Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра Информатики
Дисциплина: Программирование (П)

Пояснительная записка к курсовой работе
Тема работы:
«Приложение для обработки растровых изображений»

Исполнитель
студентка гр. 653501 _____ Ковалевская А.А.
(подпись дата)

Руководитель _____ Козуб В.Н.
(подпись дата)

_____
(оценка)

Минск 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время ни одна область человеческой деятельности не может обойтись без применения цифровых технологий для создания и обработки изображений. Растровый принцип представления идеально подходит для изображений с плавными переходами между цветами и яркостями, такими как большинство устройств ввода-вывода графической информации, таких как мониторы (за исключением векторных устройств вывода), матричные и струйные принтеры, цифровые фотоаппараты, сканеры, а также сотовые телефоны.

Растровая графика позволяет создать практически любой рисунок, вне зависимости от сложности, но качество изображения зависит от того, с каким разрешением (количеством пикселов на дюйм) создавался файл, и от того, где он будет печататься или просматриваться. Необходимость определенным образом корректировать изображение возникает по самым различным причинам, начиная от предобработки данных дистанционного зондирования Земли и заканчивая продвижением себя в Instagram. Так что еще один растровый редактор никому не помешает.

Растровое (пиксельное) изображение – это набор числовых значений, определяющих цвета, которые должны представлять пикселы в выводном устройстве. Растровое изображение представляет собой фиксированную решетку пикселов, каждый из которых имеет соответствующее значение яркости или координату цвета.

Для обработки растровых изображений применяются специализированные программы – растровые графические редакторы, позволяющие изменять их посредством специальной обработки – фильтров, эффектов, наложения текстур, теней, подсветки и пр.

Самым мощным по функциональности графическим редактором является Adobe Photoshop. Он позволяет изменять расположение и яркость пикселов для создания специальных эффектов, процедуры тоно- и цветокоррекции и т.д.

Целью данной работы является создание приложения, обладающего некоторыми функциональными возможностями для обработки растровой графики.

Для достижения поставленной цели необходимо решить следующие задачи:

– изучить базовые алгоритмы обработки растровых изображений;

– определить логическую схему и разработать технические аспекты реализации программного продукта;

– создать функциональные и обслуживающие компоненты приложения;

– протестировать работу приложения.

1 ТЕХНИЧЕСКИЕ АСПЕКТЫ РАЗРАБОТКИ ПРИЛОЖЕНИЯ

1.1 Логика и структура программы

Профессиональные программы обработки растровой графики предоставляют специалисту много возможностей по созданию и обработке цифровых изображений.

Для разработки приложения была выбрана среда ***Microsoft Visual Studio 2015***, платформа ***.NET Framework 4.6***, язык ***C# 6.0.*** Данный выбор объясняется тем, что, во-первых, он относится к семье языков с С-подобным синтаксисом, так что после изучения С и С++ писать на нем достаточно привычно. C# обладает достаточно мощным инструментарием объектно-ориентированного программирования: имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, события, свойства, обобщённые типы и методы, итераторы, анонимные методы, исключения и др.

Программа ***ImageEditor*** создавалась как приложение ***Windows Forms***. Использовались элементы управления пользовательского интерфейса *Windows Forms* из библиотеки классов *.NET Framework* (классы *Form, Button, ToolBar, TrackBar, PictureBox* и др.).

Для работы с растровыми изображениями использовался класс ***Bitmap***, определенный в пространстве имен ***System.Drawing***. Битовая карта состоит из данных пикселей графического изображения и его атрибутов, может быть создана из файлов следующих форматов: BMP, GIF, EXIF, JPG, PNG и TIFF. Класс представляет изображение в модели RGBA (пиксель имеет размер 4 байта, последний – альфа – отвечает за канал прозрачности, как в файлах форматов *.bmp и *.png).

В общем случае обработка растровой графики сводится к изменению цвета, тона, контраста, насыщенности точек всего изображения целиком или его отдельной области.

Класс ***Bitmap*** содержит методы доступа к пикселю и изменения значений пикселя, но они очень медленные, т.к. для каждого обращения пиксель нужно заблокировать, изменить и разблокировать. Изображения часто содержат миллионы пикселей, поэтому их обработка – очень ресурсо- и времязатратная процедура. Поэтому поиск путей повышения производительности выполняемых процессов является особенно актуальным. В нашем случае обоснованным становится использование неуправляемых указателей. Скорость работы при этом увеличивается в десятки раз.

Чтобы работать с изображением как с массивом байт (а каждый пиксель представляет собой 4 байта в модели RGBA), используются методы класса ***Bitmap: LockBits***, блокирующий заданную область объекта ***Bitmap*** в системной памяти, и ***UnlockBits***, разблокирующий эту область. Для хранения данных об объекте используется класс ***BitmapData*** из пространства имен ***System.Drawing.Imaging***. Он содержит метод ***Scan0***, который возвращает

указатель на первый байт изображения. Нужно отметить, что байты пикселя идут в порядке синий-зеленый-красный-альфа.

Совокупность данных компонентов позволила разработать программный продукт ***ImageEditor***. Основной функционал данного продукта включает в себя такие функции, как изменение баланса яркости и контрастности, изменение цветовой насыщенности, наложение фильтров. Созданные в курсовой работе режимы являются лишь минимальным набором специфических приемов редактирования, которыми пользуется дизайнер-график.

1.2 Функциональные компоненты приложения

Для проведения комплекса процедур по обработке растрового изображения были использованы различные алгоритмы фильтрации, изменения баланса изображения и его цветности, организованные в следующие структурные элементы: классы, делегаты, структуры, методы, события (рисунок 1.1).

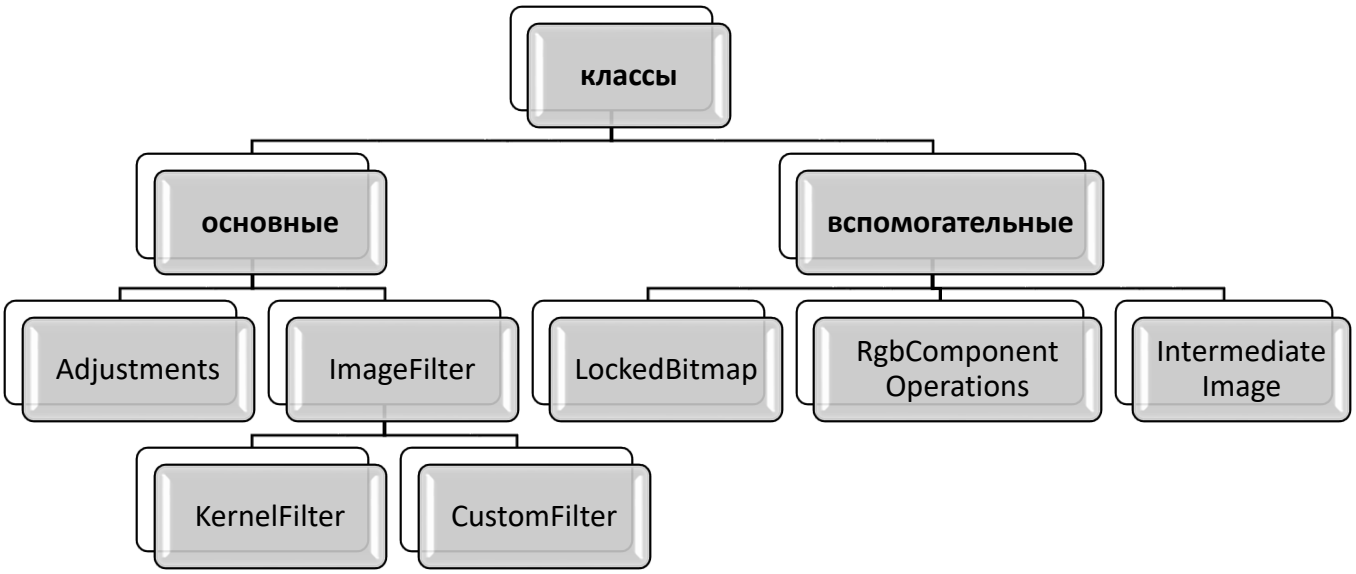

Рисунок 1.1 – Функциональные компоненты приложения ***ImageEditor***

1. Класс ***Adjustments*** объединяет функции, предназначенные для работы с цветностью изображений.

Для обобщения операций с цветовыми составляющими пикселя вводится тип делегата ***ApplyToPixel*** (см. приложение). Объекты данного типа могут ссылаться на методы, принимающие в качестве аргументов неуправляемый указатель на отдельный байт изображения (подразумевается, что первый в пикселе, то есть синий) и вещественное число – коэффициент интенсивности коррекции. Предполагается, что делегируемые методы изменяют значения RGBA-компонентов пикселя по определенному алгоритму.

Внутри класса определено несколько закрытых подходящих по сигнатуре к ***ApplyToPixel*** методов, реализующих следующие операции с пикселями: коррекция яркости, контраста, баланс красного, синего и зеленого каналов, перевод в полутоновый и пороговый режимы, гамма-коррекция, экспозиция, сепия, инверсия цвета.

Для непосредственного побайтового редактирования изображения используется метод ***Adjust***. Он осуществляет проход по всем байтам изображения, которое предварительно блокируется в системной памяти. Имеется две перегрузки этого метода (обе закрытые).

Первая версия в качестве аргументов принимает исходное изображение, коэффициент интенсивности коррекции и объект делегата ***ApplyToPixel***. К каждому пикселю применяются делегируемые методы, причем во все передается один и тот же коэффициент.

Но для некоторых преобразований изображения было бы удобно к каждому пикселю применить набор методов и передать им разные коэффициенты. Чтобы объединить метод и коэффициент, внутри класса создана вспомогательная структура ***Adjustment.***

Таким образом, вторая версия метода ***Adjust*** принимает исходное изображение и список объектов ***Adjustment***. К каждому пикселю применяется метод, на который указывает поле-делегат элемента списка, со своим коэффициентом.

Методы, доступные пользователям класса, организуют корректную работу ключевого метода ***Adjust***. Они принимают изображение, которое нужно обработать, и, если нужно, коэффициенты интенсивности коррекции. Внутри этих методов создается и настраивается объект-делегат ***ApplyToPixel*** или список экземпляров ***Adjustment,*** которые передаются в ***Adjust***.

2. Класс ***ImageFilter*** – абстрактный класс, реализующий базовые методы работы «квадратного» фильтра. Форма фильтра влияет на то, каким образом выбирается окрестность изменяемого пикселя. Это может быть квадрат, прямоугольник, стрелка, линия и др. Так как фильтр квадратный, для определения размера окрестности достаточно знать длину стороны квадрата (в пикселях).

Для формирования окрестности составляющей пикселя используется закрытый метод ***GetRgbComponentNeighborhood***. Он принимает неуправляемый указатель на байт изображения и создает одномерный массив соседних (т.е. входящих в квадратную окрестность) значений RGBA-компонентов того же цветового канала.

Класс содержит один абстрактный метод – ***GetNewRgbComponentValue***. Он принимает одномерный массив-окрестность и возвращает целое число в диапазоне от 0 до 255. Классы-наследники должны реализовать свой способ вычисления значения RGBA-компонента.

В закрытый метод ***Apply*** должны быть переданы входное изображение и выходное изображение (пустое такого же размера). Он создаст промежуточное

изображение – объект класса ***IntermediateImage***. После этого выходное и промежуточное изображение блокируются в системной памяти, и осуществляется их побайтовый обход. При этом каждому байту выходного изображения присваивается значение, вычисленное абстрактным методом ***GetNewRgbComponentValue*** по окрестности соответствующего байта промежуточного изображения.

3. Класс ***KernelFilter*** наследуется от абстрактного класса ***ImageFilter***. Он предназначен для фильтров, работающих с использованием квадратной матрицы (ядра) свертки.

Вводятся еще два параметра: одномерный массив, являющийся на самом деле трансформированной двумерной матрицей свертки, и коэффициент фильтра. Чтобы избежать ошибок, класс имеет единственный конструктор, принимающий в качестве аргумента двумерную матрицу и вещественное число. По размеру этой матрицы определяется размер фильтра и используемой окрестности.

Чтобы вычислить новое значение цветовой составляющей (отклик фильтра), нужно поэлементно умножить элементы матрицы свертки (весовые множители), просуммировать все произведения и умножить полученное число на коэффициент фильтра. Эту операцию и реализует метод ***GetNewRgbComponentValue*** в ***KernelFilter***, переопределяя абстрактный метод класса-предка.

Пользователь класса должен вызвать метод ***Apply*** и передать исходное изображение. Внутри этого метода будет создано новое пустое изображение – впоследствии выходное – такого же размера, байтам которого будут присвоены новые значения.

4. Класс ***CustomFilter*** также наследуется от абстрактного класса ***ImageFilter***. Он предназначен для работы с фильтрами, зависящими только от размера окрестности и значений входящих в нее пикселей.

Способов задать новое значение цвета существует очень много, поэтому в классе вводится тип делегата ***ComputeRgbComponentValue,*** сигнатура которого совпадает с сигнатурой нуждающегося в переопределении абстрактного метода класса-предка ***GetNewRgbComponentValue***. Чтобы использовать этот делегат внутри этой реализации, у класса имеется поле типа ***ComputeRgbComponentValue***. Значение этому полю присваивается в открытых методах, а в теле ***GetNewRgbComponentValue*** вызываются делегируемые им методы. Таким образом пользователь класса может применить к изображению фильтры эрозия, размытие и др.

5. Класс ***LockedBitmap*** используется для блокирования и разблокирования изображения (объекта ***Bitmap***) в системной памяти на основе методов ***LockBits*** и ***UnlockBits*** класса ***Bitmap***.

6. Класс ***RgbComponentOperations*** содержит статические методы с формулами, по которым вычисляется значение RGBA компонента. Кроме того, в данном классе имеется статический метод ***ControlRgbComponentOverflow***, который проверяет, входит ли значение в диапазон от 0 до 255, и если нет, то урезает его. Результат можно безопасно присвоить RGBA-компоненту.

7. Класс ***IntermediateImage*** формирует промежуточное изображение, к которому в дальнейшем будет применятся квадратный фильтр. Оно имеет ширину и высоту, большие, чем у входного, на длину стороны фильтра. В центр промежуточного изображения копируется исходное изображение, а края заполняются крайними пикселями входного изображения. Таким образом, у бывших крайних пикселей входного изображения тоже появляется окрестность, к которой можно применить матрицу свертки.

1.3 Обслуживающие компоненты приложения

Класс ***MainForm*** содержит закрытые поля ***original*** – исходное изображение, ***workingCopy*** – рабочая копия изображения, с которой происходят все изменения, и ***backup*** – резервная копия рабочей копии, представляющие собой объекты класса ***Bitmap***. Они инициализируются при открытии графического файла. ***MainForm*** также содержит поля типов вспомогательных форм. Они инициализируются в конструкторе главной формы.

Для координации обслуживающих элементов приложения использовался механизм событий. Для упрощения этих действий в программе определен тип делегат ***ImageProcessingEventHandler***. В качестве второго аргумента он принимает объект класса ***ImageProcessingEventArgs***, который хранит ссылку на изображение.

В ***MainForm*** определено событие: ***AdjustmentCall*** типа ***ImageProcessingEventHandler***. Необходимые обработчики в других классах подписываются на эти события при своем создании в конструкторе главной формы.

AdjustmentCall генерируется с помощью метода ***OnAdjustmentCall*** при нажатии на любую из кнопок в разделе ***Adjustments*** панели меню. Одновременно с этим на экране появляется необходимая вспомогательная форма, которая содержит необходимые элементы управления, регулирующие интенсивность цветокоррекции изображения. Чтобы пользователь видел изменения, которые вызывает коррекция, необходимо в каждый момент времени передавать изображение в рабочую область главной формы.

Класс формы содержит метод-обработчик события ***AdjustmentCall***, куда передается объект ***ImageProcessingEventArgs,*** проинициализированный ссылкой на рабочую копию изображения.

Каждая вспомогательная форма имеет три события типа ***ImageProcessingEventHandler***: ***ProcessingCompleted, ProcessingApproved*** и ***ProcessingCanceled.*** В свою очередь, главная форма предоставляет обработчиков этих событий.

Обработчиком события ***ProcessingCompleted*** в ***MainForm*** является функция ***ViewProcessedImage***, которая в качестве аргументов принимает ***ImageProcessingEventArgs***, который в этом случае хранит ссылку обработанную копию изображения, и отображает его в ***pictureBox***.

При нажатии на кнопки ***OK*** и ***Cancel*** вспомогательной формы генерируются события ***ProcessingApproved*** и ***ProcessingCanceled*** соответственно. В главной форме их обработкой занимаются методы ***ApproveProcessing*** (изменяет ***workingCopy*** на обработанное изображение) и ***CancelProcessing*** (ничего не изменяет, отображает исходную ***workingCopy*** в ***pictureBox***).

Обработчики событий вспомогательных форм подписываются на них при создании этих форм в конструкторе главной формы.

Чтобы пользователь мог отменить одно последнее действие, в главной форме имеется два метода: ***BackUpWorkingCopy*** (копирует рабочую копию в резервную) и ***RestoreBackupWorkingCopy*** (перекопирует резервную копию в рабочую). Первый вызывается перед вызовом любого метода, который может изменить изображение, а второй – при нажатии на кнопку ***Undo***.

2 ОПИСАНИЕ РАБОТЫ ПРИЛОЖЕНИЯ

2.1 Интерфейс и дизайн приложения

В процессе разработки ***ImageEditor*** был реализован пользовательский интерфейс, позволяющий выполнять все предусмотренные программным модулем операции быстро, легко и интуитивно понятно.

Пользовательский интерфейс представляет собой область для обработки изображений, меню открытия файла, меню выбора фильтров а также различные инструменты управления фильтрами: слайдеры и поля для ввода значений (рисунок 2.1).

Рисунок 2.1 – Пользовательский интерфейс приложения ***ImageEditor***

При запуске приложения на экране появляется главное окно. В верхней части размещена панель меню, в центре – рабочая область для обработки изображений. До открытия файла доступны только кнопки ***Open, Exit*** раздела меню ***File***. При нажатии на кнопку ***Open*** появляется диалоговое окно выбора файла (разрешения *.jpg, *.bmp, *.png). После выбора картинка отобразится в рабочей области, а все кнопки и разделы меню станут доступны (рисунок 2.2).

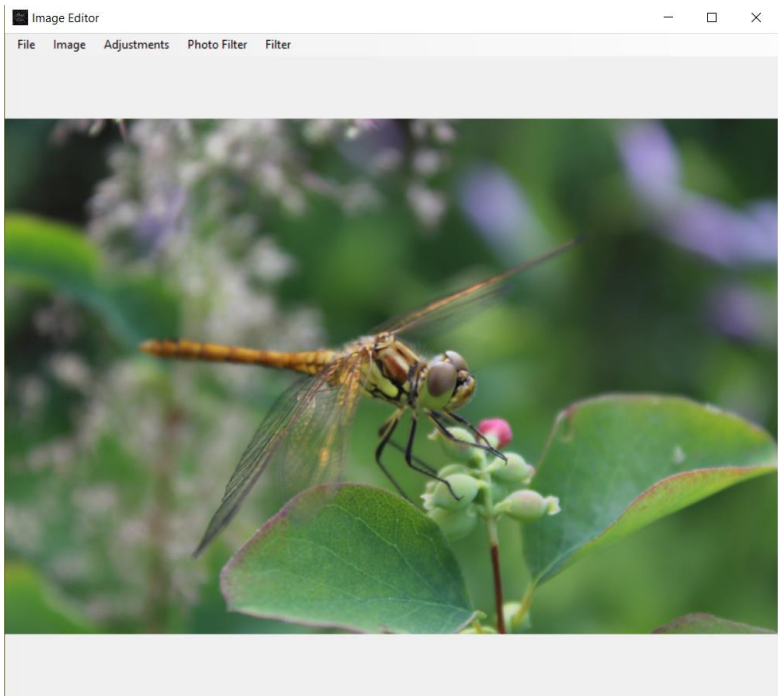

Рисунок 2.2 – Оригинал редактируемого изображения

В разделе ***Adjustments*** имеются различные кнопки с названиями операций. При нажатии на любую из них отобразится вспомогательное окно. На нем есть один или несколько слайдеров, поля для ввода значений, кнопки ***OK*** и ***Cancel***. При перемещении слайдера или изменения значения в поле ввода вызванные изменения отобразятся в рабочей области. При нажатии на кнопку ***OK*** изменения применяются, вспомогательное окошко закрывается. При нажатии на ***Cancel*** окно закрывается, изображение возвращается в исходное состояние.

В разделе ***Image*** есть кнопка ***Undo***. Она отменяет одно последнее изменение.

При выборе фильтров из разделов ***Photo Filter*** или ***Filter*** отфильтрованное изображение появится в рабочей области. При выборе фильтров из раздела ***Filter/Other*** появится вспомогательное окно с кнопками ***Cancel, OK*** и полем для ввода коэффициента интенсивности фильтра. При смене значения в поле ввода изменения отображаются в рабочей области.

При выборе ***SaveAs*** в разделе ***File*** открывается диалоговое окно сохранения файла. При выборе кнопки ***Close*** рабочее поле очистится.

2.2 Работа с изображениями

Преобразование цветности изображения производится путем присвоения в каждый пиксель каждому каналу цветового спектра значения данного пикселя, вычисляемого по заданной формуле.

Функциональным элементом управления является слайдер, что позволяет точно определить границу желаемой степени наложения данного фильтра. Результат работы данного инструмента представлен на рисунке 2.3.

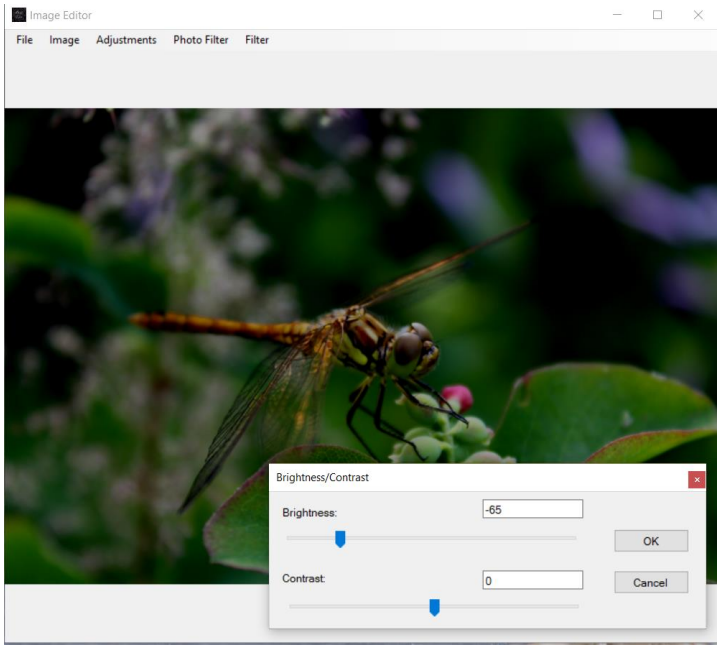

Рисунок 2.3 – Результат работы фильтра ***Brightness***

Фильтр ***Contrast*** позволяет изменять характеристику контрастности обрабатываемого изображения. Результат работы данного инструмента представлен на рисунке 2.4.

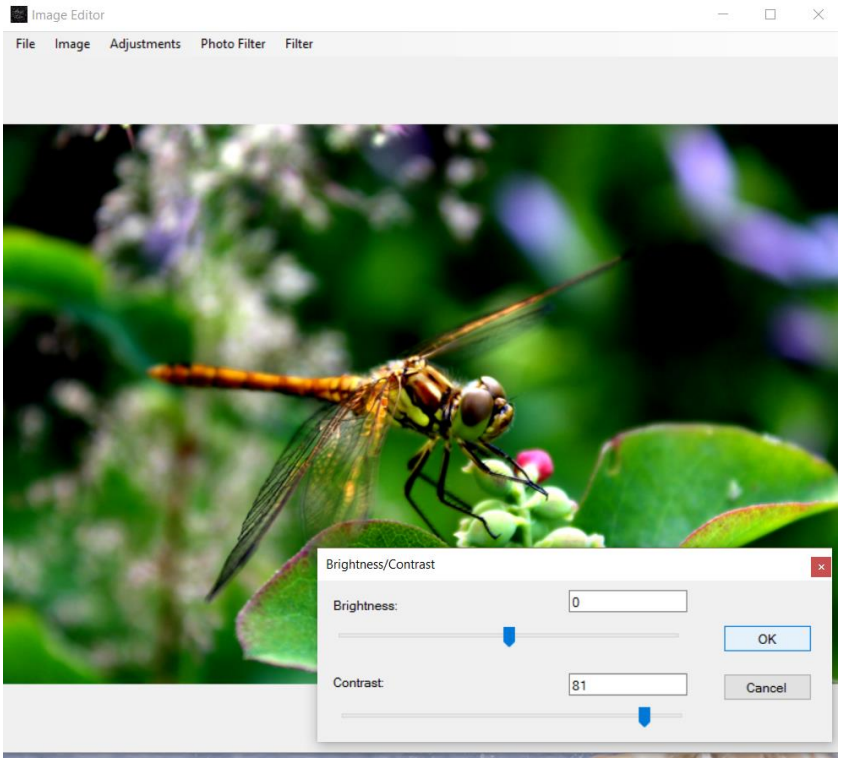

Рисунок 2.4 – Результат работы фильтра ***Contrast***

Фильтр ***BlackAndWhite*** переводит изображение в полутоновый режим. Результат работы данного инструмента представлен на рисунке 2.5.

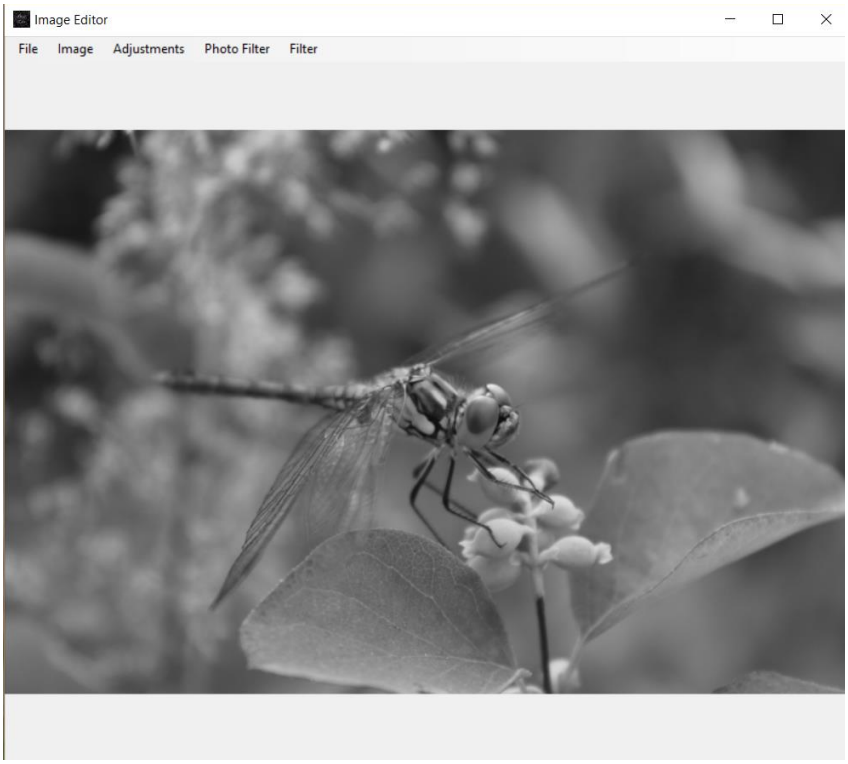

Рисунок 2.5 – Результат работы фильтра ***BlackAndWhite***

Результат работы фильтр ***Sepia*** представлен на рисунке 2.6.

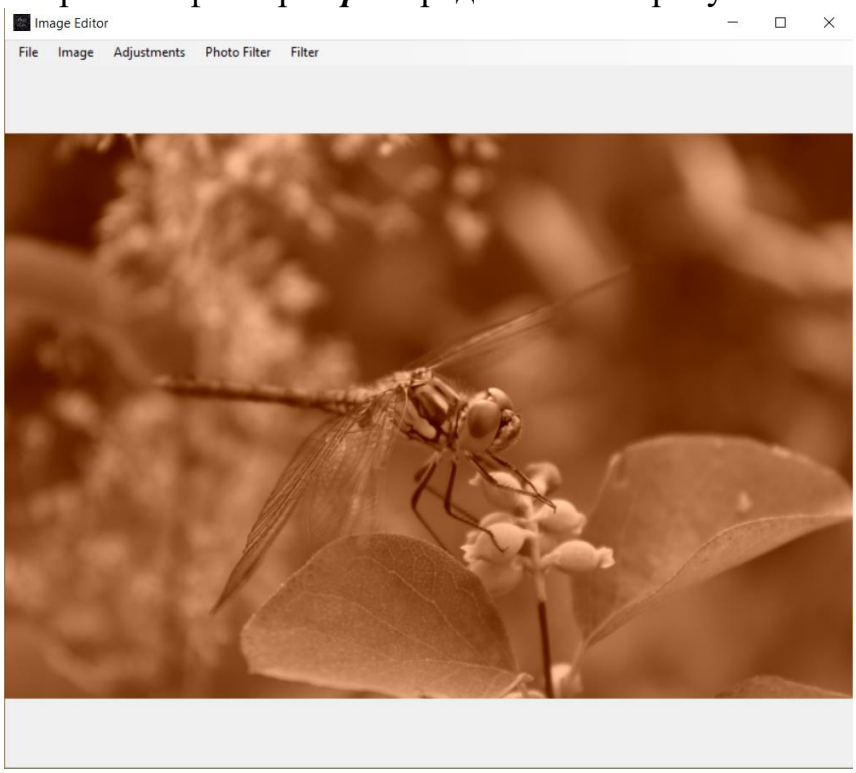

Рисунок 2.6 – Результат работы фильтра ***Sepia***

Фильтр Invert позволяет представить изображение в таком виде, когда каждый пиксель инвертирован – противоположен по цветовому тону оригиналу. Результат работы данного инструмента представлен на рисунке 2.7.

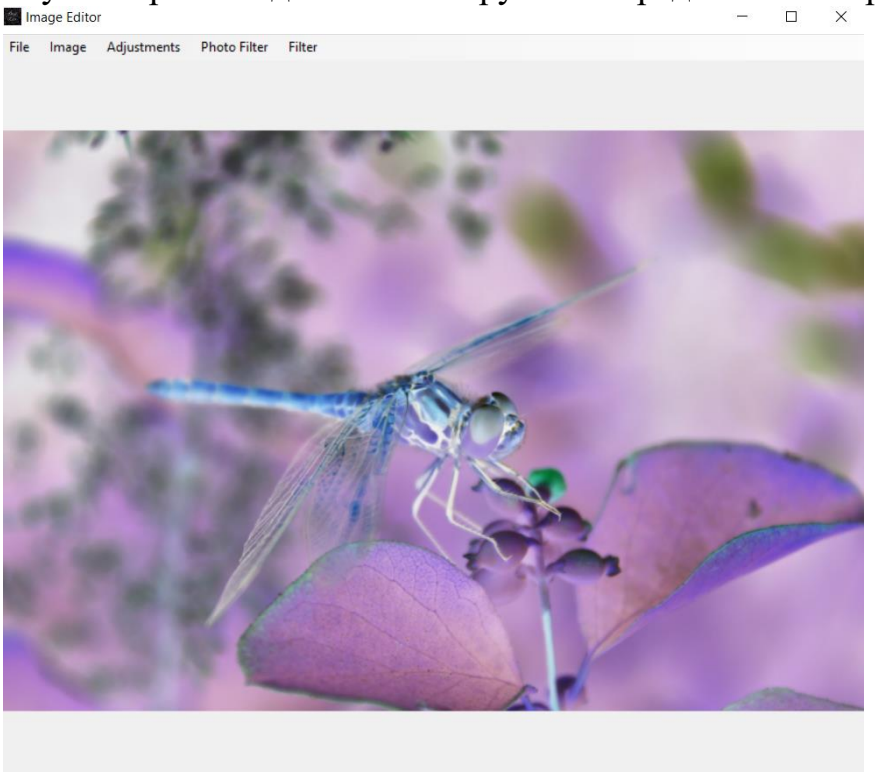

Рисунок 2.7 – Результат работы фильтра ***Invert***

Фильтр ***Threshold*** позволяет перевести изображение в пороговый режим: пиксели с яркостью, меньшей порогового значения, становятся черными, больше – белыми. Результат работы данного инструмента представлен на рисунке 2.8.

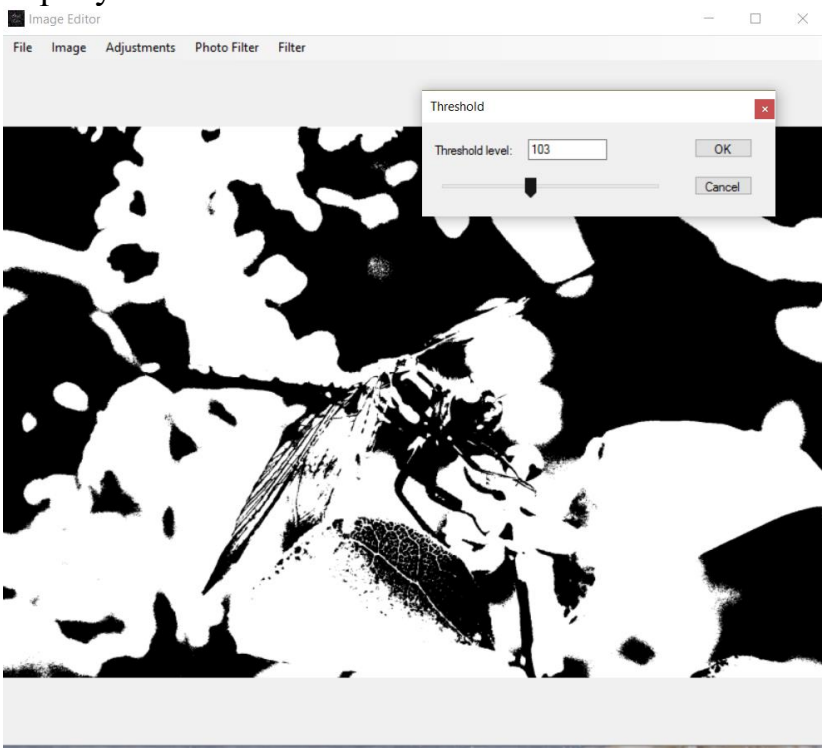

Рисунок 2.8 – Результат работы фильтра ***Threshold***

Фильтр ***Color Balance*** изменяет яркость пикселей одного из RGB-каналов. Результат работы данного инструмента представлен на рисунке 2.9.

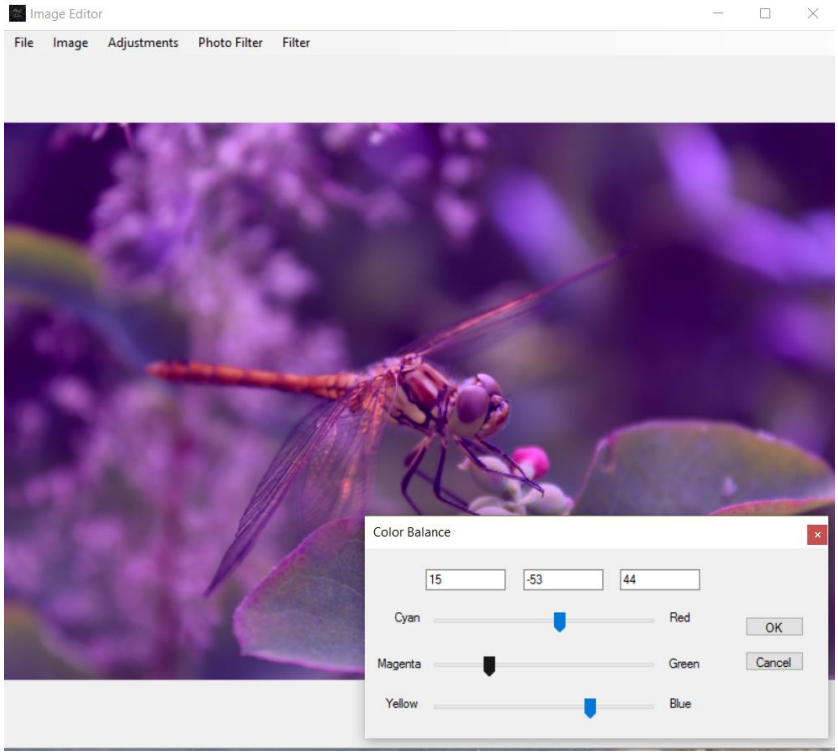

Рисунок 2.9 – Результат работы фильтра ***Color Balance***

Фильтр ***Erosion*** увеличивает размер светлых участков изображения. Результат работы данного инструмента представлен на рисунке 2.10.

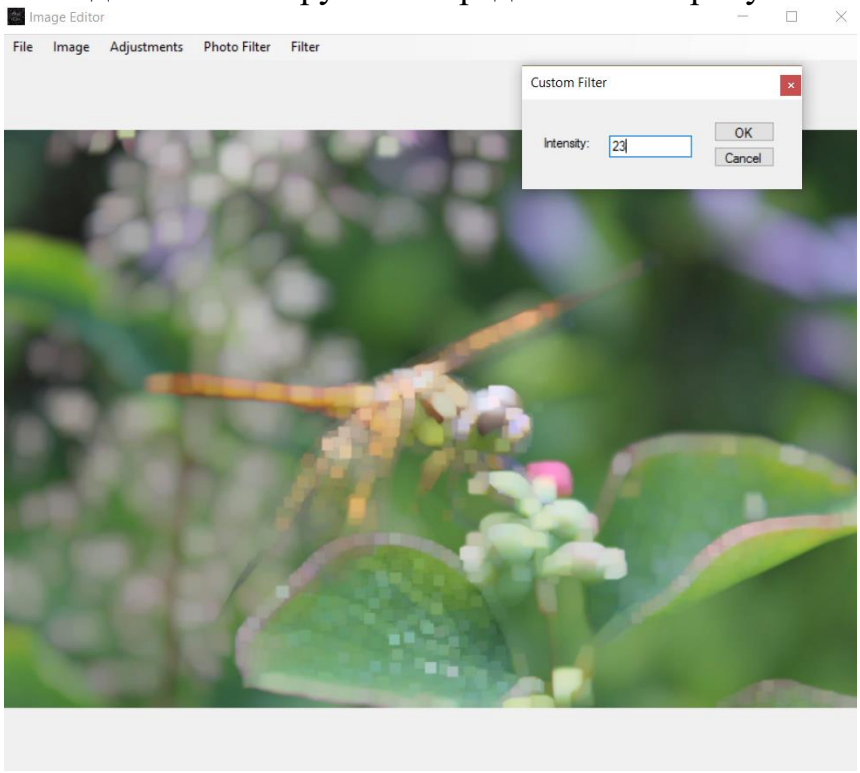

Рисунок 2.10 – Результат работы фильтра ***Erosion***

Фильтр ***Dilution*** увеличивает размер темных участков изображения. Результат работы данного инструмента представлен на рисунке 2.11.

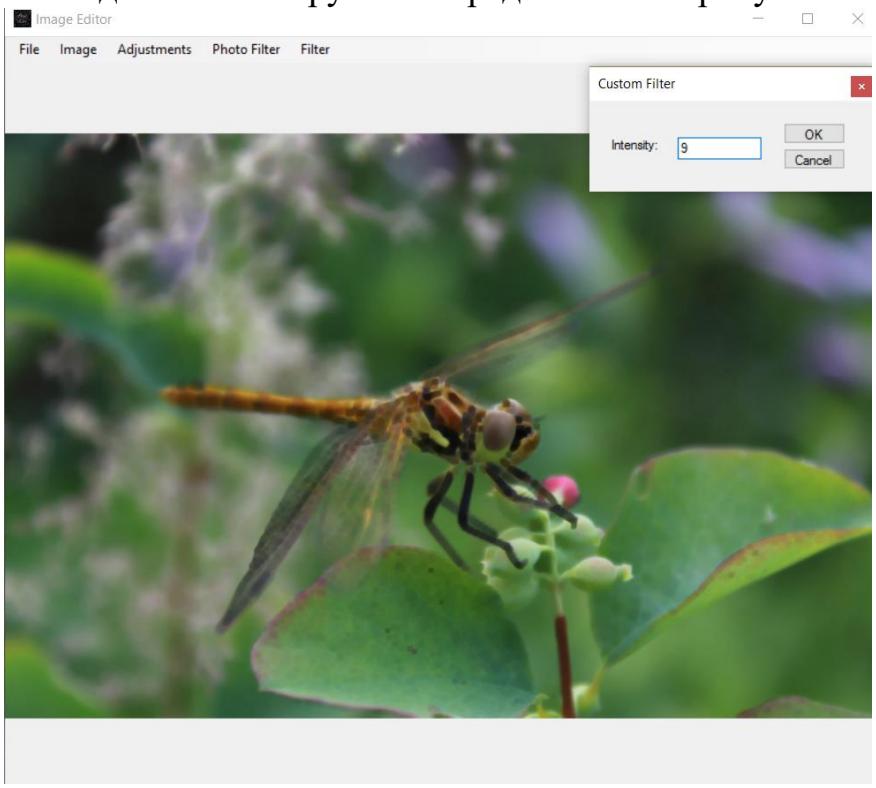

Рисунок 2.11 – Результат работы фильтра ***Dilution***

Фильтр ***Blur*** «размывает» изображение, сглаживая резкие переходы между цветами. Результат работы данного инструмента представлен на рисунке 2.12.

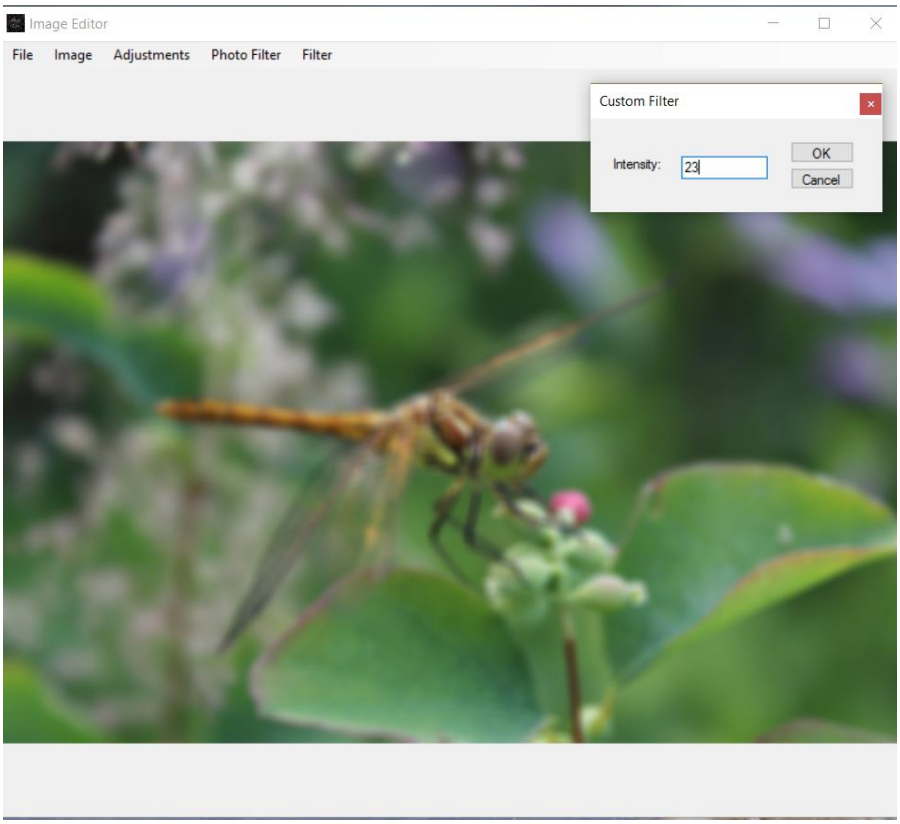

Рисунок 2.12 – Результат работы фильтра ***Blur***

ЗАКЛЮЧЕНИЕ

Результатом работы является приложение ***ImageEditor***, обладающее функциональными возможностями для обработки растровой графики. Для реализации данного проекта на основе изучения базовых алгоритмов обработки растровых изображений была разработана архитектура приложения и определены технические аспекты ее реализации.

Для разработки приложения была выбрана среда Microsoft Visual Studio 2015, платформа .NET Framework 4.6, язык C# 6.0, в которой были созданы его функциональные и обслуживающие компоненты. Это позволило не только закрепить имеющиеся навыки программирования на языке C#, но и освоить новые приемы работы с делегатами и событиями.

Программа ***ImageEditor*** создавалась как приложение ***Windows Forms***. Использовались элементы управления пользовательского интерфейса *Windows Forms* из библиотеки классов *.NET Framework* (классы *Form, Button, ToolBar, TrackBar, PictureBox* и др.).

Основной функционал ***ImageEditor*** включает в себя возможности изменения баланса яркости и контрастности, цветовой насыщенности, наложения фильтров. К сожалению, созданные в курсовой работе режимы являются лишь минимальным набором базовых приемов редактирования, и не отражают всей грандиозности задуманного.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Справочник по C# / Каталог API (Microsoft) и справочных материалов [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/index
2. .NET Framework 4.6 и 4.5 / Каталог API (Microsoft) и справочных материалов[Электронный ресурс] Режим доступа: https://msdn.microsoft.com/ru-ru/library/w0x726c2(v=vs.110).aspx
3. Полное руководство по языку программирования C# 7.0 и платформе .NET 4.7 [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/tutorial/
4. Image Processing Basics in C# [Электронный ресурс] – Режим доступа: https://www.codeproject.com/Articles/31014/Image-Processing-Basics-in-C
5. Fast image processing in C# [Электронный ресурс] – Режим доступа: https://www.codeproject.com/Articles/1080193/Fast-image-processing-in-Csharp
6. Уроки OpenGL. Программирование 3D графики [Электронный ресурс] – Режим доступа: http://esate.ru/uroki/OpenGL/uroki-OpenGL-c-sharp/

ПРИЛОЖЕНИЕ

Таблица 1 – Исходные коды основных методов, созданных для приложения ***ImageEditor***

Элементы класса	Назначение	Программный код
Функциональные компоненты		
Класс Adjustments		
ApplyToPixel	Тип делегата	`private unsafe delegate void ApplyToPixel(byte* blue, double factor);`

Adjust	Применяет к каждому байту делегируемый метод	`private unsafe void Adjust(Bitmap bitmap, double factor, ApplyToPixel applyToPixel)`
		`{`
		`  LockedBitmap lockedBitmap = new LockedBitmap(bitmap);`
		`  Parallel.For(0, lockedBitmap.HeightInPixels, iY =>`
		`  {`
		`  byte* pixel = lockedBitmap.FirstByte + (iY * lockedBitmap.Stride);`
		`  for (int iX = 0; iX < lockedBitmap.WidthInBytes; iX += lockedBitmap.BytesPerPixel)`
		`  {`
		`  applyToPixel(pixel, factor);`
		`  pixel += lockedBitmap.BytesPerPixel;`
		`  }`
		`  });`
		`  lockedBitmap.Unlock(bitmap);`
		`}`

Adjustment	Вспомогательная структура	`struct Adjustment`
		`{`
		`  public ApplyToPixel adjustment;`
		`  public double factor;`
		`  public Adjustment(ApplyToPixel applyToPixel, double f)`
		`  {`
		`  adjustment = applyToPixel;`
		`  factor = f;`
		`  }`

		`}`

Adjust	Применяет к каждому байту все методы, делегируемые элементами списка	`private unsafe void Adjust(Bitmap bitmap, List<Adjustment> adjustmentsPack)`
		`{`
		`  LockedBitmap lockedBitmap = new LockedBitmap(bitmap);`
		
		`  Parallel.For(0, lockedBitmap.HeightInPixels, iY =>`
		`  {`
		`  byte* pixel = lockedBitmap.FirstByte + (iY * lockedBitmap.Stride);`
		`  for (int iX = 0; iX < lockedBitmap.WidthInBytes; iX += lockedBitmap.BytesPerPixel)`
		`  {`
		`  foreach (Adjustment adjustment in adjustmentsPack)`
		`  adjustment.adjustment(pixel, adjustment.factor);`
		
		`  pixel += lockedBitmap.BytesPerPixel;`
		`  }`
		`  });`
		
		`  lockedBitmap.Unlock(bitmap);`
		`}`

	Один из методов, подходящий по сигнатуре делегату *ApplyToPixel*	`private unsafe void ApplyBrightnessToPixel(byte* blue, double factor)`
		`{`
		`  for (Argb i = Argb.blue; i <= Argb.red; ++i)`
		`  {`
		`  byte componentValue = *(blue + (byte)i);`
		`  *(blue + (byte)i) =`
		`  RgbComponentOperations.ChangeBrightness(componentValue, factor);`
		`  }`
		`}`

Класс ImageFilter		

GetRgbComponentNeighborhood	Формирует квадратную окрестность байта изображения	`private unsafe byte[] GetRgbComponentNeighborhood(LockedBitmap lockedBitmapData,`
		`  byte* rgbComponent)`
		`{`
		`  byte[] neighborhood = InitializePixelNeighborhood();`
		`  byte* neighbor = rgbComponent - gap * lockedBitmapData.Stride - gapInBytes;`
		`  int offset = lockedBitmapData.Stride - size * bytesPerPixel;`
		
		`  for (int iY = 0; iY < size; ++iY)`

		{
		for (int iX = 0; iX < size; ++iX)
		{
		neighborhood[iY*size + iX] = *neighbor;
		neighbor += bytesPerPixel;
		}
		
		neighbor += offset;
		}
		
		return neighborhood;
		}

	Применяет фильтр к каждому байту изображения	private unsafe void ApplyFilterToBitmap(LockedBitmap intermediate, LockedBitmap bitmap)
		{
		Parallel.For(0, bitmap.HeightInPixels, iY =>
		{
		byte* outputCurrentByte = bitmap.FirstByte + (iY * bitmap.Stride);
		byte* intermediateCurrentByte =
		intermediate.FirstByte + ((iY + gap) * intermediate.Stride) + gapInBytes;
		
		for (int i = 0; i < bitmap.WidthInBytes; i += bytesPerPixel)
		{
		for (Argb j = Argb.blue; j <= Argb.red; ++j)
		*(outputCurrentByte + (byte)j) =
		ComputeNewRgbComponentValue(GetRgbComponentNeighborhood(intermediate,
		I intermediateCurrentByte + (byte)j));
		
		*(outputCurrentByte + (byte)Argb.alfa) =
		*(intermediateCurrentByte + (byte)Argb.alfa);
		
		outputCurrentByte += bytesPerPixel;
		intermediateCurrentByte += bytesPerPixel;
		}
		});
		}

Класс KernelFilter		

ComputeNewRgbComponentValue	Реализация абстрактного метода класса-предка	`protected override byte ComputeNewRgbComponentValue(byte[] neighborhood)`
		`{`
		`  double sum = 0;`
		
		`  for (int i = 0; i < neighborhood.Length; ++i)`
		`  sum += neighborhood[i] * kernel[i];`
		
		`  sum *= factor;`
		
		`  return RgbComponentOperations.ControlOverflow(sum);`
		`}`

Класс CustomFilter		
ComputeRgbComponentValue	Тип делегата	`public delegate byte ComputeRgbComponentValue(byte[] neighborhood);`
	Поле типа делегата	`ComputeRgbComponentValue computeRgbComponentValue;`

ComputeNewRgbComponentValue	Реализация абстрактного метода класса-предка	`protected override byte ComputeNewRgbComponentValue(byte[] neighborhood)`
		`{`
		`  return computeRgbComponentValue(neighborhood);`
		`}`

	Метод, подходящий по сигнатуре делегату *ComputeRgbComponentValue*	`private byte ApplyErosion(byte[] neighborhood)`
		`{`
		`  byte maximum = neighborhood[0];`
		
		`  for (int i = 0; i < neighborhoodSize; ++i)`
		`  if (neighborhood[i] > maximum)`
		`  maximum = neighborhood[i];`
		
		`  return maximum;`
		`}`

Класс RgbComponentOperations		

ControlOverflow	Контролирует переполнение RGBA-компонента	`public static byte ControlOverflow(double componentValue)`
		`{`
		`  return (byte)(componentValue < 0 ? 0 : (componentValue > 255 ? 255 : componentValue));`
		`}`

	Один из методов класса	`public static byte Threshold(byte red, byte green, byte blue, byte threshold)`
		`{`
		`  return (byte)(0.2125 * red + 0.7154 * green + 0.0721 * blue > threshold ? 255 : 0);`
		`}`

Служебные компоненты		

ImageProcessingEventArgs	Передает ссылки на изображение при генерации событий	`public class ImageProcessingEventArgs`
		`{`
		`  public Bitmap Image { get; set; }`
		`  public ImageProcessingEventArgs(Bitmap bitmap)`
		`  {`
		`  Image = bitmap;`
		`  }`
		`  public ImageProcessingEventArgs() { }`
		`}`

ImageProcessingEventHandler	Тип делегата для основных событий	`public delegate void ImageProcessingEventHandler(object sender, ImageProcessingEventArgs e);`
AdjustmentCall	Событие главной формы	`public event ImageProcessingEventHandler AdjustmentCall;`

OnAdjustmentCall	Генерирует событие главной формы	`protected virtual void OnAdjustmentCall()`
		`{`
		`  AdjustmentCall?.Invoke(this, new ImageProcessingEventArgs(workingCopy));`
		`}`

	Пример генерации	`private void brightnessContrastToolStripMenuItem_Click(object sender, EventArgs e)`
		`{`
		`  BackUpWorkingCopy();`
		`  OnAdjustmentCall();`
		`  brightnessContrastForm.ShowDialog();`

	события в главной форме	}

	Обработчик события главной формы	public void SetInputImage(object sender, ImageProcessingEventArgs e)
		{
		input = e.Image;
		}

ProcessingCompleted	Событие вспомогательной формы	public event ImageProcessingEventHandler ProcessingCompleted;

OnProcessingCompleted	Генерирует события вспомогательной формы	protected virtual void OnProcessingCompleted(Bitmap processedImage)
		{
		ProcessingCompleted?.Invoke(this, new ImageProcessingEventArgs(processedImage));
		}

	Пример генерации события в вспомогательной форме	private void brightnessTrackBar_ValueChanged(object sender, EventArgs e)
		{
		Bitmap preview = new Bitmap(input);
		brightnessContast.AdjustBrightnessAndContrast(preview, brightnessTrackBar.Value,
		contrastTrackBar.Value);
		OnProcessingCompleted(preview);
		}

ViewProcessedImage	Обработчик события вспомогательной формы	private void ViewProcessedImage(object sender, ImageProcessingEventArgs e)
		{
		pictureBox.Image = e.Image;
		}