radioKirsch;
private System.Windows.Forms.RadioButton radioWallis;
private System.Windows.Forms.GroupBox groupBox1;
private System.Windows.Forms.GroupBox groupBoxOptions;
private System.Windows.Forms.CheckBox checkBoxShowHistogram;
private System.Windows.Forms.Button btnApply;
private System.Windows.Forms.Button btnTestPatterns;
private System.Windows.Forms.Panel panel1;
private System.Windows.Forms.NumericUpDown numericMean;
private System.Windows.Forms.NumericUpDown numericVariance;
private System.Windows.Forms.NumericUpDown numericGain;
private System.Windows.Forms.Label labelMean;
private System.Windows.Forms.Label labelVariance;
private System.Windows.Forms.Label labelGain;
private System.Windows.Forms.ComboBox comboApertureSize;
private System.Windows.Forms.Label labelAperture;
private System.Windows.Forms.Button btnSaveResult;
    }
}

```

Листинг вспомогательного файла HistogramForm.cs

```

using System;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;

namespace lab1
{
    public partial class HistogramForm : Form
    {
        private readonly Bitmap originalImage;
        private readonly Bitmap processedImage;
        private const int HistWidth = 500;
        private const int HistHeight = 250;
        private const int BorderPadding = 40;
        private const int VerticalSpacing = 60; // Расстояние между гистограммами

        public HistogramForm(Bitmap original, Bitmap processed = null)
        {
            InitializeComponent();

            Console.WriteLine($"ДО присваивания: Original={original != null},
Processed={processed != null}");

            this.originalImage = original;
            this.processedImage = processed != null ? new Bitmap(processed) : null; //
            // Создаем копию если есть

            Console.WriteLine($"ПОСЛЕ присваивания: Original={this.originalImage !=
null}, Processed={this.processedImage != null}");

            // Отладочное сообщение для проверки

```

```

        Console.WriteLine($"Processed image is null: {processed == null}");

        CalculateFormSize();
        this.Text = "Гистограммы яркости";
        this.DoubleBuffered = true;
        this.BackColor = Color.White;
        this.Paint += HistogramForm_Paint;
    }

    private void CalculateFormSize()
    {
        int height = HistHeight + 2 * BorderPadding + 60;
        if (processedImage != null) // Проверяем именно поле класса
        {
            height += HistHeight + VerticalSpacing;
            Console.WriteLine("Adding space for processed image histogram");
        }
        this.ClientSize = new Size(HistWidth + 2 * BorderPadding, height);
    }

    private void HistogramForm_Paint(object sender, PaintEventArgs e)
    {
        Graphics g = e.Graphics;
        g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;

        // Всегда рисуем оригинальную гистограмму
        DrawSingleHistogram(g, originalImage, "Исходное изображение", BorderPadding);

        // Рисуем обработанную только если она есть
        if (processedImage != null) // Проверяем поле класса
        {
            Console.WriteLine("Drawing processed image histogram");
            int yOffset = BorderPadding + HistHeight + VerticalSpacing;
            DrawSingleHistogram(g, processedImage, "Обработанное изображение",
yOffset);
        }
    }

    private void DrawSingleHistogram(Graphics g, Bitmap image, string title, int
yOffset)
    {
        // 1. Вычисляем гистограмму яркости
        int[] histogram = new int[256];
        for (int x = 0; x < image.Width; x++)
        {
            for (int y = 0; y < image.Height; y++)
            {
                Color pixel = image.GetPixel(x, y);
                int brightness = (int)(pixel.R * 0.3 + pixel.G * 0.59 + pixel.B *
0.11);
                histogram[brightness]++;
            }
        }

        // 2. Считаем статистику
        int maxBrightness = 0;
        int maxCount = 0;
        int totalPixels = 0;
        long sumBrightness = 0;

        for (int i = 0; i < 256; i++)
        {
            if (histogram[i] > maxCount)
            {
                maxCount = histogram[i];
            }
        }
    }

```

```

        maxBrightness = i;
    }
    totalPixels += histogram[i];
    sumBrightness += histogram[i] * i;
}
double avgBrightness = (totalPixels > 0) ? (double)sumBrightness /
totalPixels : 0;

// 3. Логарифмическое масштабирование
double[] logHist = new double[256];
int maxNonZero = histogram.Max();
for (int i = 0; i < 256; i++)
{
    logHist[i] = histogram[i] > 0 ? Math.Log10(histogram[i]) : 0;
}
double maxLog = (maxNonZero > 0) ? Math.Log10(maxNonZero) : 1;

// 4. Рисуем оси и подписи
using (Pen axisPen = new Pen(Color.Black, 2))
{
    // Ось X
    g.DrawLine(axisPen, BorderPadding, yOffset + HistHeight,
        BorderPadding + HistWidth, yOffset + HistHeight);
    // Ось Y
    g.DrawLine(axisPen, BorderPadding, yOffset,
        BorderPadding, yOffset + HistHeight);
}

// Подписи осей
using (Font axisFont = new Font("Arial", 10, FontStyle.Bold))
{
    // Ось X
    g.DrawString("Яркость (0-255)", axisFont, Brushes.Black,
        BorderPadding + HistWidth / 2 - 50, yOffset + HistHeight +
20);

    // Ось Y (вертикальный текст)
    var yAxisTitleFormat = new StringFormat
    {
        FormatFlags = StringFormatFlags.DirectionVertical
    };
    g.DrawString("Количество (log10)", axisFont, Brushes.Black,
        BorderPadding - 25, yOffset + HistHeight / 2 - 50,
yAxisTitleFormat);
}

// 5. Рисуем столбцы гистограммы
int barWidth = 2;
int gap = 1;
for (int i = 0; i < 256; i++)
{
    if (histogram[i] == 0) continue;

    double logValue = logHist[i];
    float barHeight = (float)(logValue / maxLog * HistHeight);
    int xPos = BorderPadding + (i * (HistWidth / 256));

    using (var brush = new SolidBrush(title.Contains("Исходное") ?
Color.SteelBlue : Color.Orange))
    {
        g.FillRectangle(brush, xPos + gap, yOffset + HistHeight - barHeight,
            barWidth, barHeight);
    }
}

```

```

        // 6. Выводим статистику
        using (Font statsFont = new Font("Arial", 9, FontStyle.Bold))
        {
            string statsText = $"Макс. яркость: {maxBrightness}\n" +
                               $"Пикселей: {totalPixels}\n" +
                               $"Средняя: {avgBrightness:F1}";

            g.DrawString(statsText, statsFont, Brushes.Black,
                          BorderPadding + HistWidth + 10, yOffset + 20);
        }
    }
}

```

Листинг вспомогательного файла `HistogramForm.Designer.cs`

```

namespace labal
{
    partial class HistogramForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
        otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.SuspendLayout();
            //
            // HistogramForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(776, 599);
            this.Name = "HistogramForm";
            this.Text = "Form2";
            this.ResumeLayout(false);

        }

        #endregion
    }
}

```

Результаты выполнения работы

Работоспособность разработанного приложения продемонстрирована на рисунках 3 – 21.

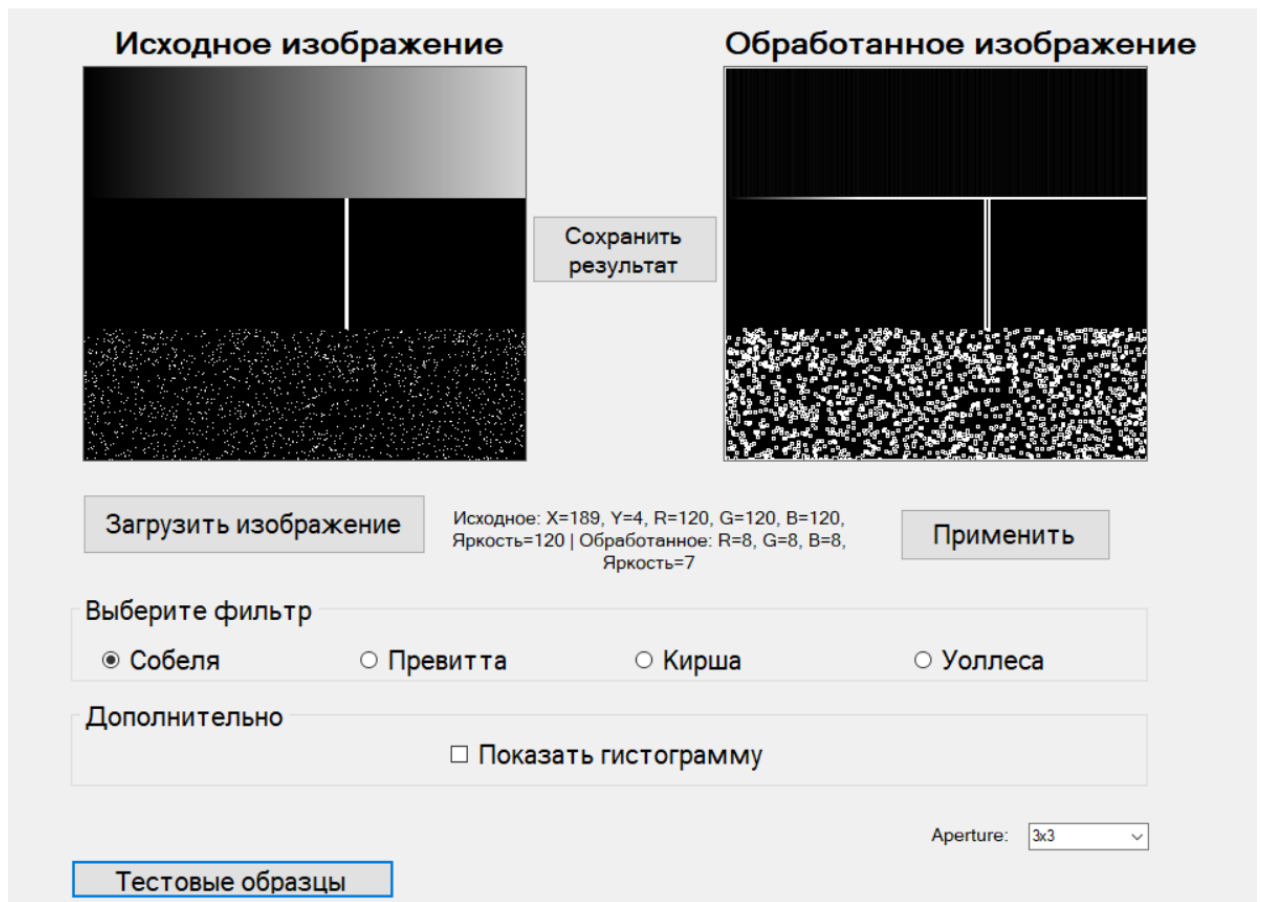


Рисунок 3 – Тестовый образец (фильтр Собеля, апертура 3x3)

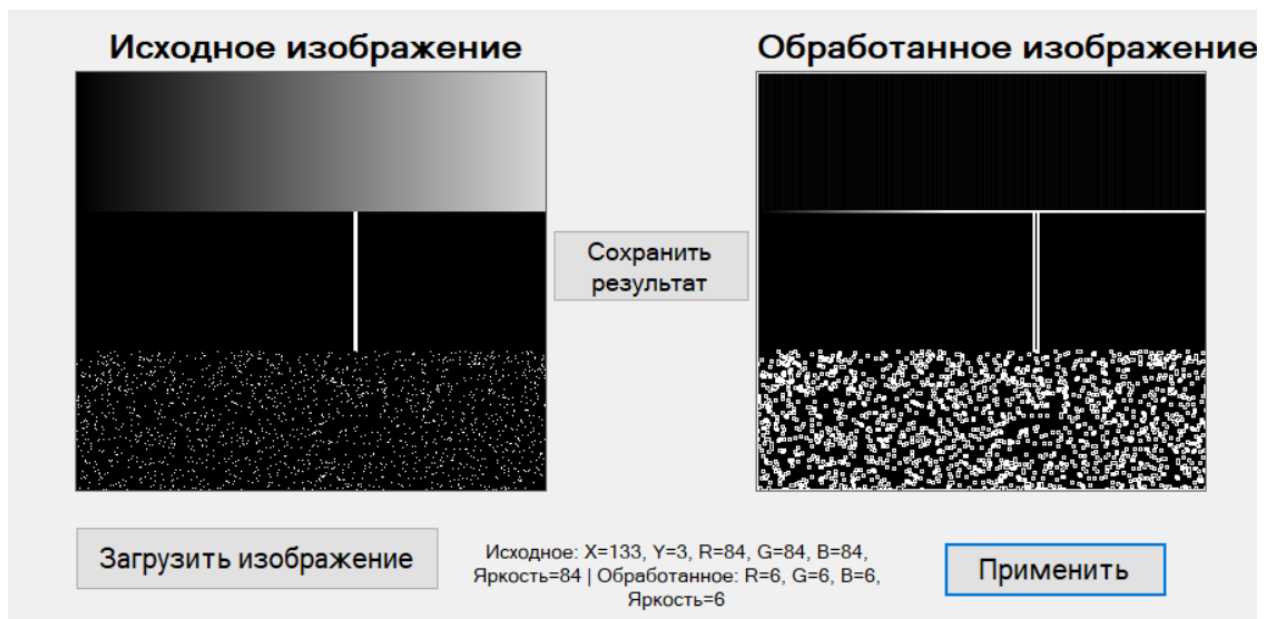


Рисунок 4 – Тестовый образец (фильтр Превитта, апертура 3x3)

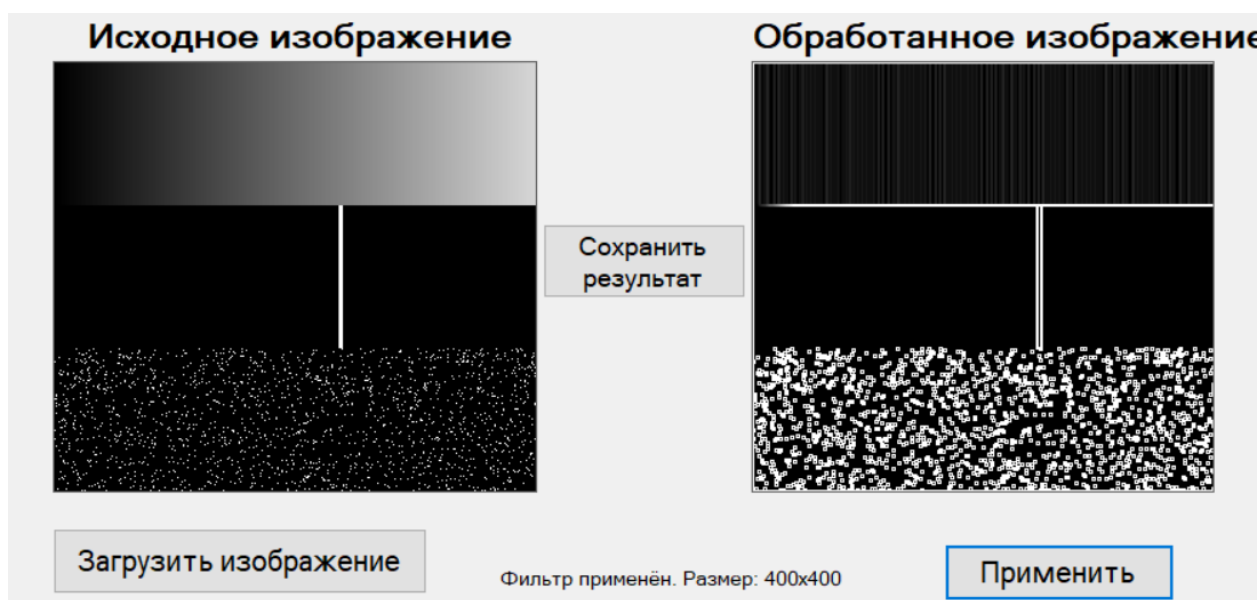


Рисунок 5 – Тестовый образец (фильтр Кирша, апертюра 3x3)

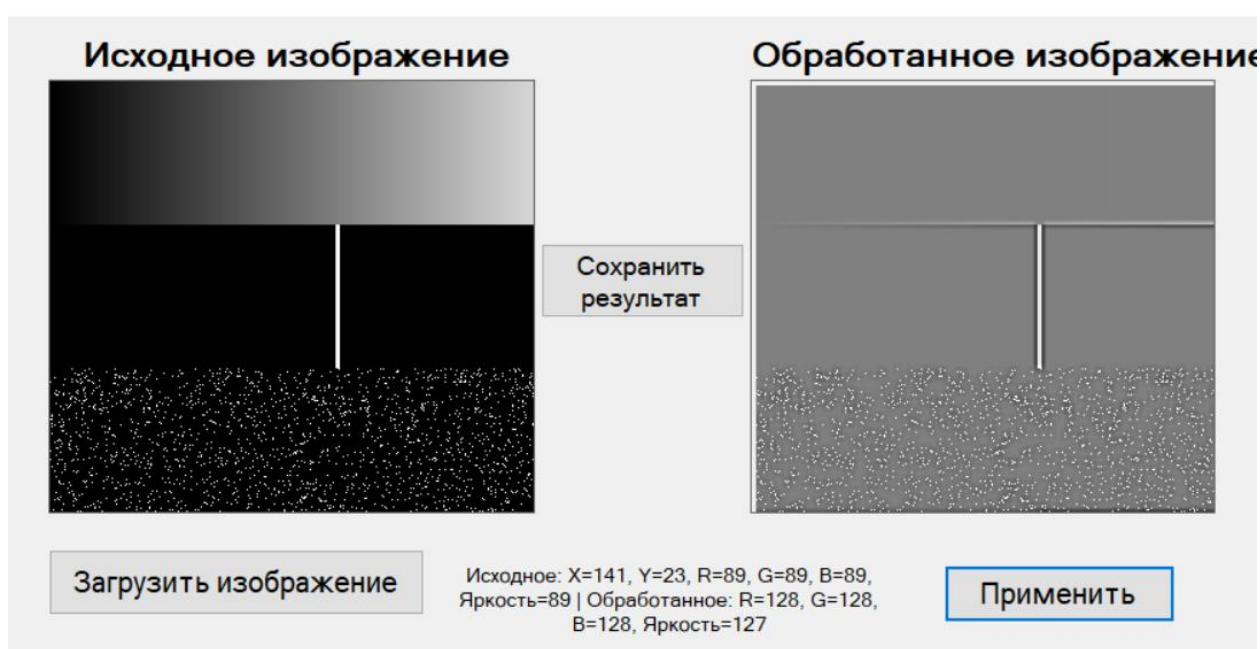


Рисунок 6 – Тестовый образец (фильтр Уоллеса, Mean = 128, Variance = 50, Gain = 1)

Следует отметить, что после применения выбранного фильтра создаётся текстовый файл **analysis_results**, который содержит в себе всю необходимую информацию для анализа. Пример вывода продемонстрирован ниже:

Результаты анализа:

--- Градиент ---

Средняя яркость (оригинал): 126,69

Средняя яркость (результат): 7,17

Среднее изменение: 121,56

Максимальное изменение: 254,00

--- Резкая граница ---

Средняя яркость (оригинал): 1,91

Средняя яркость (результат): 5,14

Среднее изменение: 4,48

Максимальное изменение: 255,00

--- Шум ---

Средняя яркость (оригинал): 13,01

Средняя яркость (результат): 86,05

Среднее изменение: 90,04

Максимальное изменение: 255,00

--- Перепад ---

Средняя яркость (оригинал): 150,00

Средняя яркость (результат): 3,75

Среднее изменение: 149,29

Максимальное изменение: 200,00

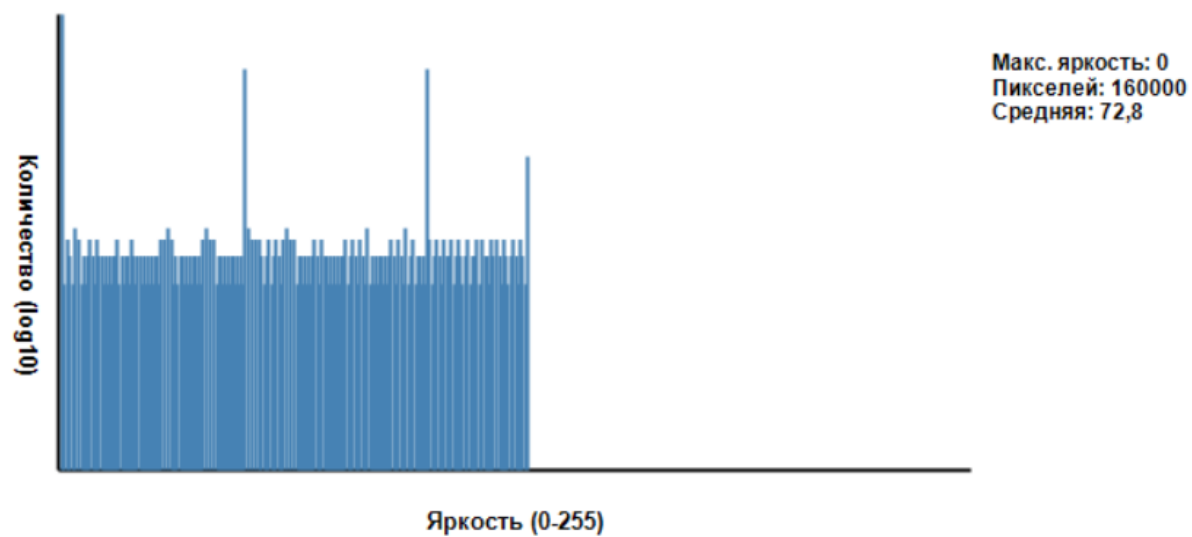


Рисунок 7 – Гистограммы (фильтр Собеля)

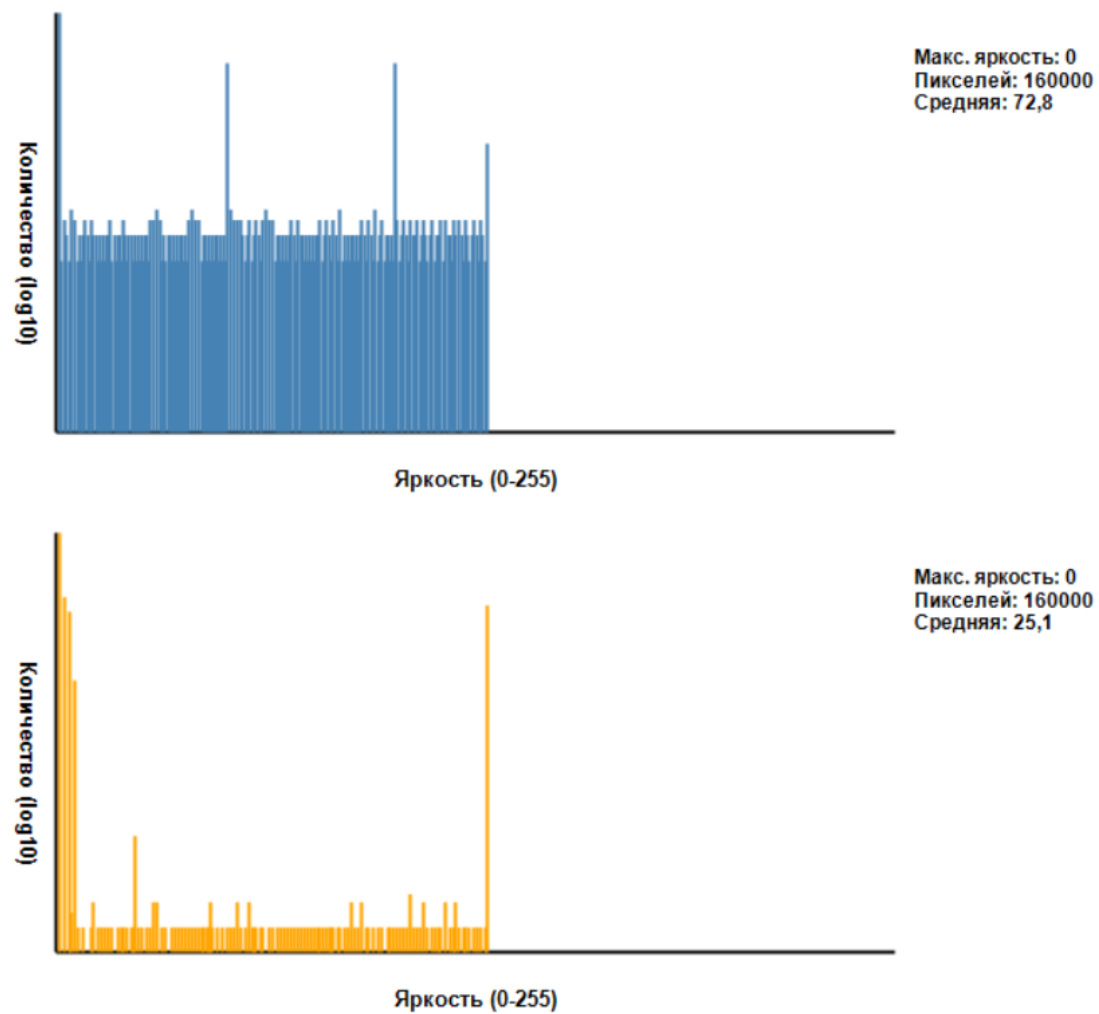


Рисунок 8 – Гистограммы (фильтр Превитта)

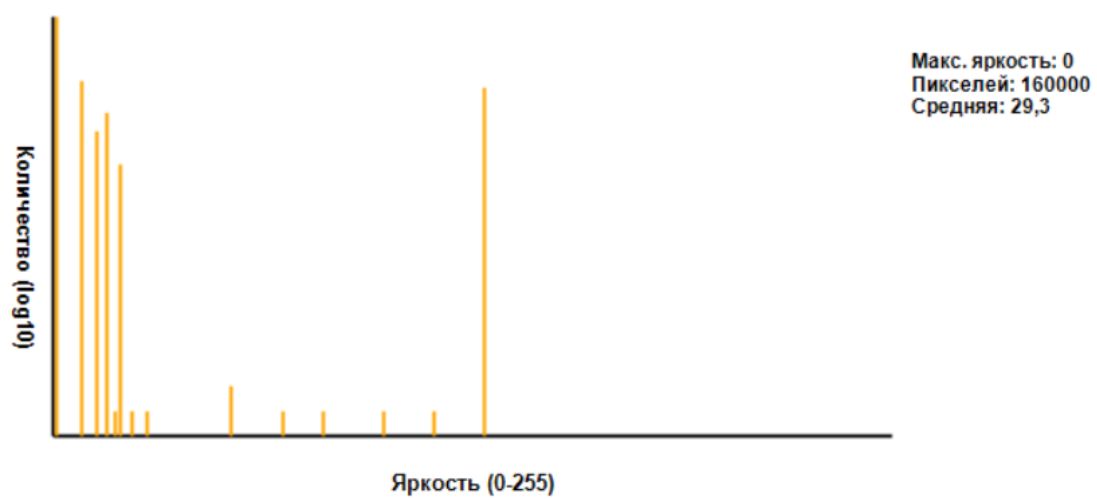
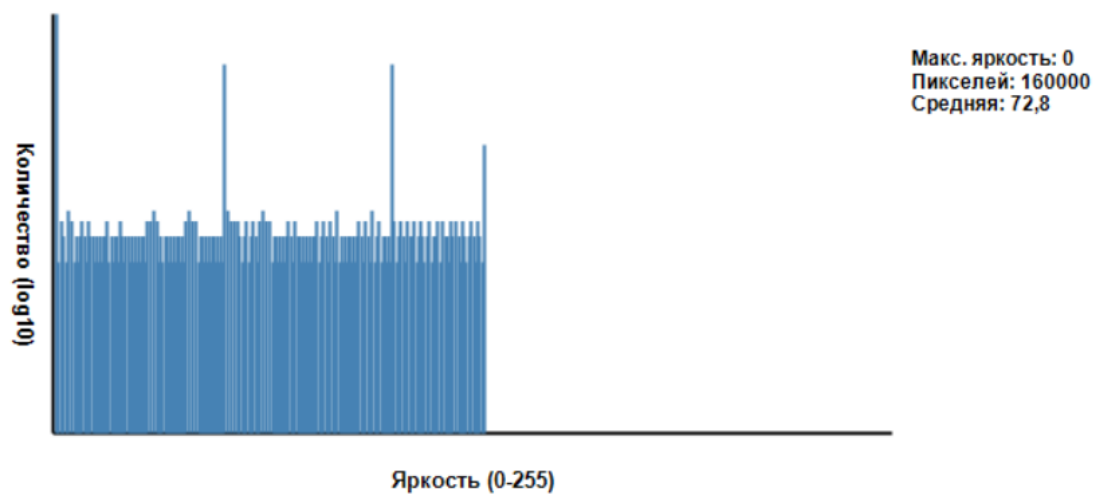


Рисунок 9 – Гистограммы (фильтр Кирша)

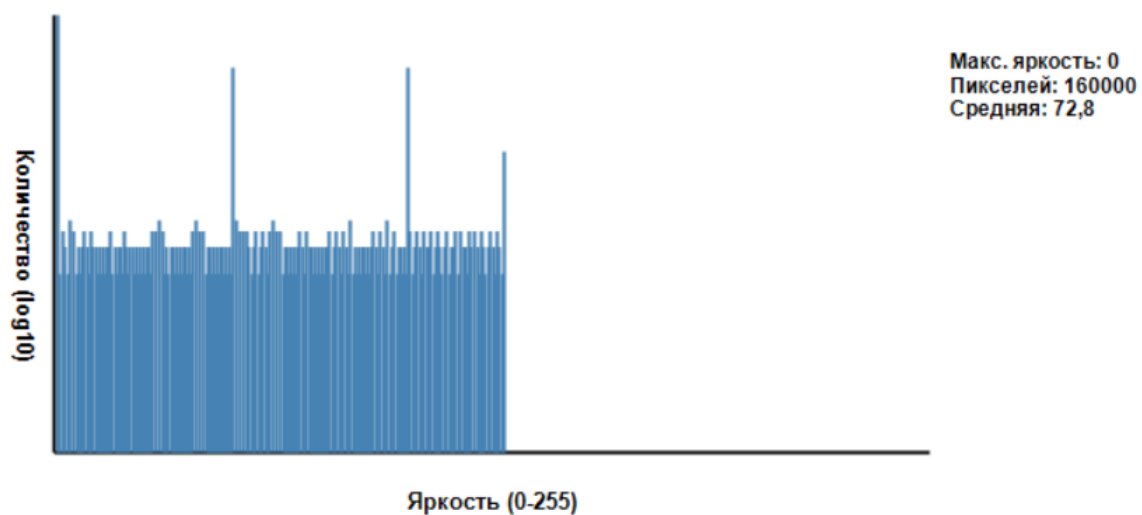


Рисунок 10 – Гистограммы (фильтр Уоллеса)

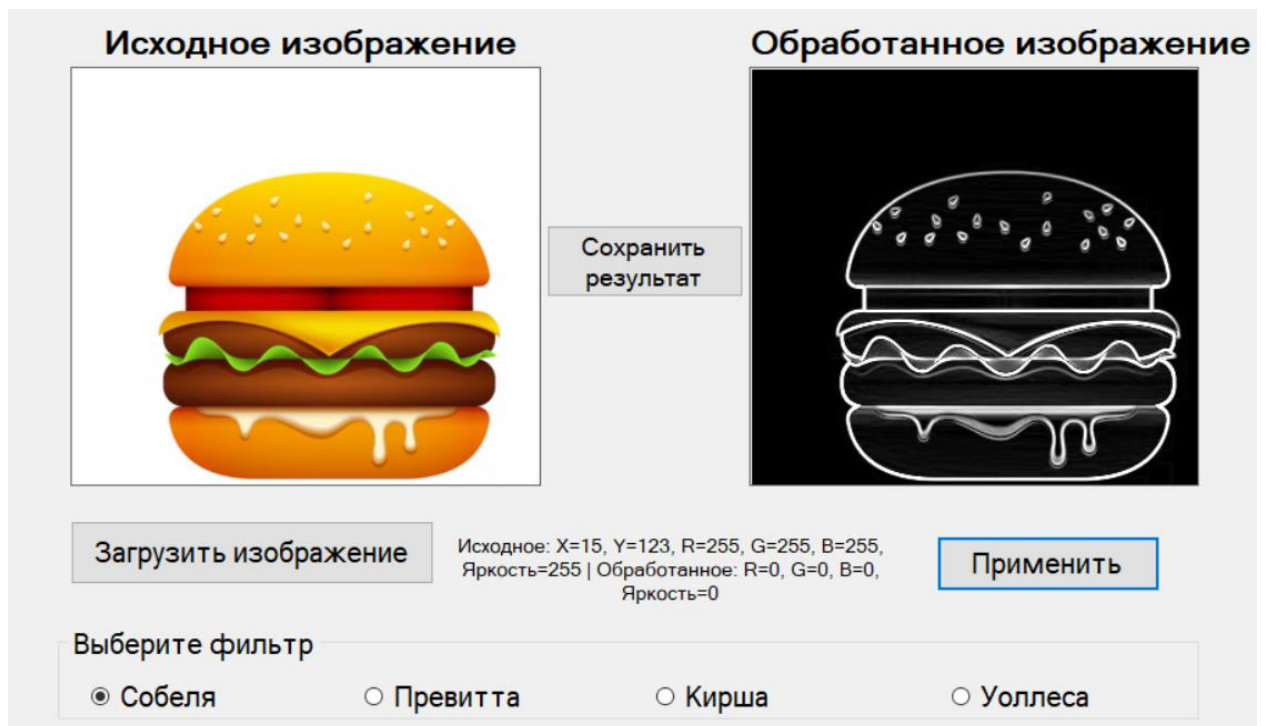


Рисунок 11 – Загруженное изображение (фильтр Собеля, апертура 3x3)

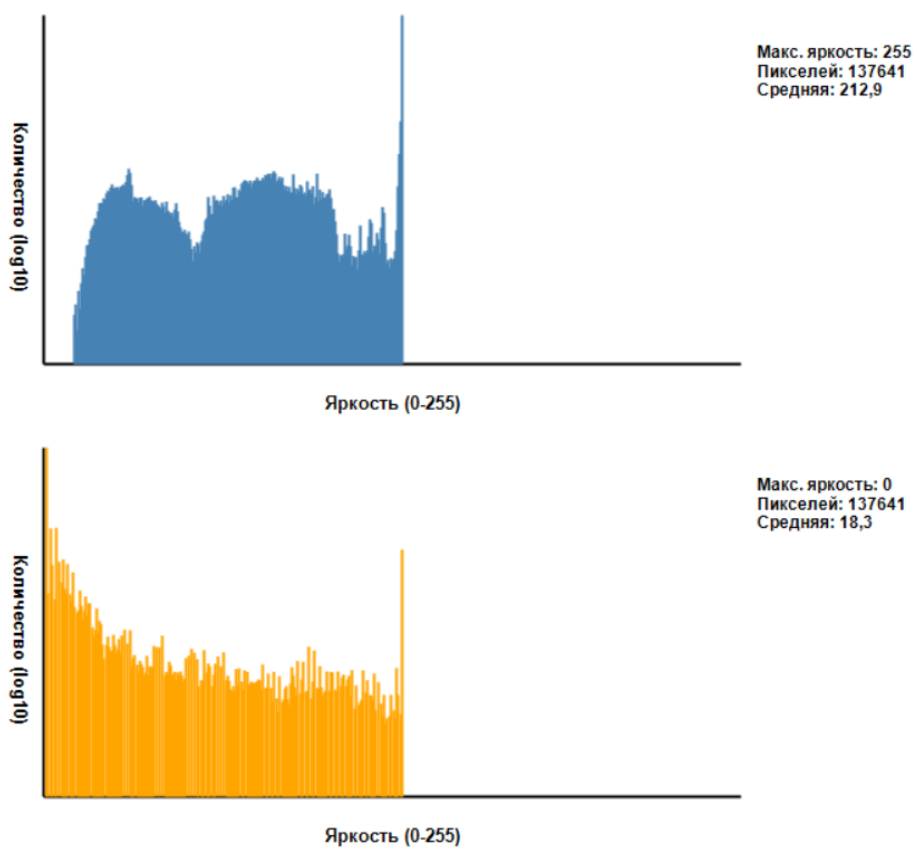


Рисунок 12 – Загруженное изображение (гистограммы, фильтр Собеля)

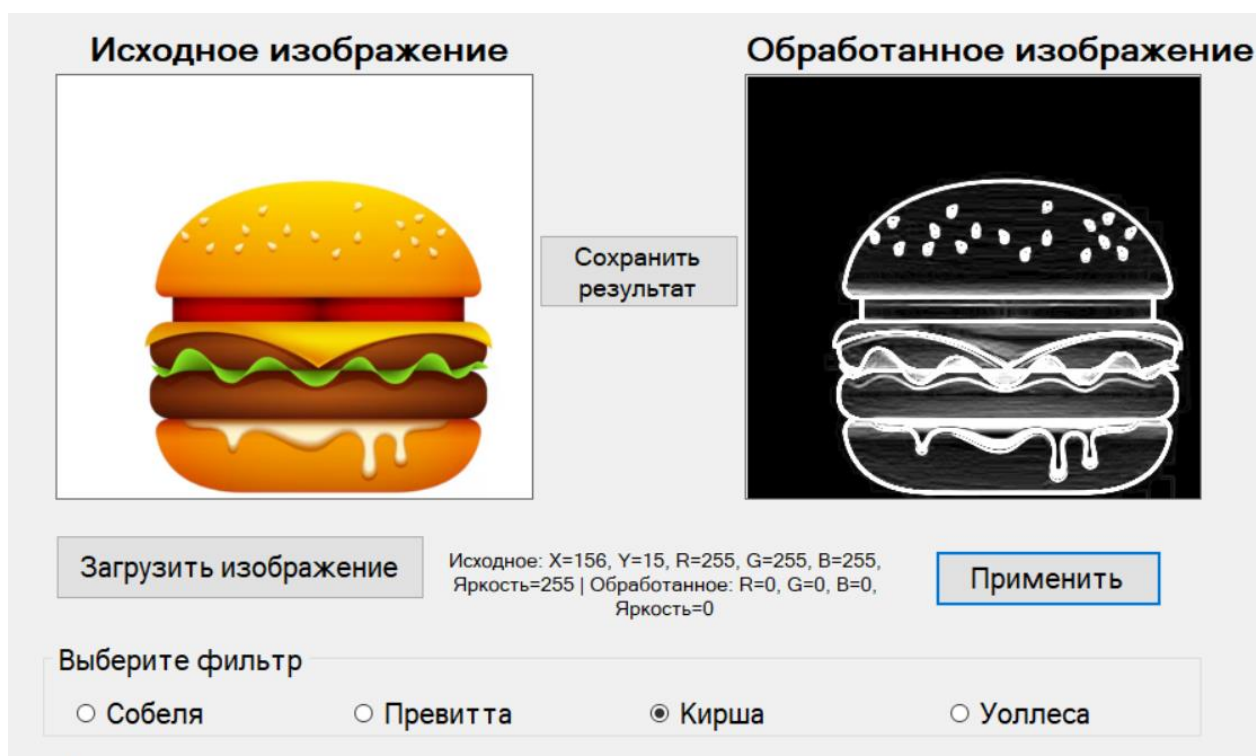


Рисунок 13 – Загруженное изображение (фильтр Кирша, апертура 3x3)

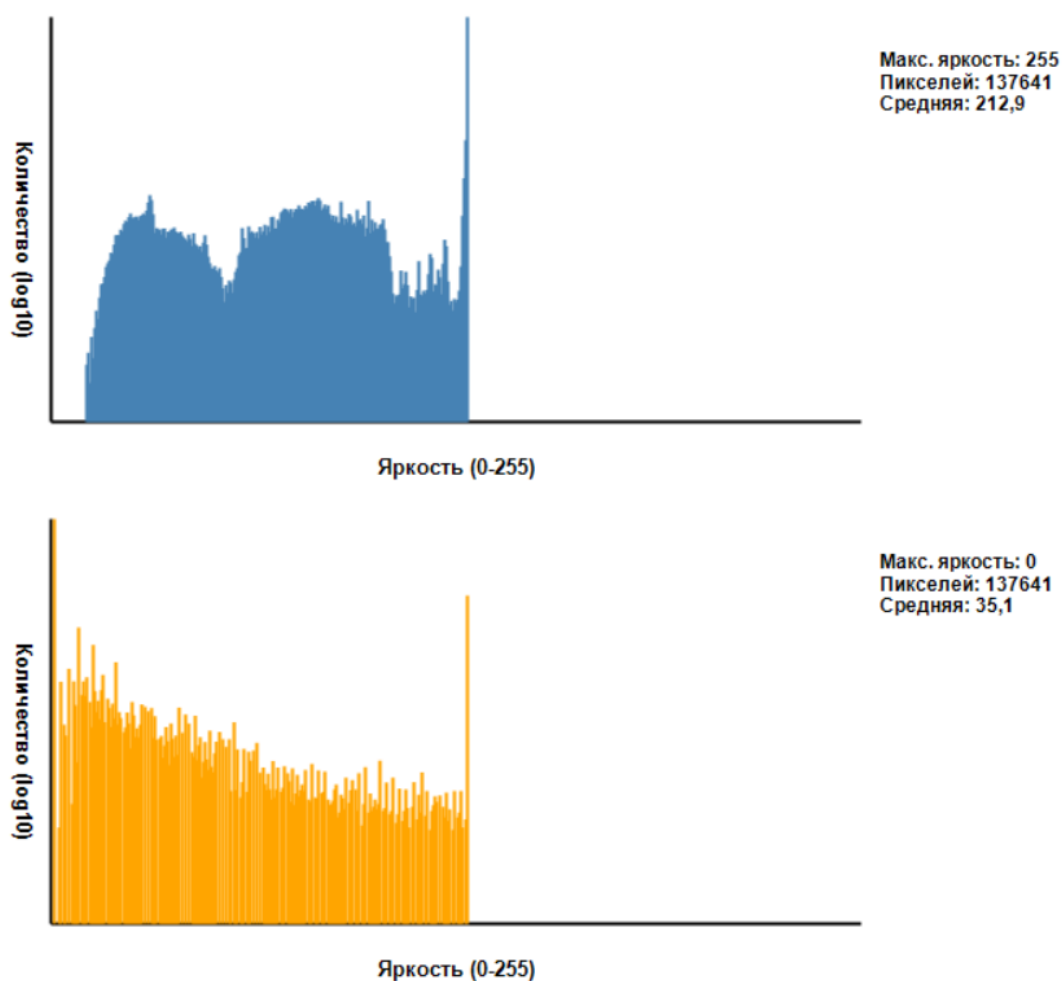


Рисунок 14 – Загруженное изображение (гистограммы, фильтр Кирша)

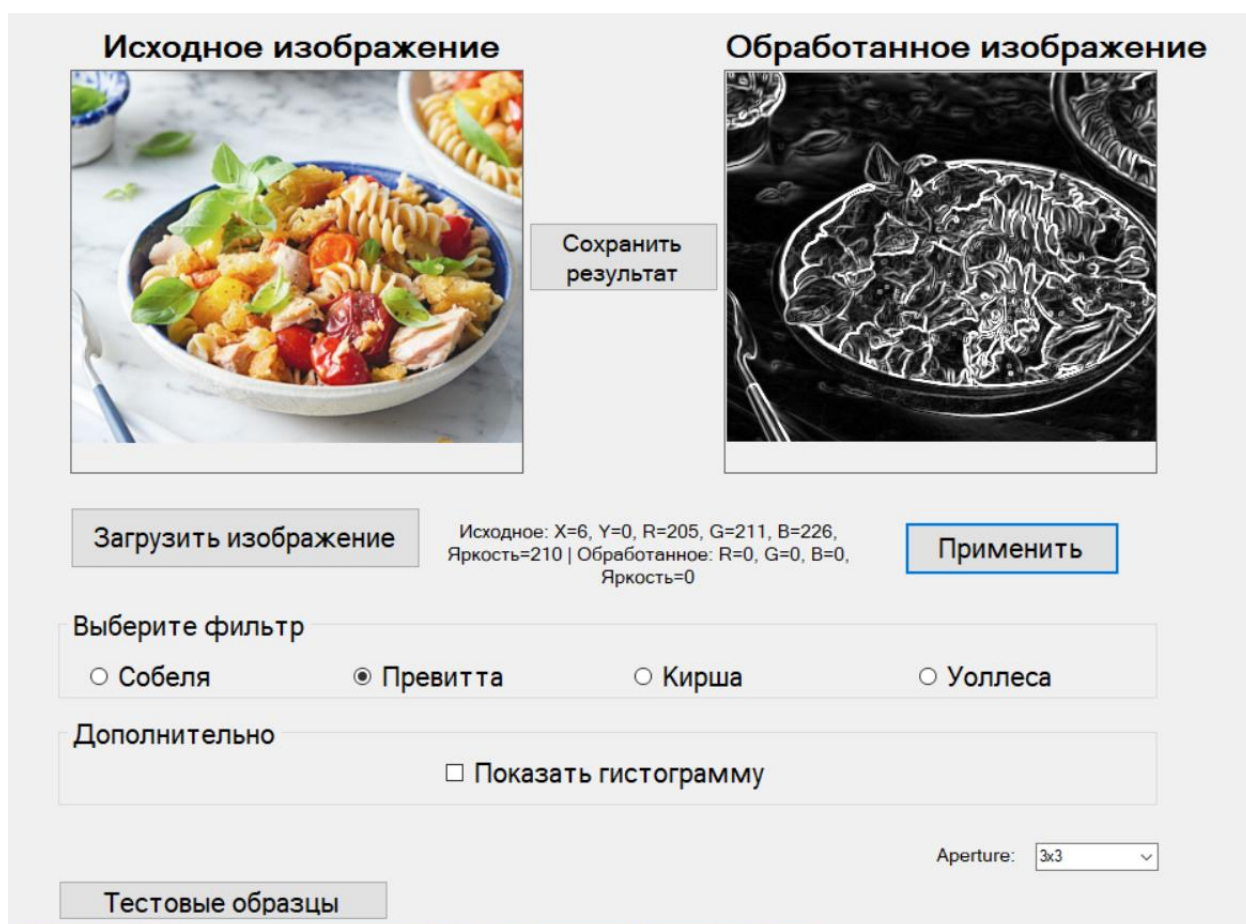


Рисунок 15 – Загруженное изображение (фильтр Превитта, апертура 3x3)

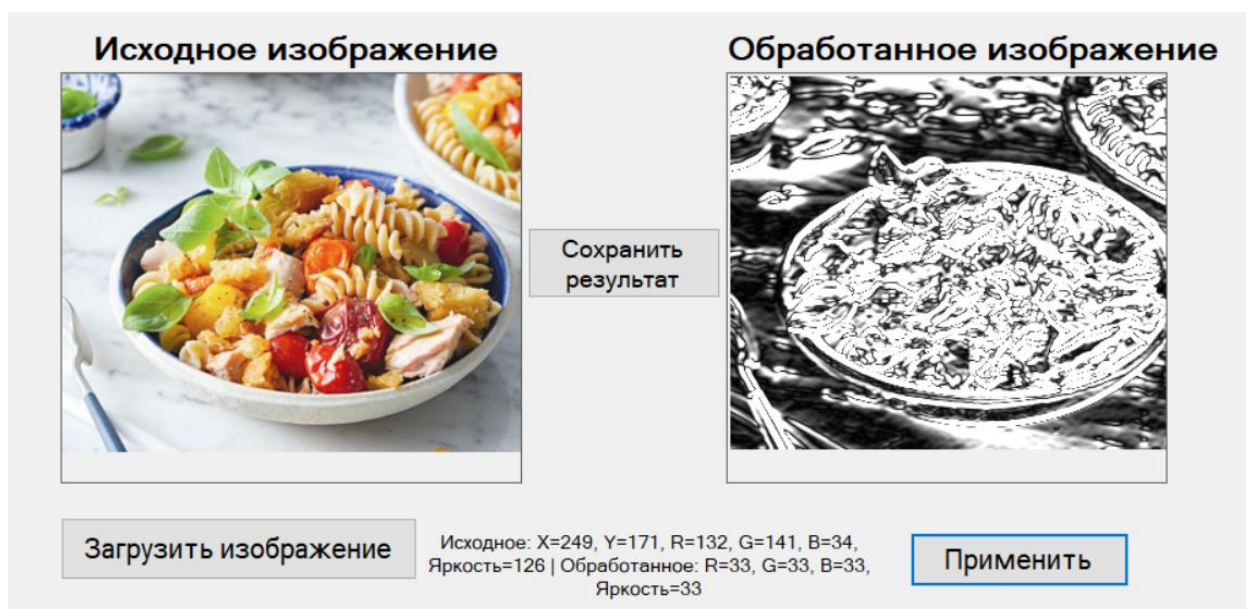




Рисунок 16 – Загруженное изображение (фильтр Превитта, апертура 5x5)

Исходное изображение



Обработанное изображение



Сохранить результат

Загрузить изображение

Исходное: X=184, Y=73, R=227, G=195, B=87, Яркость=192 | Обработанное: R=152, G=152, B=152, Яркость=152

Применить

Выберите фильтр

☐ Собеля

☐ Превитта

☐ Кирша

☒ Уоллеса

Дополнительно

☐ Показать гистограмму

Mean: 128


Variance: 50

Gain: 1


Тестовые образцы

Рисунок 17 – Загруженное изображение (фильтр Уоллеса, Mean = 128, Variance = 50, Gain = 1)

Исходное изображение



Обработанное изображение



Сохранить результат

Загрузить изображение

Исходное: X=243, Y=126, R=231, G=202, B=138, Яркость=203 | Обработанное: R=35, G=35, B=35, Яркость=35

Применить

Выберите фильтр

☐ Собеля

☐ Превитта

☐ Кирша

☒ Уоллеса

Дополнительно

☐ Показать гистограмму

Mean: 32

Variance: 25

Gain: 1

Тестовые образцы

Рисунок 18 – Загруженное изображение (фильтр Уоллеса, Mean = 32, Variance = 25, Gain = 1)

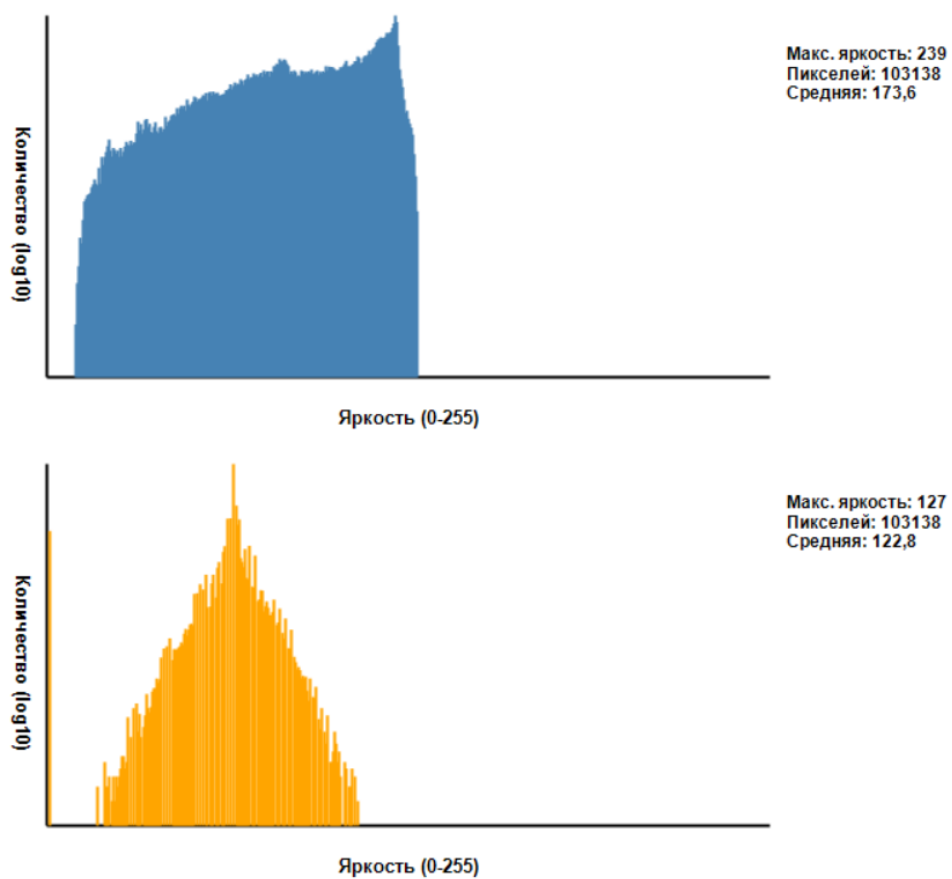


Рисунок 19 – Загруженное изображение (фильтр Уоллеса, Mean = 128, Variance = 50, Gain = 1, гистограммы)

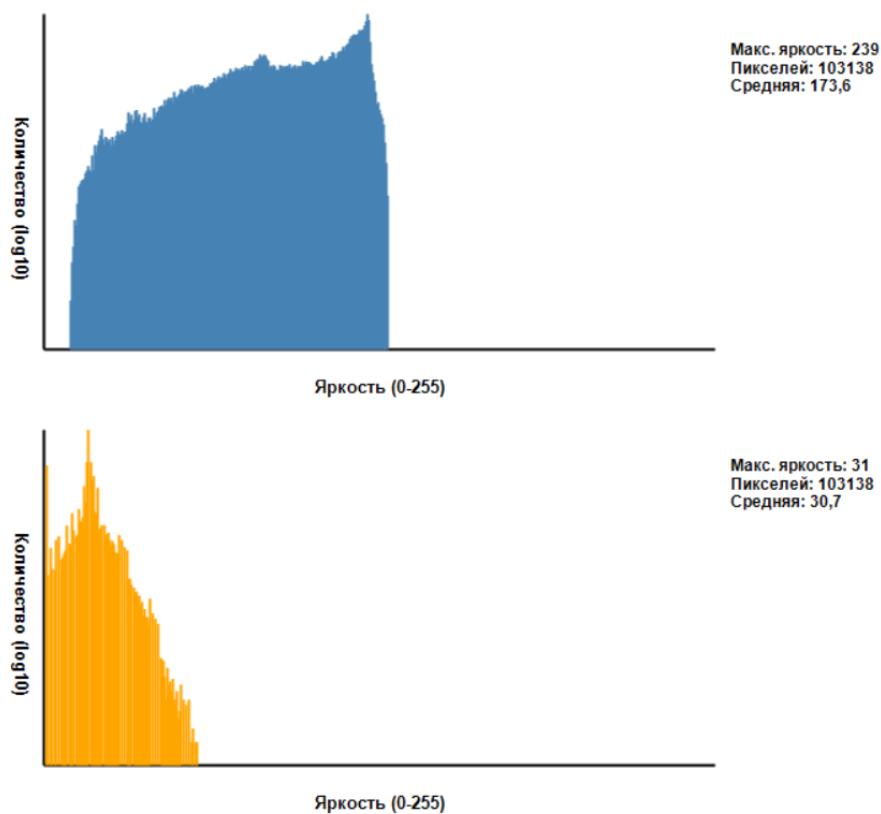


Рисунок 20 – Загруженное изображение (фильтр Уоллеса, Mean = 32, Variance = 25, Gain = 1, гистограммы)

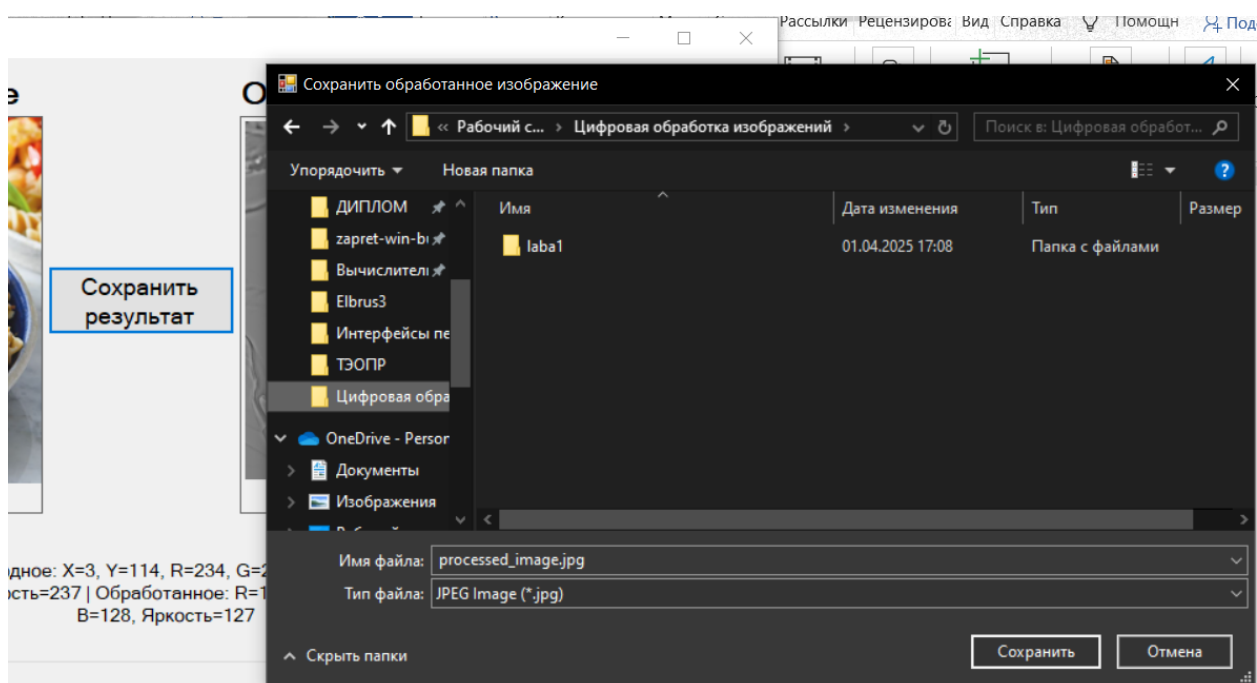


Рисунок 21 – Сохранение результата

Вывод

В ходе выполнения данной лабораторной работы было разработано Windows-приложение для цифровой обработки изображений, реализующее 4 фильтра выделения контуров. Приложение успешно реализовало все поставленные задачи, предоставив удобный интерфейс для обработки и анализа изображений.