МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий

Кафедра технологий программирования

**ЛАБОРАТОРНАЯ РАБОТА**

по дисциплине: **«Цифровая обработка изображений»**

на тему: «Методы обработки изображений для решения практических задач»

ВЫПОЛНИЛ студент группы: 15-ИТ-1

Стеняев А.Д.

ПРОВЕРИЛ зав. кафедры ВСиС

Богуш Р.П.

Полоцк 2019 г.

**Цель работы:** реализация методов обработки изображения для решения практических

задач..

**Ход выполнения лабораторной работы:**

**1 Реализация «каркаса» программы.**

Перед началом работы над реализацией алгоритмов обработки изображений необходимо подготовить основу приложения. Основная задача программы – показать пользователю начальный и итоговый результат. Отталкиваясь от этого было принято решение в качестве основы написать простой графический интерфейс, в центре которого располагается исходное изображение. К этому изображению в дальнейшем будут применяться все алгоритмы обработки изображений. С помощью меню и диалоговых окон будет происходить взаимодействие пользователя с программой.

Для реализации программы был выбран язык программирования C++ в совокупности с фреймворком Qt. С помощью Qt будут реализованы все графические элементы управления, а также возможность загружать и отображать изображение с возможностью получать доступ к данным этих изображений. Стоит отметить некоторые ограничений, которые предоставляет фреймворк при работе с изображениями:

- цветные изображения всегда имеют 4 компоненты;

- чёрно-белые изображения, использующие одну компоненту, имеют ширину байт в строке кратную четырём, что создаёт некоторые неудобства при обработке изображения.

Реализация программы начинается с создания проекта, управление которым берёт на себя QMAKE. Основу приложения составляет класс QMainWindow, который предоставляет удобный интерфейс для управления окном, а также для добавления меню. Центр окна занимает объект QLabel, который будет выводить изображение. Некоторые изображения могут не вмещаться в видимую часть окна, поэтому QLabel дополнительно оборачивается в QScrollArea, который добавляет полосы прокрутку и инкапсулирует процесс управления ими. В завершении необходимо добавить меню в верхнюю часть окна и добавить в него «действий». Результат построения базовой части приложения представлен на рисунке 1.1.

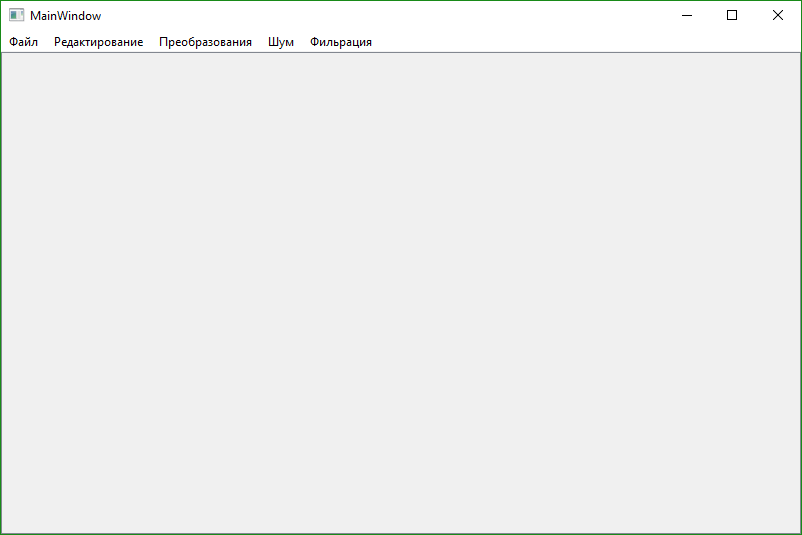


Рисунок 1.1 – внешний вид главного окна приложения

Следующий шаг – возможность загружать изображение. Для этого в меню был добавлен пункт «Открыть», который вызывает системный диалог выбора файлов. После выбора файла изображение загружается и помещается на лэйбл, приготовленный ранее. Результат загрузки изображения представлен на рисунке 1.2.

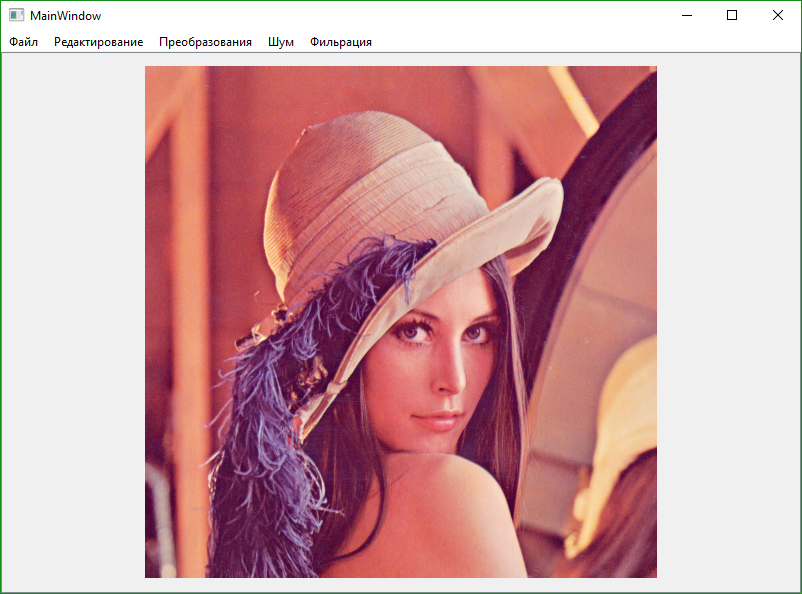


Рисунок 1.2 – результат загрузки изображения

Для удобства дополнительно была реализована структура, которая хранит в себе сразу два изображения: исходное и обработанное. Она нужна для реализации возможности получать результат выполнения операции в отдельном изображении, тем самым имея возможность внести некоторые изменения в алгоритм обработки изображения. Если результат устраивает – изображения просто меняются местами. Так же эта структура имеет метод, который выполняет обработку изображения. Для удобства был выбран вариант, при котором каждый алгоритм представляет из себя функцию с определённым интерфейсом – функторами. Структура, принимающая функтор будет активировать его, передавая внутрь исходное изображение и ожидая получить результирующие изображение. В C++ функторами могут быть как обычные функции, так и объекты классов с перегруженным оператором «скобочки», в следствии чего они могут хранить какое-то состояние, необходимое им для работы.

Диалоги – важная часть при работе с пользователем. Каждый алгоритм имеет свои настройки, поэтому, для получения желаемого результата, пользователю необходимо предоставить графический интерфейс, содержащий все элементы управления для подготовки алгоритма к обработке изображения. Не маловажным является возможность быстро увидеть приблизительный результат обработки изображения. Для реализован класс диалога, который инкапсулирует в себе процессы установки базовых параметров шаблонного диалога, а также предоставляет возможность адаптивно показывать пользователю результат обработки изображения. Адаптивность заключается в том, что изображения, у которых ширина больше высоты будут отображаться «над» управляющими элементами, а в противном случае – слева от них. Макеты диалогов представлены на рисунке 1.3.

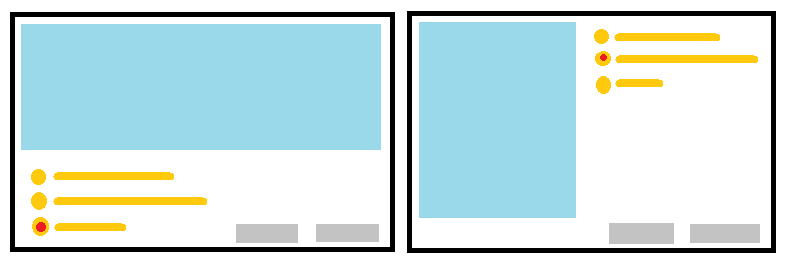


Рисунок 1.3 – макет адаптивного интерфейса диалога настроек алгоритма обработки

**2. Реализация базовых алгоритмов**

**2.1. Алгоритм масштабирования изображения**

Первый алгоритм, который должен быть реализован – алгоритм масштабирования изображения. Его основная задача в программе – уменьшать изображения для предоставления миниатюр в диалогах настройки алгоритмов обработки этих изображений. Для реализации базового и самого быстрого масштабирования был выбран «метод ближайшего соседа». Данный алгоритм является очень простым и быстрым, однако предоставляет низкое по качеству результирующее изображение, так как каждый пикселы результирующего изображения являются сырыми пикселами исходного изображения. Алгоритм работает следующим образом: на место пиксела результирующего изображения выбирается пиксел из исходного изображения с помощью масштабирования координат пиксела результирующего изображения в координаты исходного изображения. Схематический пример приведён на рисунке 2.1.1.

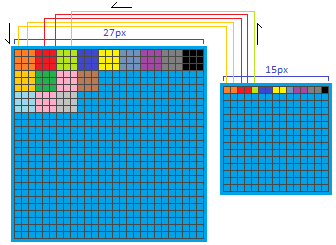


Рисунок 2.1.1 – Схематичное представление выборки пикселя в результирующее изображение

На схеме представлено масштабирование из большей картинки в меньшую, хотя это работает и в обратную сторону. Математически связь координат можно описать двумя формулами:

и , где

Xi – Координата Х в исходном изображении,

Xo – координата X в результирующем изображении,

Yi – координата Y в исходном изображении,

Yo – координата Y в результирующем изображение.

Для демонстрации результата приведём пример диалогового окна для выбора алгоритма получения чёрно-белого изображения, о котором будет рассказано дальше. Пример диалогового окна с уменьшенным изображением на фоне оригинала представлен на рисунке 2.1.2.

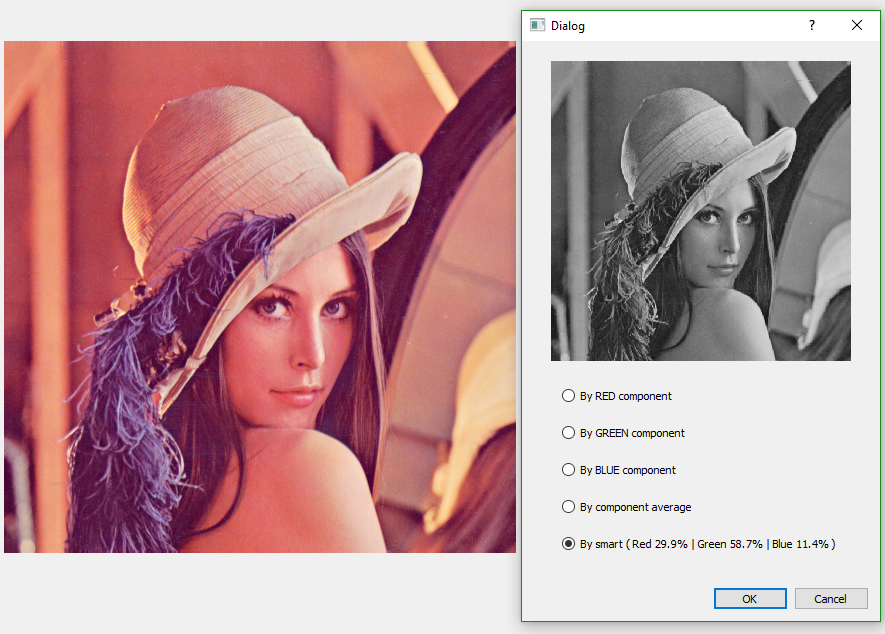


Рисунок 2.1.2 – результат масштабирования исходного изображения

На рисунке 2.1.2 виден основной побочный эффект такого масштабирования – ступенчатость в местах, где в оригинале был плавный переход.

Исходный код преобразования представлен в листинге 2.1.1

Листинг 2.1.1 – масштабирование цветного изображения

for (int y = 0; y < newHeight; ++y) {

for (int x = 0; x < newWidth; ++x) {

int newX = int( double( x ) / \_scaleWidth );

int newY = int( double( y ) / \_scaleHeight );

int offsetInput = newY \* width \* 4 + newX \* 4;

int offsetOutput = y \* newWidth \* 4 + x \* 4;

outputImageBits[ offsetOutput++ ] = inputImageBits[ offsetInput++ ];

outputImageBits[ offsetOutput++ ] = inputImageBits[ offsetInput++ ];

outputImageBits[ offsetOutput++ ] = inputImageBits[ offsetInput++ ];

outputImageBits[ offsetOutput ] = inputImageBits[ offsetInput ];

}

}

**2.2. Преобразование в полутоновое изображение**

Процесс преобразования цветного изображения в чёрно-белое является простой задачей, однако её роль весьма высока, так как полутоновые изображения используются в других алгоритмах, например, в бинаризации изображения (об этом далее). Есть много способов получить оттенок серого из цветного пиксела, например, по одной из компонент цвета, по всем трём в равной степени или же в определённом соотношении. Наиболее достоверным и правильным является выделение яркости путём умножения каждой компоненты цвета на определённый множитель, выведенный экспериментальным путём. В данной программе будут использованы следующие пропорции цветов: 29.9% для красной компоненты, 58.7% для зелёной компоненты и 11.4% для синей компоненты.

В программе реализованы разные методы получение полутонового изображения, а результаты преобразований представлены на рисунке 2.2.1.



Рисунок 2.2.1 – результаты полутонового преобразования пятью способами.

**2.3. Бинаризация изображения**

Бинаризованое изображение – частный случай чёрно-белого изображения, в котором каждый пиксел имеет одно из допустимых значений – минимальная(чёрный) или максимальная(белый) яркость. Бинарные изображения часто используются для выделения каких-то объектов на фоне других для упрощения последующего анализа изображения. Простой пример: на листке бумаги лежат предметы, положение которых надо определить. Алгоритмов бинаризации огромное множество и каждый подходит для решения определённых задач. В рамках программы реализовано два алгоритма: бинаризация с верхним/нижним порогом и с применением адаптивного алгоритма Оцу.

Бинаризация с верхним/нижним порогом – самый простой вид бинаризации, так как порог в нём задаётся вручную пользователем (в программе этот порог равен 128). Метод Оцу – алгоритм нахождения порога бинаризации путём разделения набора пикселов на два класса: полезные и фоновые. Порог рассчитывается таким образом, чтобы внутриклассовая дисперсия была минимальной.

Алгоритм метода Оцу:

Шаг 1: вычислить гистограмму.

Шаг 2: ………………………….

Результаты применения алгоритмов бинаризации представлен на рисунке 2.3.1.

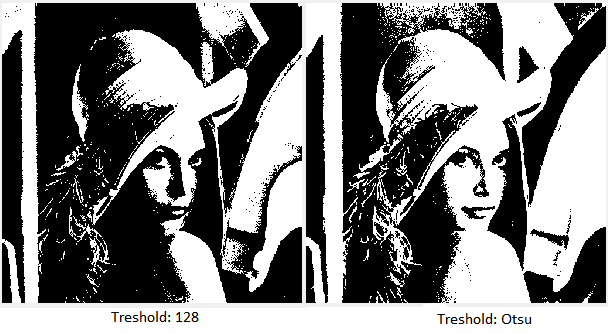


Рисунок 2.3.1 – результаты применения алгоритмов бинаризации

**2.4. Увеличение яркости изображения**

Увеличение яркости изображения – это процесс увеличение яркости каждого пиксела этого изображения. Яркость пиксела – интенсивность свечения этого пиксела. Для увеличения яркости в программе был реализован простой алгоритм, который равномерно уменьшает яркость всех пикселов. Результат Увеличения яркости изображения приведён на рисунке 2.4.1.

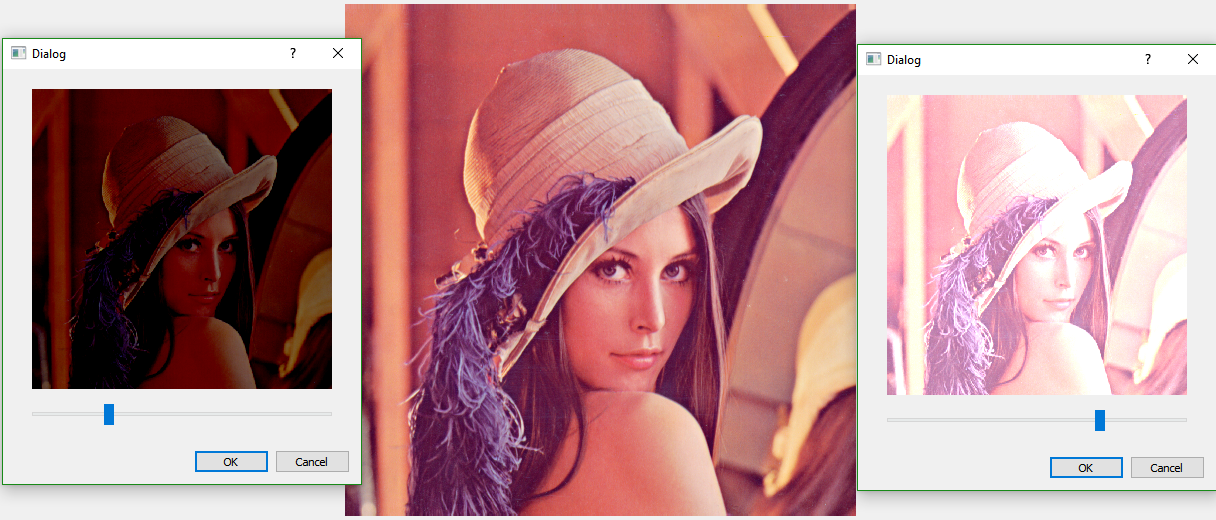


Рисунок 2.4.1 – результат увеличение и уменьшение яркости изображения

2.5. Шум «Соль и перец»

Шум «Соль и перец»

2.6. Гауссов шум

Гауссов шум

3. Фильтрация изображений методом свёртки

3.1. Низкочастотная фильтрация

Низкочастотные фильтры

3.2. Высокочастотная фильтрация

Высокочастотные фильтры

3.3. Медианная фильтрация

3.4. Тиснение

3.5. Выделение контуров

3.6. Линейное контрастирование

3.7. Нелинейное контрастирование

Вывод