Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка приложения для фильтрации и обработки изображений»

Студент группы **ИУ7-53Б** _____ **В.С. Винокуршин**
(Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы _____ **А.О. Оленев**
(Подпись, дата) (И.О. Фамилия)

***2022** г.*

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **5**

1 Аналитическая часть **7**

1.1 Представление графической информации в компьютере 7

1.2 Выбор грифического формата 8

1.2.1 Формат BMP 9

1.3 Основные понятия и обработка изображений 9

1.4 Фильтры обработки изображения 10

1.4.1 Точечные фильтры 10

1.4.2 2D линейная фильтрация изображений 12

1.4.3 Свойства линейных фильтров 14

1.4.4 Нелинейная фильтрация изображений 15

2 Конструкторская часть **17**

2.1 Точечные фильтры 17

2.1.1 Нетагив 17

2.1.2 Преобразование к оттенкам серого 17

2.2 Матрица свёртки 17

2.3 Фильтр размытия 18

2.3.1 Нормальное распределение 18

2.3.2 Функция Гаусса 19

2.4 Медианный фильтр 20

2.5 Проблема матричных фильтров 20

2.6 Выбор используемых типов и структур данных 21

3 Технологическая часть **25**

3.1 Требования к программе 25

3.2 Выбор языка программирования и среды разработки 25

3.3 Реализация алгоритмов обработки изображений 26

3.3.1 Изменение яркости и контрастности изображения 26

3.3.2 Фильтры 28

3.4 Интерфейс программы 31

3.5 Демонстрация работы программы 32

4 Исследовательская часть **37**

4.1 Технические характеристики 37

4.2 Постановка эксперимента 37

4.2.1 Цель эксперимента 37

4.2.2 Проведение эксперимента 37

ЗАКЛЮЧЕНИЕ **40**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **41**

ВВЕДЕНИЕ

В современном мире использование изображений растет с каждым днем. В конце концов, большинство людей легче воспринимают любую информацию именно визуально. В то же время, наряду со значительным повышением уровня развития технологий, методы обработки изображений играют очень значительную роль. Вот уже около двухсот лет процесс улучшения фотографий продолжается и не прекращается даже сегодня. Новые формы обработки изображений рождаются каждый день, некоторые совершенствуются, другие заменяются более новыми и интересными. Благодаря современным технологиям методы обработки изображений полностью меняют представление об изображении.

Области применения цифровой обработки изображений весьма разнообразны. Она имеет свое распространение во многих науках, в создании рекламы и т.д. Если вернуться на несколько лет назад, то можно вспомнить, что фотографии того времени были в основном черно-белыми, лишь постепенно начинавшими приобретать цвет. Но с годами они теряют свое прежнее состояние, теряют свое качество. С совершенствованием технологии удалось добиться того, что вернуть фотографии первоначальный вид – дело нескольких минут. Обработка изображений обеспечивает улучшение изображения, сжатие данных для хранения и передачи по каналам связи, а также анализ, распознавание и интерпретацию визуальных изображений.

Цель данной работы – реализовать программу, способную помочь пользователю обработать выбранное им изображение, а также, наложить на него один или несколько известных фильтров.

Чтобы достичь поставленной цели, требуется решить следующие задачи:

— выбрать формат изображения, над которым будут проводиться операции для редактирования изображения;
— определить методы фильтрации изображения, рассмотреть их основные характеристики;

— познакомиться с известными фильтрами обработки изображения;

— полученные знания применить на практике.

1 Аналитическая часть

1.1 Представление графической информации в компьютере

Любая информация, в том числе и графическая, может быть представлена в аналоговой или дискретной форме. При аналоговом представлении физическая величина принимает бесконечное множество значений, причем ее значения изменяются непрерывно. При дискретном представлении физическая величина принимает конечное множество значений, причем ее величина изменяется скачкообразно. Преобразование графической информации из аналоговой формы в дискретную производится путем пространственной дискретизации. Пространственную дискретизацию можно сравнить с построением изображения из мозаики. Изображение разбивается на отдельные маленькие фрагменты (точки), причем каждому фрагменту присваивается значение его цвета, то есть код цвета.

Качество кодирования изображения зависит от двух параметров. Оно тем выше, чем меньше размер точки и соответственно большее количество точек составляет изображение; чем большее количество цветов, то есть большее количество возможных состояний точки изображения, используется.

Графическая информация на экране монитора представляется в виде растрового изображения, которое формируется из определенного количества строк, которые в свою очередь содержат определенное количество точек.

Качество изображения определяется разрешающей способностью монитора, т.е. количеством точек, из которых оно складывается. Чем больше разрешающая способность, т.е. чем больше количество строк растра и точек в строке, тем выше качество изображения. В современных персональных компьютерах обычно используются три основные разрешающие способности экрана: 1920×1080, 1336×768 и 1536×864 точки [1].

В простейшем случае (черно-белое изображение без градаций серого цвета) каждая точка экрана может иметь одно из двух состояний – «черная» или

«белая», то есть для хранения ее состояния необходим 1 бит.

Цветные изображения формируются в соответствии с двоичным кодом цвета каждой точки, хранящимся в видеопамяти. Они могут иметь различную глубину цвета, которая задается количеством битов, используемым для кодирования цвета точки. Наиболее распространенными значениями глубины цвета являются 8, 16, 24 или 32 бита. Каждый цвет можно рассматривать как возможное состояние точки, тогда количество цветов, отображаемых на экране монитора, может быть вычислено по формуле: $N = 2^I$, где I – глубина цвета.

Цветное изображение на экране монитора формируется за счет смешивания трех базовых цветов: красного, зеленого и синего. Такая цветовая модель называется RGB-моделью.

Для получения богатой палитры цветов базовым цветам могут быть заданы различные интенсивности. Например, при глубине цвета в 24 бита на каждый из цветов возможны $N = 2^8 = 256$ уровней интенсивности, заданные двоичными кодами (от минимальной – 00000000 до максимальной – 11111111) [2].

1.2 Выбор грифического формата

Графический формат определяет способ сохранения графической информации в файле, а также форму хранения информации (используемый алгоритм сжатия). Как и изображения, графические форматы делятся на растровые и векторные. В компьютерной графике применяют, по меньшей мере, три десятка форматов файлов для хранения изображений. Единого формата графических файлов, пригодного для всех приложений, не существует, поэтому имеет место быть проблема сохранения изображений для последующей их обработки. Однако некоторые форматы стали стандартными для целого ряда предметных областей. Крайне важно различать векторные (WMF, DXF, CGM и др.) и растровые (JPEG, PNG, GIF, BMP, ICO и др.) форматы [3].

Файлы векторного формата содержат описания рисунков в виде набора простейших графических объектов. В файлах же растровой графики запоминается цвет каждого пикселя на рисунке, поэтому такие файлы занимают, как

правило, большой объем памяти. Один из возможных способов решения этой проблемы – сжатие информации, т. е. уменьшение размеров файла за счет изменения способа организации данных в нем. Обычно каждый конкретный алгоритм хорошо сжимает только изображения вполне определенной структуры.

Для решения задач данной работы был выбран формат BMP.

1.2.1 Формат BMP

BMP (полное имя Bitmap) – это стандартный формат файла изображения в операционной системе Windows, который можно разделить на две категории: зависящее от устройства растровое изображение (DDB) и независимое от устройства растровое изображение (DIB), которое очень широко используется. Он использует формат хранения растровых изображений, в дополнение к дополнительной глубине изображения, не использует никакого другого сжатия, поэтому пространство, занимаемое файлом BMP, очень велико. Глубина изображения файлов BMP может быть выбрана из l, 4, 8, 24 и 32 бит. При хранении данных в файле BMP изображение сканируется в порядке слева направо и снизу вверх. Поскольку формат файла BMP является стандартом для обмена данными, связанными с графиками, в среде Windows, все графические и графические программы, работающие в среде Windows, поддерживают формат изображений BMP [4].

Этот формат был выбран, так как его удобно использовать при создании картинок, которые не теряют качества после их изменений, то есть изображения в этом формате можно редактировать какое угодно количество раз, а их качество ничуть не пострадает (в отличие от того же JPEG).

1.3 Основные понятия и обработка изображений

Изображением называется двумерная функция $f(x, y)$ в которой x и y – координаты в пространстве (на плоскости), а значение функции которой в любой точке, определяется парой координат (x, y), называется интенсивностью изображения в данной точке. Изображение будет являться цифровым, если величины x, y и функция принимают конечное число дискретных значений.

Цифровой обработкой изображений называется обработка цифровых изображений, которые обрабатываются благодаря цифровым вычислительным машинам. Цифровое изображение состоит из конечного числа элементов, каждый из которых расположен в конкретном месте и принимает определенное значение, соответственно называются элементами изображения или пикселями. Для элементов цифрового изображения чаще всего используется термин «пиксель» [5].

Наличие массовой памяти большого объема обязательно для практических задач обработки изображений. Для хранения изображения размером 1024×1024 пикселя, в котором яркость каждого пикселя представляется 8-битовой величиной, необходим один мегабайт памяти, если не используются средства сжатия изображений. При работе с тысячами или даже миллионами изображений наличие достаточной внешней памяти в системе обработки изображений может оказаться проблематичным.

Пиксель цифрового изображения является оптически однородным, и внутри его отдельные элементы изображения не выделяются. Экспе-риментально установлено, что для воспроизведения на цифровом снимке компактного объекта его размер должен быть не менее четырех пикселей.

Растровое изображение строится из составляющих его пикселей, размещаемых построчно слева направо и сверху вниз, а доступ к какому-либо из них осуществляется по номеру соответствующего столбца (ix) и строки (jy) [6].

Растровые координаты пикселя (ix) и (iy) относятся к его центру, хотя с помощью математического аппарата они могут быть найдены с точностью порядка 0,1 от его размера. В этих случаях для доступа к пикселю используется целая часть его растровых координат.

1.4 Фильтры обработки изображения

1.4.1 Точечные фильтры

Самый простой фильтр – это оператор точки. Каждое значение пикселя умножается на скалярное значение. Эта операция может быть записана следу-

ющим образом (1):

$$g(i,j) = K \times f(i,j), \tag{1}$$

где:

— входное изображение равно F, а значение пикселя в точке (i, j) обозначается как $f(i,j)$;

— выходное изображение равно G, а значение пикселя в (i, j) обозначается как $g(i,j)$;

— K – скалярная константа.

Этот тип операций с изображением известен как линейный фильтр. В дополнение к умножению на скалярное значение, каждый пиксель также может быть увеличен или уменьшен на постоянное значение [7]. Таким образом, общая операция с точками может быть записана следующим образом (2):

$$g(i,j) = K \times f(i,j) + L \tag{2}$$

Эта операция может быть применена как к изображениям в оттенках серого, так и к изображениям RGB. Для изображений RGB каждый канал будет изменен с помощью этой операции отдельно. Следующее является результатом изменения как K, так и L. На втором изображении K = 0,5 и L = 0,0, в то время как на третьем изображении K равно 1,0, а L равно 10. Для конечного изображения справа K = 0,7 и L = 25. Как можно заметить, изменение K меняет яркость изображения, а изменение L меняет контрастность изображения (рисунок 1).

Рисунок 1 – Пример точечной фильтрации

1.4.2 2D линейная фильтрация изображений

В то время как предыдущий фильтр является точечным фильтром, пиксели изображения также содержат информацию вокруг пикселя. Для извлечения этой информации при фильтрации существует несколько фильтров окрестностей.

Пусть у нас есть следующее исходное изображение (рисунок 2):

0	1	1	0
0	0	0	1
0	0	1	0
0	1	1	1

Рисунок 2 – Исходное изображение двойки

Это простое двоичное изображение числа 2. Чтобы получить определенную информацию из этого изображения, можно напрямую использовать все значения пикселей. Но вместо этого, для упрощения, можно применить к этому фильтры. Определяется матрицу, меньшую, чем заданное изображение, которая работает в окрестности целевого пикселя. Эта матрица называется ядром, пример такой матрицы представлен на рисунке 3.

Операция определяется сначала наложением матрицы ядра на исходное изображение, затем взятием произведения соответствующих пикселей и возвратом суммы всех произведений. На рисунке 4 нижняя область размером 3×3 в исходном изображении накладывается на заданную матрицу ядра, и соответствующие значения пикселей из ядра и изображения умножаются. Результиру-

0	0	0
0	1	0
0	0	0

Рисунок 3 – Пример ядра

ющее изображение показано справа и представляет собой суммирование всех предыдущих произведений пикселей:

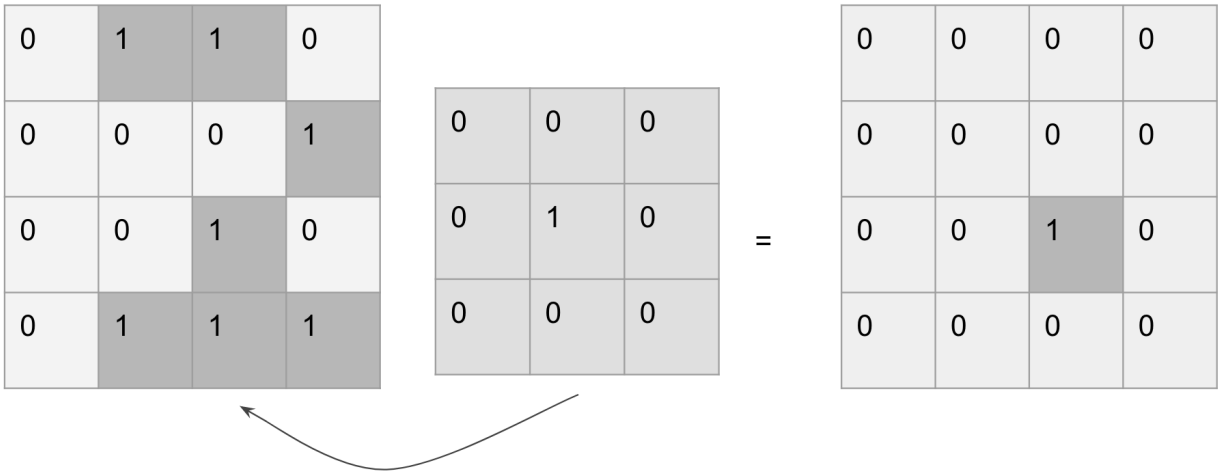

Рисунок 4 – Пример применения матрицы ядра

Эта операция повторяется путем перемещения ядра по строкам изображений, а затем по столбцам изображений [7]. От матрицы свертки зависит результат преобразования изображения.

Ядро фильтра, заданное на прямоугольной окрестности N, может рассматриваться как матрица $n \times n$, где длины сторон являются нечетными числами. При задании ядра матрицей ее следует центрировать. Если пиксель (x, y) находится в окрестности краев изображения, то координаты A(x-i, y-j) для определенных (i, j) могут соответствовать несуществующим пикселям A за пределами изображения.

Данную проблему можно разрешить несколькими способами:

— не проводить фильтрацию для таких пикселей, обрезав изображение B по краям, или применив для их значений исходные значения изображения A;
— не включать отсутствующий пиксель в суммирование, распределив его вес h(i, j) равномерно среди других пикселей окрестности N(x, y);
— доопределить значения пикселей за границами изображения при помощи экстраполяции;
— доопределить значения пикселей за границами изображения, при помощи зеркального продолжения изображения.

Выбор способа производится с учетом конкретного фильтра и особенностей изображения.

1.4.3 Свойства линейных фильтров

Несколько приложений компьютерного зрения состоят из пошаговых преобразований входной фотографии в выходной. Это легко сделать благодаря нескольким свойствам, связанным с распространенным типом фильтров, то есть с линейными фильтрами.

1. Линейные фильтры являются коммутативными, так что можно выполнять операции умножения над фильтрами в любом порядке, и результат все равно остается тем же:

$$a \times b = b \times a. \tag{3}$$

2. Они носят ассоциативный характер, что означает, что порядок применения фильтра не влияет на результат:

$$(a \times b) \times c = a \times (b \times c). \tag{4}$$

3. Даже в случае суммирования двух фильтров можно выполнить первое суммирование, а затем применить фильтр, или также можно индивидуально применить фильтр, а затем суммировать результаты. Общий результат остается прежним.

4. Применение коэффициента масштабирования к одному фильтру и умножение на другой фильтр эквивалентно первому умножению обоих фильтров, а затем применению коэффициента масштабирования.

Эти свойства играют важную роль в других задачах компьютерного зрения, таких как обнаружение объектов и сегментация. Подходящая комбинация этих фильтров повышает качество извлечения информации и, как следствие, повышает точность.

1.4.4 Нелинейная фильтрация изображений

Хотя во многих случаях для получения требуемых результатов достаточно линейных фильтров, в нескольких других вариантах использования производительность может быть значительно увеличена с помощью нелинейной фильтрации изображений. Нелинейная фильтрация изображений является более сложной, чем линейная фильтрация. Однако эта сложность может дать вам больше контроля и лучшие результаты в ваших задачах компьютерного зрения. Она позволяет избежать дополнительного искажения изображения при удалении шума, а также значительно улучшить результаты работы фильтров на изображениях с высокой степенью зашумленности [8].

Основное отличие нелинейного фильтра от линейного заключается в том, что выход нелинейного фильтра формируется нелинейным образом от данных исходного изображения. Линейные фильтры, несмотря на разнообразие производимых ими эффектов, не позволяют проделывать некоторые самые естественные операции.

Исходные данные нелинейного фильтра и результат фильтрации находятся в логической взаимосвязи, то есть реализуются логическими операциями, такими как максимальный фильтр, минимальный фильтр, средний фильтр и т. д., Путем сравнения значений в определенной окрестности. Для этого нет фиксированного шаблона, поэтому нет конкретной передаточной функции.

Хорошим примером служит пороговая фильтрация. Результатом пороговой фильтрации служит бинарное изображение, определяемое следующим об-

разом (5):

$$g(x,y) = \begin{cases} 1, & \text{если } f(x,y) > \gamma \\ 0, & \text{иначе} \end{cases} \tag{5}$$

Величина γ является порогом фильтрации. В приложениях используется еще целый ряд простейших нелинейных фильтров. Например, модуль изображения, содержащего пиксели с отрицательным значением, или фильтр, обнуляющий все значения пикселей, меньше данного порога.

Выводы

Таким образом, можно сделать вывод, что обрабатывать изображения можно различными способами. Все они имеют различную степень сложности и эффективность, поэтому, в программе будут реализованы все тип обработки изображений, описанные выше.

2 Конструкторская часть

В данном разделе будут рассмотрены конкретные алгоритмы, которые были применены для обработки изображений, а также, будут описаны структуры данных, используемые в данной работе.

2.1 Точечные фильтры

2.1.1 Нетагив

Для применения этого фильтра все составляющие цвета каждого пискселя (R, G и B) нужно инвертировать – вычесть текущее значение из 255 (так как максимальное значение – 256).

2.1.2 Преобразование к оттенкам серого

Заключается в получении яркости каждой точки по формуле $Y = 0.3 * R + 0.59 * G + 0.11 * B$ и последующем копировании полученного значения во все три канала (R, G и B).

2.2 Матрица свёртки

Матрица свёртки – это матрица коэффициентов, которая «умножается» на значение пикселей изображения для получения требуемого результата [9].

Алгоритм применения следующий: начиная с верхнего левого пикселя, выделяется ядро размером матрицы свертки (рис. 5); проводится перемножение между элементами матрицы свертки и выделенного ядра, для примера, показанного на рисунке 5, результат будет следующим: $6 \cdot 0 + 6 \cdot 1 + 1 \cdot 0 + 3 \cdot 1 + 7 \cdot 2 + 5 \cdot 1 + 2 \cdot 0 + 3 \cdot 1 + 1 \cdot 0 = 31$. Далее вычисляется коэффициент нормирования, суммируя компоненты матрицы свёртки. Для примера это значение будет равно $0 + 1 + 0 + 1 + 2 + 1 + 0 + 1 + 0 = 6$; результат перемножения матриц делится на коэффициент нормирования: $\frac{31}{6} = 5.167$; полученное значение присваевается центральному компоненту ядра. Весь цикл повторяется для каждого пикселя изображения.

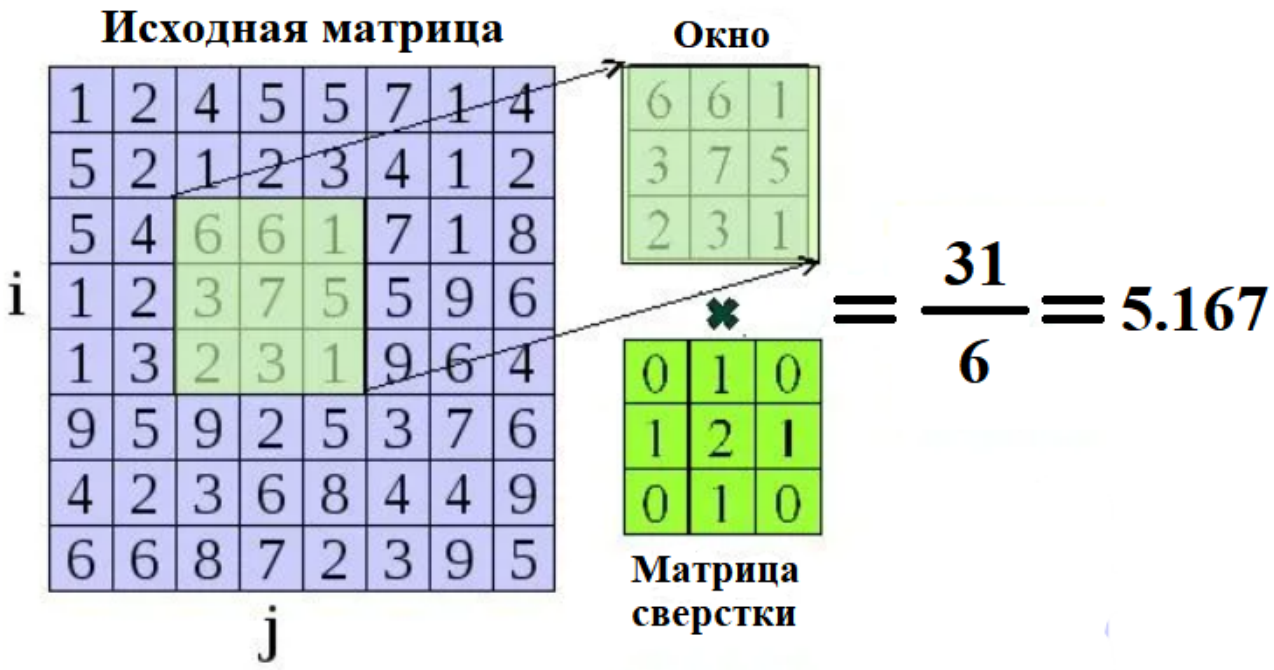

Рисунок 5 – Применение матрицы сверстки

2.3 Фильтр размытия

Наиболее часто используемым фильтром, основанным на матрице свёртки, является фильтр размытия. Фильтр меняет каждую точку выбранного ядра, делая её значение равным значению всех точек в определённом радиусе от рассматриваемой точки. Значение этого радиуса (размер матрицы) можно изменить.

Обычно матрица заполняется по нормальному (Гауссовому) закону.

2.3.1 Нормальное распределение

Главная особенность нормального закона распределения заключается в том, что он является предельным законом для других законов распределения. Нормальное распределение случайной величины – одно из непрерывных распределений, имеющее основополагающую роль в математической статистике [10].

В нормальном распределении, чем ближе к центральной точке, тем больше значение, а чем дальше от центра, тем меньше значение (рисунок 6). При вычислении среднего значения нам нужно только использовать «центральную

точку» в качестве начала координат и присвоить веса другим точкам в соответствии с их положениями на нормальной кривой, чтобы получить средневзвешенное значение. Нормальное распределение, очевидно, является желательной моделью распределения веса.

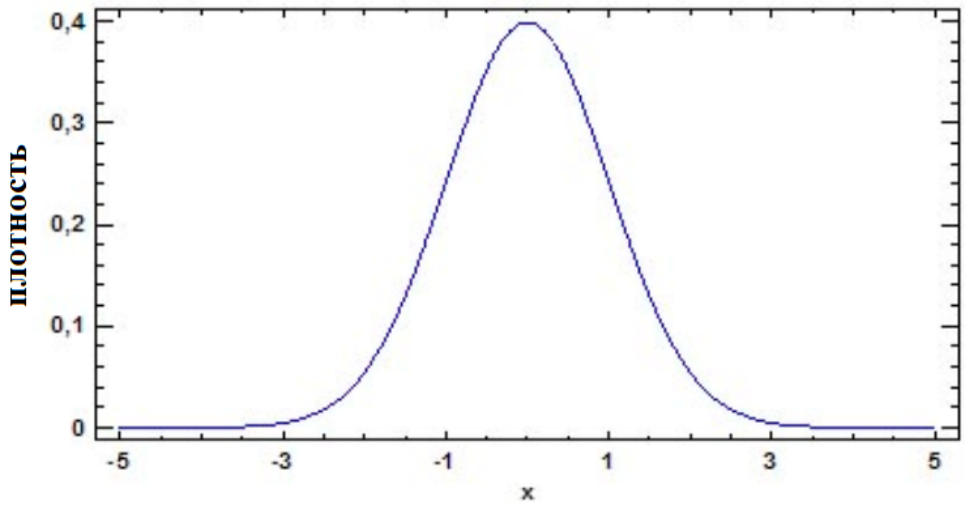

Рисунок 6 – Нормальное распределение

2.3.2 Функция Гаусса

Функция плотности нормального распределения называется «функцией Гаусса» [11]. Его одномерная форма (6).

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp^{-(x-\mu)^2/2\sigma^2} \tag{6}$$

где: μ — среднее значение x, σ — дисперсия[1] x. Поскольку при вычислении среднего значения центром является начало координат, поэтому μ равно 0 (7).

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp^{-x^2/2\sigma^2} \tag{7}$$

В соответствии с одномерной функцией Гаусса, двумерная функция Гаусса может быть получена как (8).

$$G(x, y) = \frac{1}{2\pi\sigma^2} \cdot \exp^{-(x^2+y^2)/2\sigma^2} \tag{8}$$

[1]Дисперсия — это величина, показывающая, как именно и насколько сильно разбросаны значения.

С помощью этой функции можно рассчитать вес каждой точки.

В таблице 1 приведена матрица размытия 5x5, заполненная по закону Гауссовского распределения.

0.000789	0.006581	0.013347	0.006581	0.000789
0.006581	0.054901	0.111345	0.054901	0.006581
0.013347	0.111345	0.225821	0.111345	0.000789
0.006581	0.054901	0.111345	0.054901	0.006581
0.000789	0.006581	0.013347	0.006581	0.000789

Таблица 1 – Матрица размытия 5x5, заполненная по закону Гауссовского распределения

Матрица уже нормированна, поэтому, ее определитель равен 1. От размера матрицы зависит сила размытия.

2.4 Медианный фильтр

Медианный фильтр является примером нелинейной фильтрации, обычно используется для уменьшения шума или «сглаживания» изображения.

Фильтр работает с матрицами различного размера, но в отличие от матрицы свёртки, размер матрицы влияет только на количество рассматриваемых пикселей. Алгоритм медианного фильтра следующий: для текущего пикселя, пиксели, которые «попадают» в матрицу, сортируются, и выбирается средние значение из отсортированного массива. Это значение и является выходным для текущего пикселя.

На рисунке 7 представлена работа медианного фильтра для размера ядра равного трём.

2.5 Проблема матричных фильтров

Стоит упомянуть о граничных условиях (эта проблема актуальна для всех матричных фильтров). У верхнего левого пикселя не существует «соседа» с права от него, следовательно, нам не на что умножать коэффициент матрицы

Рисунок 7 – Работа медианного фильтра

(рисунок 8).

Существует 2 решения этой проблемы.

Первое решение заключается в применении фильтра, только к «окну» изображения (рисунок 9), которое имеет координаты левого верхнего угла $(kernelSize/2, kernelSize/2)$, а для правого нижнего $(width—kernelSize/2, height—kernelSize/2)$, где kernelSize – размер матрицы, а width и height – размеры изображения.

Второй метод (рисунок 10) требует создания промежуточного изображения. Идея в том, чтобы создавать временное изображение с размерами $(width + 2 * kernelSize/2, height + 2 * kernelSize/2)$. В центр изображения копируется входная картинка, а края заполняются крайними пикселями изображения. Размытие применяется к промежуточному буферу, а потом из него извлекается результат.

Данный метод не имеет недостатков в качестве, но необходимо производить лишние вычисления.

2.6 Выбор используемых типов и структур данных

Для разрабатываемого ПО необходимо реализовать следующие типы и структуры данных.

1. Структура BITMAPFILEHEADER – заголовок файла BMP:

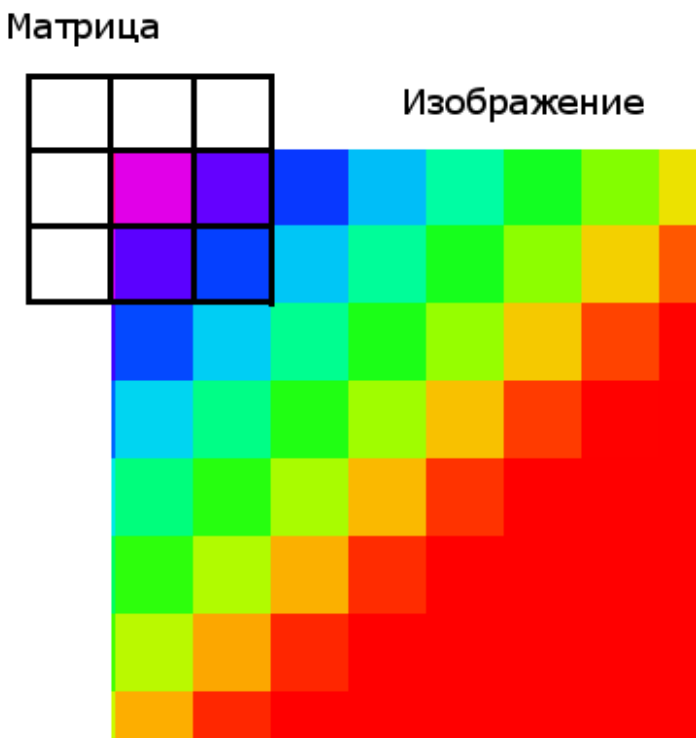

Рисунок 8 – Проблема применения матричных фильтров

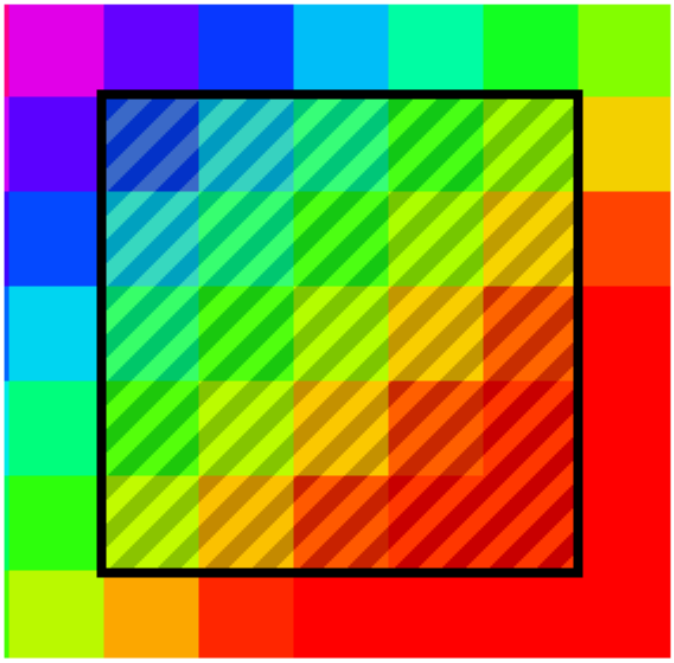

Рисунок 9 – Применение фильтра, только к «окну» изображения

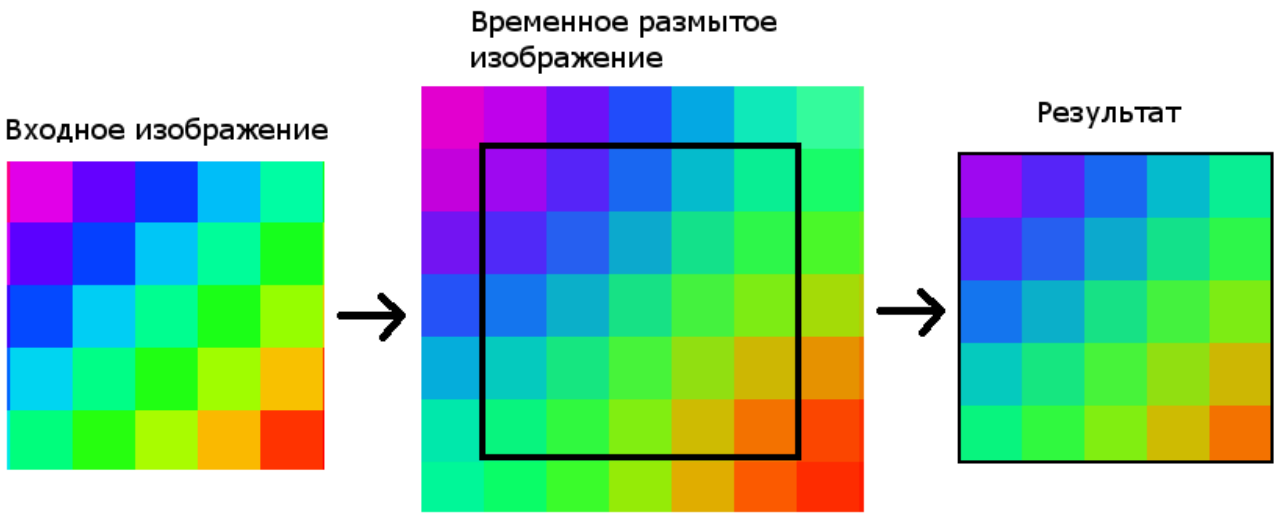

Рисунок 10 – Дополнение

— $unsigned\ int\ bfType$ – тип файла;

— $unsigned\ long\ bfSize$ – размер в байтах;

— $unsigned\ int\ bfReserved1$ и $bfReserved2$ – зарезервированные (нули);

— $unsigned\ long\ bfOffBits$ – смещение.

2. Структура BITMAPINFOHEADER – содержание файла:

— $unsigned\ int\ biSize$ – тип размер структуры;

— $size_t\ biWidth$ и $biHeight$ – ширина и высота картинки соответственно;

— $unsigned\ short\ biPlanes$ – количество плоскостей;

— $unsigned\ short\ biBitCount$ – количество бит на пиксель;

— $unsigned\ int\ biCompression$ – тип сжатия;

— $unsigned\ int\ biSizeImage$ – размер картинки в байтах;

— $int\ biXPelsPerMeter$ и $biYPelsPerMeter$ – горизонтальное и вертикальное разрешение соответственно;

— $unsigned\ int\ biClrUsed$ – количество исп. цветов;

— $unsigned\ int\ biClrImportant$ – количество важных цветов.

3. Структура RGBQUAD – пиксель в формате RGB:

— $double\ rgbBlue$ – –;

— $double\ rgbGreen - -$;

— $double\ rgbRed - -$;

— $double\ rgbReserved - -$.

4. Cnhernehf PIX – пиксель:

— $double\ value$ – значение пикселя;

— $RGBQUAD\ rgb$ – его значение в RGB.

Выводы

В данном разделе были рассмотрены конкретные алгоритмы, которые применялись для обработки изображений, была рассмотрена проблема, которая встречается при работе с матричными фильтрами, и пути ее решения. Кроме того, были описаны выбранные структуры данных.

3 Технологическая часть

В данном разделе будут рассмотрены детали реализации программного комплекса, описанного в конструкторской части работы, представлены листинги кода алгоритмов обработки изображений и приведены примеры работы программы.

3.1 Требования к программе

Программа должна предоставлять следующие возможности:

— отображение загруженного изображения, а также результатов его обработки;
— применение фильтров к изображению;
— настройка яркости и контрастности изображения;
— поворот изображения на 90 градусов по и против часовой стрелки;
— операции отмены и повторения отмененного действия;
— сохранение изображения.

3.2 Выбор языка программирования и среды разработки

Существует множество языков, а также сред программирования, многие из которых обладают достаточно высокой эффективностью, удобством и простотой в использовании. Для разработки данной программы был выбран язык C++ [12]. Данный выбор был обусловен следующими факторами:

1. Этот язык предоставляет программисту широкие возможности реализации самых разнообразных алгоритмов. Он обладает высокой эффективностью и большим набором стандартных классов и процедур.
2. Язык является объектно-ориентированным, что позволяет объединить всю логику обработки изображения в класс.
3. Язык является строго типизированным, что позволяет защититься от неконтролируемых ошибок.
4. В данном языке имеется большое количество библиотек и шаблонов, позволяющих не тратить время на изобретение готовых конструкций.

В качестве среды разработки был выбран фреймворк Qt [13] в силу следующих факторов:

— широкие возможности работы с изображениями, в том числе и попиксельно;
— наличие более функциональных аналогов стандартной библиотеки шаблонов в том числе для разнообразных структур данных;
— возможность написать интерфейс для программы.

3.3 Реализация алгоритмов обработки изображений

3.3.1 Изменение яркости и контрастности изображения

В листинге 1 представлены алгоритмы изменения яркости и контрастности изображений.

Листинг 1: Изменение яркости и контрастности изображения

```
// Изменение яркости
void Image::changeBrightness(const int &value) {
    for (size_t i = 0; i < bmiHeader.biHeight; i++) {
        for (size_t j = 0; j < bmiHeader.biWidth; j++) {
            pixels[i][j].rgbRed += value;
            pixels[i][j].rgbGreen += value;
            pixels[i][j].rgbBlue += value;
        }
    }
}

// Изменение контрастности
void Image::changeContrast(const int &value) {
    RGBQUAD lAB = {0, 0, 0, 0};

    for (size_t i = 0; i < bmiHeader.biHeight; i++) {
        for (size_t j = 0; j < bmiHeader.biWidth; j++) {
            lAB.rgbRed += pixels[i][j].rgbRed;
            lAB.rgbGreen += pixels[i][j].rgbGreen;
```

```
20                  lAB.rgbBlue += pixels[i][j].rgbBlue * 0.114;
21              }
22          }
23
24          lAB.rgbRed /= bmiHeader.biHeight *
25              bmiHeader.biWidth * 0.299;
26          lAB.rgbGreen /= bmiHeader.biHeight *
27              bmiHeader.biWidth * 0.587;
28          lAB.rgbBlue /= bmiHeader.biHeight *
29              bmiHeader.biWidth * 0.114;
30          double k = 1.0 + value / 100.0;
31
32          for (size_t i = 0; i < bmiHeader.biHeight; i++) {
33              for (size_t j = 0; j < bmiHeader.biWidth; j++) {
34                  double deltaB = pixels[i][j].rgbBlue -
35                      lAB.rgbBlue;
36                  pixels[i][j].rgbBlue = overFlow(lAB.rgbBlue +
37                      k * deltaB);
38
39                  double deltaG = pixels[i][j].rgbGreen -
40                      lAB.rgbGreen;
41                  pixels[i][j].rgbGreen = overFlow(lAB.rgbGreen
42                      + k * deltaG);
43
44                  double deltaR = pixels[i][j].rgbRed -
45                      lAB.rgbRed;
46                  pixels[i][j].rgbRed = overFlow(lAB.rgbRed +
47                       * deltaR);
48              }
49          }
50      }
```

3.3.2 Фильтры

В листинге 2 представлены алгоритмы фильтров негатива, оттенки серого, размытия и уменьшения шума.

Листинг 2: Фильтры

```
// Фильтр негатива
void Image::negativeFilter() {
    for (size_t i = 0; i < bmiHeader.biHeight; i++) {
        for (size_t j = 0; j < bmiHeader.biWidth; j++) {
            pixels[i][j].rgbBlue = 255 -
                pixels[i][j].rgbBlue;
            pixels[i][j].rgbGreen = 255 -
                pixels[i][j].rgbGreen;
            pixels[i][j].rgbRed = 255 -
                pixels[i][j].rgbRed;
        }
    }
}

// Фильтр оттенков серого
void Image::grayFilter() {
    for (size_t i = 0; i < bmiHeader.biHeight; i++) {
        for (size_t j = 0; j < bmiHeader.biWidth; j++) {
            double s = (pixels[i][j].rgbBlue +
                        pixels[i][j].rgbGreen +
                        pixels[i][j].rgbRed) / 3;

            pixels[i][j].rgbBlue = s;
            pixels[i][j].rgbGreen = s;
            pixels[i][j].rgbRed = s;
        }
    }
}
```

```
29
30  // Фильтр размытия
31  void Image::blurFilter() {
32      double **coeffs = generate_coeff(7, 1.94);
33      double tmp_matrix[7][7];
34
35      for (size_t i = 3; i < bmiHeader.biHeight - 3; i++)
36      {
37          for (size_t j = 3; j < bmiHeader.biWidth - 3; j++)
38          {
39              for (size_t k = i - 3, i1 = 0; i1 < 7;
40                  k++, i1++)
41                  for (size_t l = j - 3, j1 = 0; j1 < 7;
42                      l++, j1++)
43                      tmp_matrix[i1][j1] =
44                          pixels[k][l].rgbBlue;
45
46              pixels[i][j].rgbBlue =
47                  overFlow(maskMatrix(tmp_matrix, coeffs));
48
49              for (size_t k = i - 3, i1 = 0; i1 < 7;
50                  k++, i1++)
51                  for (size_t l = j - 3, j1 = 0; j1 < 7;
52                      l++, j1++)
53                      tmp_matrix[i1][j1] =
54                          pixels[k][l].rgbGreen;
55
56              pixels[i][j].rgbGreen =
57                  overFlow(maskMatrix(tmp_matrix, coeffs));
58
59              for (size_t k = i - 3, i1 = 0; i1 < 7;
60                  k++, i1++)
61                  for (size_t l = j - 3, j1 = 0; j1 < 7;
62                      l++, j1++)
63                      tmp_matrix[i1][j1] =
```

```
64                          pixels[k][l].rgbRed;
65
66                  pixels[i][j].rgbRed =
67                      overFlow(maskMatrix(tmp_matrix, coeffs));
68              }
69          }
70      }
71
72      // Фильтр упонимежния шума
73      void Image::noiseFilter() {
74          for (size_t i = 1; i < bmiHeader.biHeight - 1; i++)
75          {
76              for (size_t j = 1; j < bmiHeader.biWidth - 1; j++)
77              {
78                  std::vector<PIX> tmp;
79
80                  for (size_t k = i - 1, i1 = 0; i1 < 3;
81                      k++, i1++)
82                      for (size_t l = j - 1, j1 = 0; j1 < 3;
83                          l++, j1++)
84                          tmp.push_back({
85                              0.3 * pixels[k][l].rgbRed +
86                              0.59 * pixels[k][l].rgbGreen +
87                              0.11 * pixels[k][l].rgbBlue,
88                              pixels[k][l]});
89
90                  std::sort(tmp.begin(), tmp.end(),
91                            [](const PIX &first,
92                               const PIX &second) {
93                                return first.value <
94                                second.value;
95                            });
96
97                  pixels[i][j] = tmp[4].rgb;
98              }
```

```
99          }
100     }
101
102     // Применение маски к матрице
103     double Image::maskMatrix(double A[7][7], double **B) {
104         double res = 0;
105
106         for (int i = 0; i < 7; i++)
107             for (int j = 0; j < 7; j++)
108                 res += A[i][j] * B[i][j];
109
110         return res;
111     }
112
113     double Image::overFlow(double midpix) {
114         if (midpix > 255) {
115             return 255;
116         } else if (midpix < 0) {
117             return 0;
118         } else {
119             return midpix;
120         }
121     }
```

3.4 Интерфейс программы

Общий интерфейс программы выглядит следующим образом (рис. 11).

Функции представленных кнопок следующие:

— кнопка «Загрузить изображение» – позволяет пользователю выбрать изображение с диск, которое в последствии будет обрабатываться;

— слайдеры «Яркость» и «Контрастность» – отвечают за одноименные параметры изображения;

— кнопка «Применить настройки» – считать значения со слайдеров «Яр-

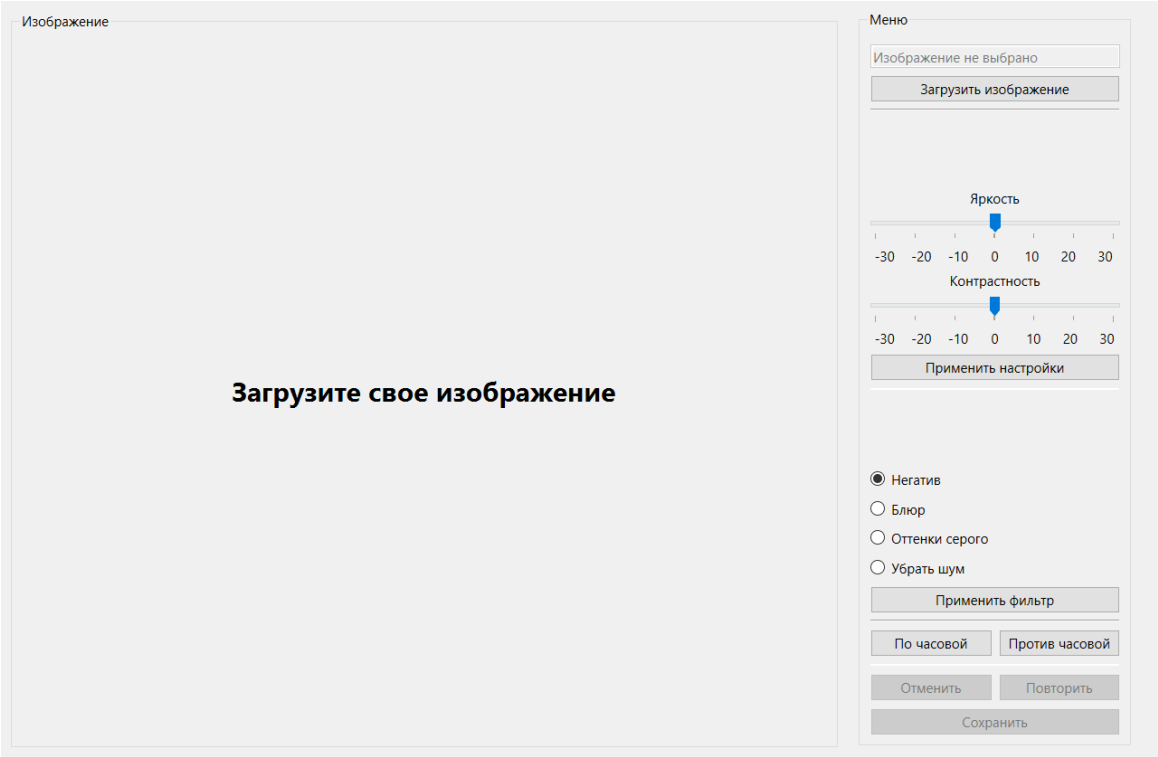

Рисунок 11 – Интерфейс программы

кость» и «Контрастность» и изменить параметры изображения на эти значения;

— радиокнопки «Нетагив», «Блюр», «Оттенки серого» и «Убрать шум» – отвечают за выбор фильтра;

— кнопка «Применить фильтр» – применяет выбранный фильтр к загруженному изображению;

— кнопки «По часовой» и «Против часовой» – вращают изображение по или против часовой стрелки соответственно;

— кнопки «Отменить» и «Повторить» – отвечают за историю преобразований, первая отменяет предыдущее действие, а вторая повторяет последнее отмененное действие;

— кнопка «Сохранить» – сохраняет результат обработки на диск.

3.5 Демонстрация работы программы

На рисунках 12, 13, 14 и 15 показаны результаты применения фильтров негатива, размытия, оттенков серого и уменьшения шума соответственно.

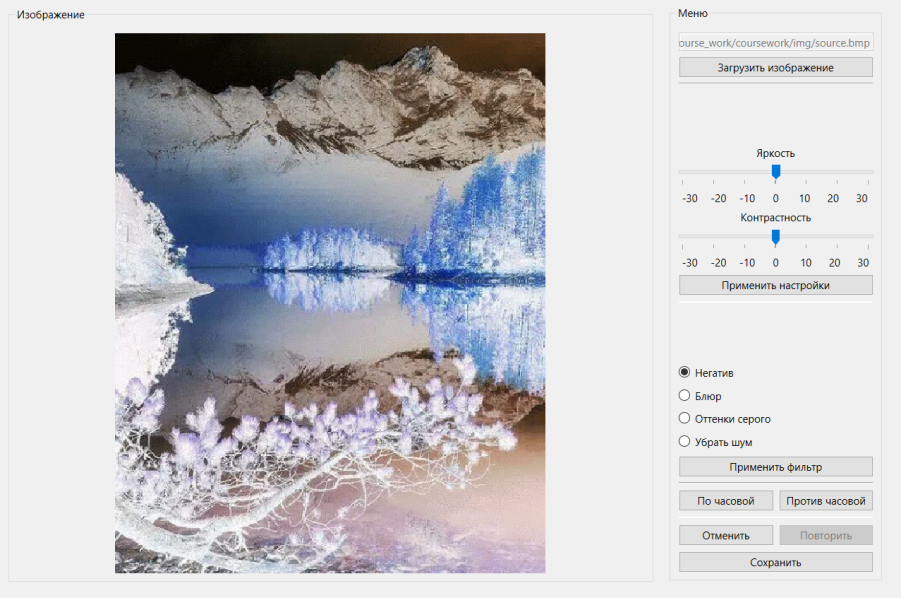

Рисунок 12 – Фильтр негатива

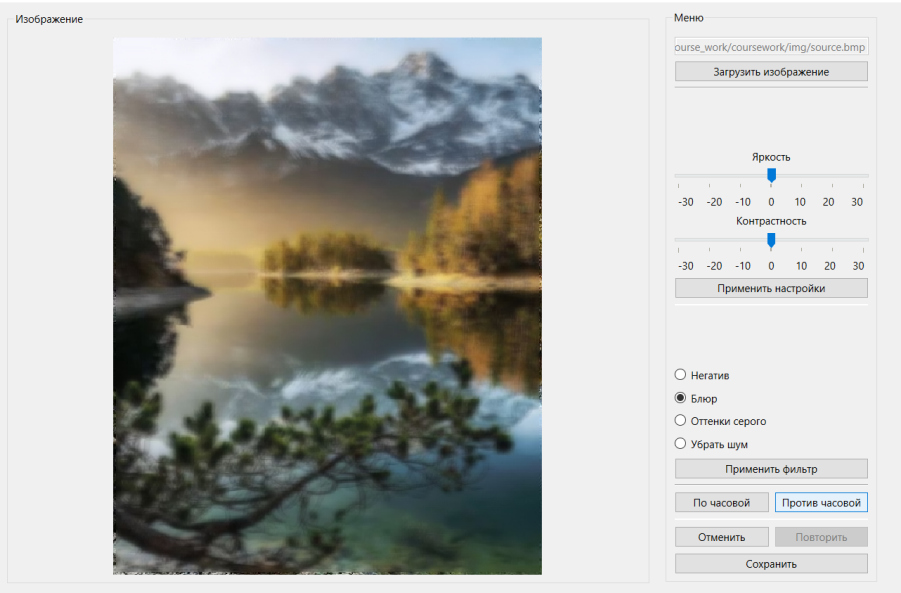

Рисунок 13 – Фильтр размытия

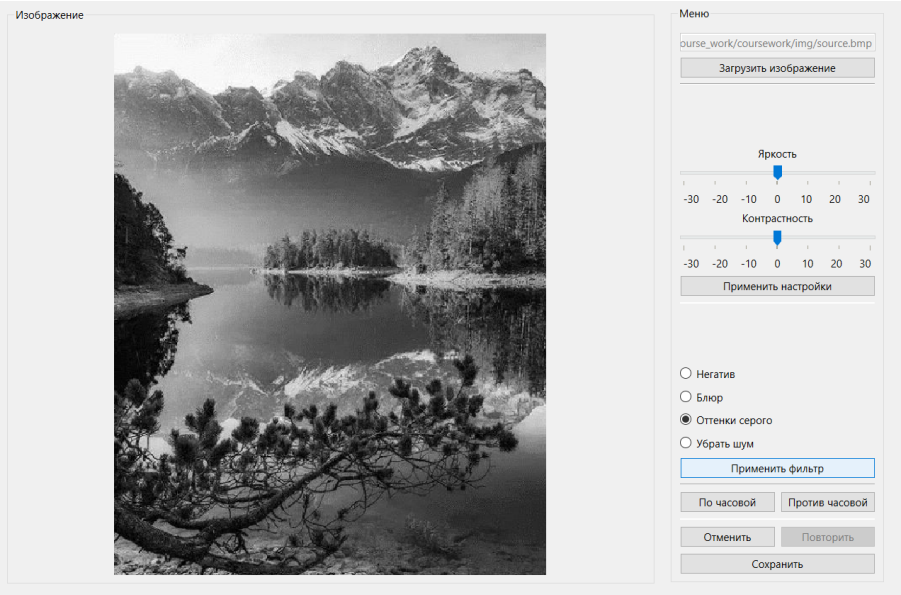

Рисунок 14 – Фильтр оттенков серого

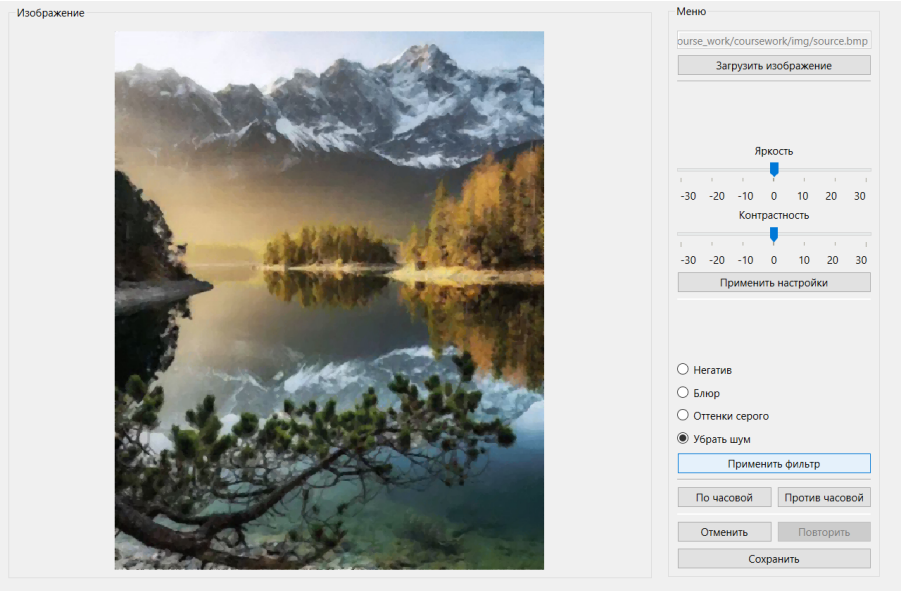

Рисунок 15 – Фильтр уменьшения шума

Далее, на рисунках 16 и 17 представлены результаты изменения яркости, контрастности и поворота изображения последовательно.

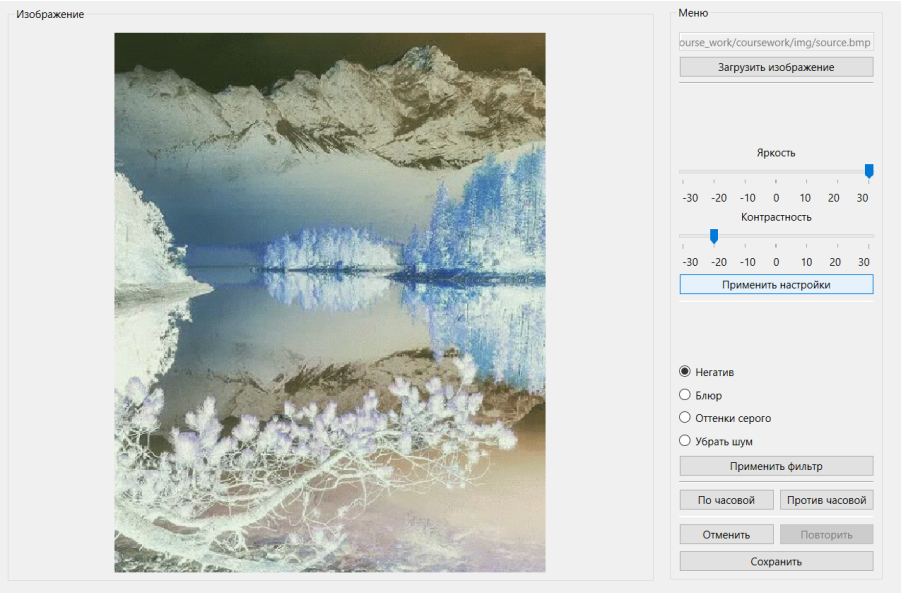

Рисунок 16 – Изменение яркости и контрастности

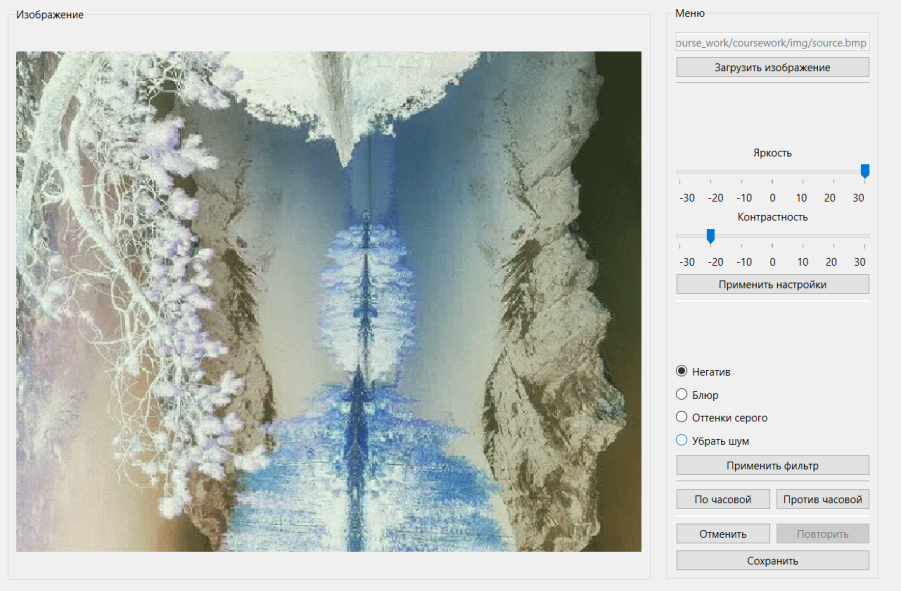

Рисунок 17 – Поворот изображения

Выводы

В данном разделе были рассмотрены детали реализации программного комплекса, представлены листинги кода алгоритмов обработки изображений и приведены примеры работы программы.

4 Исследовательская часть

В текущем разделе будет поставлена цель эксперимента, проведён сам эксперимент и сделаны соответствующие выводы.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, представлены далее:

— операционная система — Windows 10 [14];

— память — 8 ГБ;

— процессор — 8 x AMD Ryzen 5 3500U [15].

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Постановка эксперимента

4.2.1 Цель эксперимента

Цель эксперимента – выявить зависимость времени работы алгоритмов обработки изображений от разрешения изображений.

Время обработки изображения напрямую зависит от его разрешения, так как чем больше разрешение, тем больше будет матрица пикселей этого изображения.

4.2.2 Проведение эксперимента

Для проведения эксперимента были отобраны три изображения:

1) изображение с низким разрешением (240x164);
2) изображение со средним разрешением (512x640);
3) изображение с высоким разрешением (1620x2160).

У всех трех изображений глубина цвета равняется 24 бит.

В таблице 2 представлены результаты замеров времени (в мс) работы алгоритмов обработки изображений для трех типов, представленных выше.

Также, на рисунке 18 и 19 приведены графические результаты замеров

	240x164	512x640	1620x2160
Негатив	0.4879	4.569	43.3642
Оттенки серого	0.4591	4.5026	47.0616
Размытие	57.0516	510.937	5350.58
Снижение шума	133.406	1125.05	11741.2
Яркость	0.5155	4.4389	46.4873
Контрастность	1.312	11.13	124.974

Таблица 2 – Результаты замеров времени

работы алгоритмов в зависимости от разрешения исходного изображения.

Вывод

Исходя из полученных результатов, можно сделать вывод, что время работы любого алгоритма обработки изображений (из представленных в данной работе) линейно зависит от разрешения этого изображения. При увеличении разрешения в 10 раз – время работы алгоритмов также увеличивается в 10 раз.

Также, можно отметить, что алгоритмы, основанные на матрице свертки работаю дольше, чем точечные. Это обуславливается тем, что при применении точечного фильтра каждый пиксель обрабатывается ровно один раз, а при применении матрицы свертки – 9 раз.

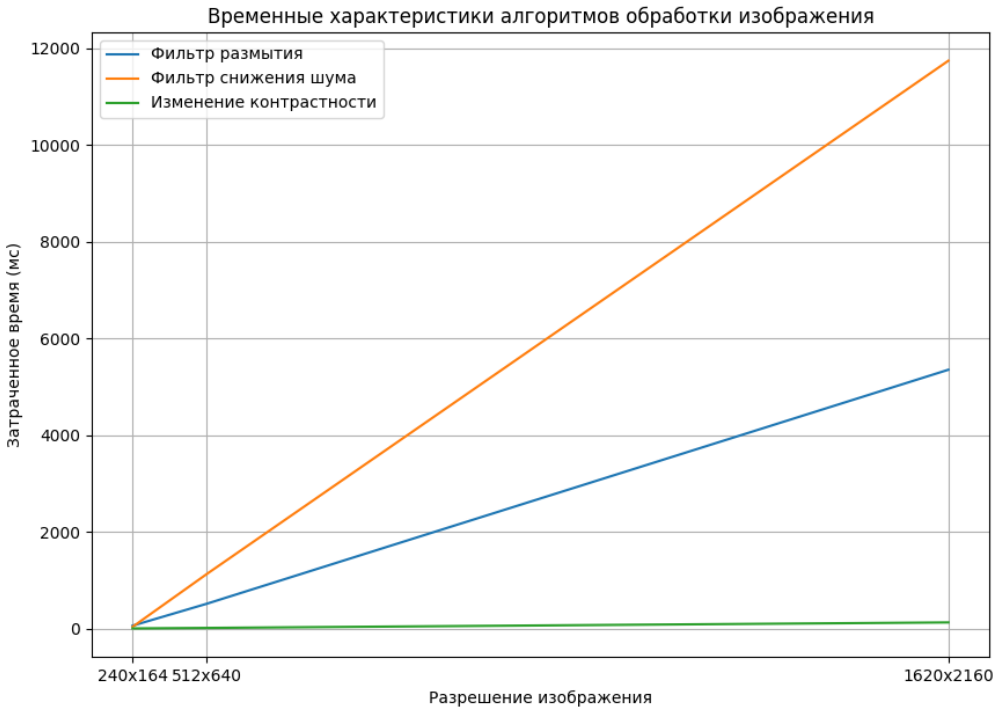

Рисунок 18 – Замеры времени – 1

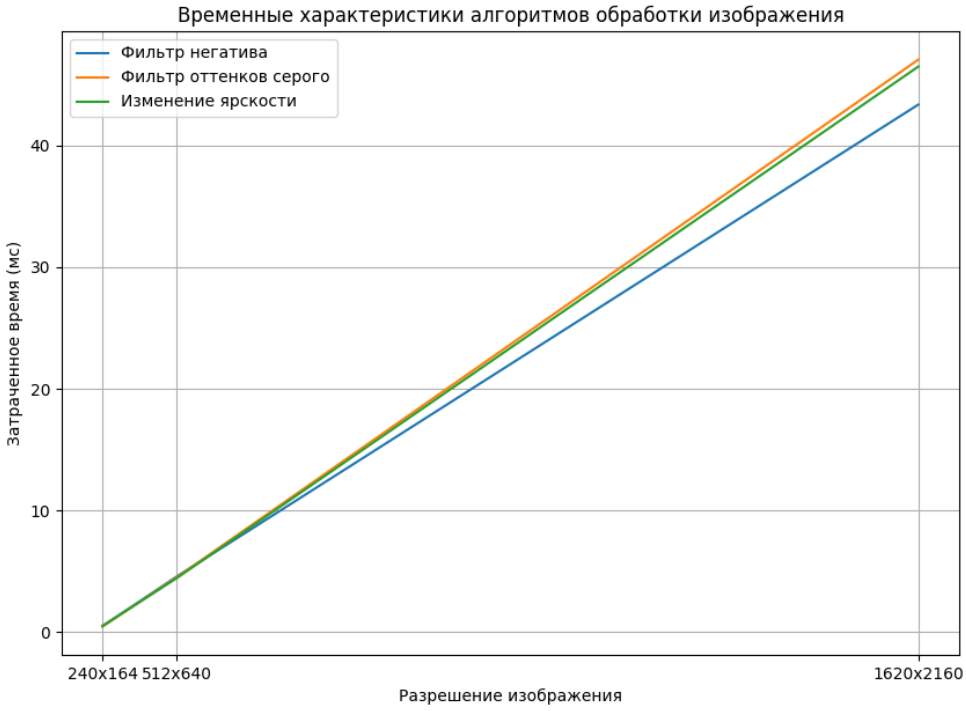

Рисунок 19 – Замеры времени – 2

ЗАКЛЮЧЕНИЕ

В ходе курсового проекта была реализована программа, способная помочь пользователю обработать выбранное им изображение, а также, наложить на него один или несколько известных фильтров.

В ходе выполнения данной работы были выполнены следующие задачи:

— выбран формат изображения, над которым будут проводиться операции для редактирования изображения;
— определены методы фильтрации изображения, рассмотрены их основные характеристики;
— проведено знакомство с известными фильтрами обработки изображения;
— полученные знания применены на практике.

В ходе выполнения эксперимента была установлена линейная зависимость времени выполнения алгоритмов обработки изображений от его разрешения. Были определены алгоритмы, которые требуют больших временных затрат.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Desktop Screen Resolution Stats Worldwide [Электронный ресурс]. – Режим доступа: https://gs.statcounter.com/screen-resolution-stats/desktop/worldwide, свободный – (08.12.2022)

2. ”Компьютерная графика : учеб. пособие”. С.Г. Шульдова, 2019. с. 299

3. ”Biomedical imaging and intervention journal volume 2, ”Image file formats”. Tan, Lip, 2006. с. 1–8

4. ”High Technology Letters volume 27, ”An Analysis of Different Image Formats for Steganography”. V. Krishna, 2021. с. 165–177

5. ”Understanding Digital Image Processing”. V. Tyagi, 2018. с. 365

6. ”Digital Image Processing A Complete Guide”. R. Gonzalez, R. Woods, 2020. с. 244

7. ”Image Filtering - Techniques, Algorithm and Applications”. B. Desai, U. Kushwaha, 2020. с. 971–975

8. ”Nonlinear Image Processing”. Sanjit K. Mitra and Giovanni L. Sicuranza, 2001. с. 203–223

9. ”Image convolution: a linear programming approach for filters design”. G. Capobianco, C. Cerrone, 2021. с. 8941–8956

10. ”The SAGE Encyclopedia of EducationalResearch, Measurement, and Evaluation – Normal Distribution”. J. Dorothy, Musselwhite and Brian C. Wesolowski, 2018. с. 5

11. ”Gaussian Random Functions volume 32, ”Mathematics and Its Applications”. M. A. Lifshits, 1995. с. 281

12. Программирование на C/C++ [Электронный ресурс]. – Режим доступа: http://www.c-cpp.ru/, свободный – (08.12.2022)

13. Qt framework [Электронный ресурс]. – Режим доступа https://doc.qt.io/, свободный – (08.12.2022)

14. Windows [Электронный ресурс]. – Режим доступа: https://www.microsoft.com/ru-ru/windows, свободный – (08.12.2022)

15. Процессор AMD Ryzen™ 5 3500U с графикой Radeon™ Vega 8 [Электронный ресурс]. – Режим доступа: https://www.amd.com/ru/products/apu/amd-ryzen-5-3500u, свободный – (08.12.2022)