ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

КАФЕДРА ИНФОРМАЦИОННЫХ И РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

РАЗРАБОТКА ГРАФИЧЕСКОГО РЕДАКТОРА ДЛЯ МЕДИЦИНСКИХ ИЗОБРАЖЕНИЙ

Выпускная квалификационная работа

обучающейся по направлению подготовки
12.03.04 «Биотехнические системы и технологии»
очной формы обучения,
группы 12001514
Бугрим Алины Геннадьевны

Научный руководитель
к.т.н, доцент Шамраева Е.О.

БЕЛГОРОД 2019

РЕФЕРАТ

Разработка графического редактора для медицинских изображений – Бугрим Алина Геннадьевна, выпуская квалификационная работа бакалавра, Белгород, Белгородский государственный национальный исследовательский университет (НИУ «БелГУ»), количество страниц 61, включая приложения 87, количество рисунков 51, количество таблиц 2, количество использованных источников 27.

КЛЮЧЕВЫЕ СЛОВА: графический редактор, рентгенограмма, томограмма, повышение визуального качества, выделение костных структур, цифровые методы обработки изображений.

ОБЪЕКТ ИССЛЕДОВАНИЯ: процесс цифровой обработки медицинских изображений.

ПРЕДМЕТ ИССЛЕДОВАНИЯ: графический редактор для обработки медицинских изображений.

ЦЕЛЬ РАБОТЫ: повышение качества вторичной обработки медицинских изображений персоналом больницы за счет разработки графического редактора для медицинских изображений

ЗАДАЧИ ИССЛЕДОВАНИЯ: обзор методов цифровой обработки изображений; анализ особенностей обработки медицинских изображений; проектирование графического редактора для обработки медицинских изображений; программная реализация графического редактора для медицинских изображений

МЕТОДЫ ИССЛЕДОВАНИЯ: медианный, линейный и морфологический фильтры; лапласиан, градиент и морфологический метод выделения границ; степенной фильтр; эквализация гистограммы; пороговая сегментация; преобразование в негатив; оценка качества изображений: PSNR, СКО.

ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ: спроектирован и разработан графический редактор для медицинских изображений.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ И ТЕРМИНОВ

БФВЧ – фильтр высоких частот Баттерворта

БФНЧ – фильтр низких частот Баттерворта

ГФВЧ – гауссов фильтр высоких частот

ГФНЧ – гауссов фильтр низких частот

ИФВЧ – идеальный фильтр высоких частот

ИФНЧ – идеальный фильтр низких частот

КТ – компьютерная томография

МедГраф – графический редактор для обработки медицинских изображений

РГ – рентгенография

СКО – среднеквадратичное отклонение

ФВЧ – фильтр высоких частот

ФНЧ – фильтр низких частот

DFD – (Data flow diagrams) диаграмма потоков данных

PSNR – (Peak signal-to-noise ratio) пиковое отношение сигнала к шуму

UML – (Unified Modeling Language) унифицированный язык моделирования

ВВЕДЕНИЕ

В настоящее время развитие медицины находится на достаточно высоком уровне. То, что казалось невозможным несколько лет назад, на сегодняшний день получает реальное воплощение в жизнь. Современные клиники и медицинские центры трудно представить без оснащения разнообразным инновационным оборудованием, используемым для лечения и профилактики множества заболеваний.

Однако не всегда достаточно иметь медицинское оборудование высшего класса, чтобы обеспечить лечение пациента. Прежде всего необходимо правильно установить диагноз, а для этого сотрудник медицинского учреждения нуждается в достаточном количестве информации, благодаря которой сможет назначить курс лечения. При установлении диагноза и проведении лечения врачи все больше полагаются на данные медицинский исследований, наиболее информативными из которых являются биомедицинские изображения. Медицинское или диагностическое изображение представляет собой структурно-функциональный образ органов человека, который необходим для диагностики заболеваний и изучения анатомо-физиологической картины организма.

Основной источник медицинской информации, получаемый с помощью средств лучевой диагностики, можно поделить на две группы: аналоговой и цифровой медицинский образ. К первой относятся изображения, в которых заключена информация непрерывного характера, например, пленочная рентгенограмма, а вторая имеет в своей основе ячеистую структуру, содержащую информацию в виде цифр; к данной группе можно отнести снимки, получаемые с помощью компьютерного томографа, цифрового рентгеновского аппарата. Для хранения медицинских изображений с целью отслеживания динамики развития заболевания и эффективности лечения аналоговые снимки оцифровывают. При этом искажается или теряется часть данных. Изначально цифровые изображения имеют лучшее качество, чем

оцифрованные аналоговые снимки. Однако и они не лишены дефектов. Врачу бывает сложно поставить диагноз по исходному изображению. Поэтому актуальной задачей является разработка графического редактора для вторичной обработки медицинских изображений, с помощью которого врач сможет в ручном, полуавтоматическом или автоматизированном режиме улучшить качество изображений для более точной диагностики.

Целью выпускной квалификационной работы является повышение качества вторичной обработки медицинских изображений персоналом больницы за счет разработки графического редактора для медицинских изображений.

Для достижения поставленной цели в выпускной квалификационной работе необходимо решить следующие задачи:

- провести обзор методов цифровой обработки изображений;
- проанализировать особенности обработки медицинских изображений;
- спроектировать графический редактор для обработки медицинских изображений;
- реализовать графический редактор для медицинских изображений на языке высокого уровня.

Объектом является процесс цифровой обработки медицинских изображений.

Предметом является графический редактор для обработки медицинских изображений.

Пояснительная записка состоит из введения, трех разделов, заключения и списка использованных источников.

1 Анализ проблемы обработки медицинских изображений

При установлении диагноза и проведении лечения врачи все больше полагаются на медицинские изображения, которые дают основной объем информации о пациенте и его заболевании.

Основываясь на медицинских данных, а именно на медицинских изображениях, врач получает соответствующую информацию для анализа. Однако не всегда по изображению можно однозначным способом идентифицировать заболевание. Это связано с тем, что происходят отклонения результатов измерений медицинских переменных от реальных значений, так как существуют погрешности измерений, различные помехи, электро-магнитные наводки неправильная настройка оборудования съема и первичной обработки информации [13].

Также медицинские снимки могут попасть под воздействие некоторых факторов окружающей среды, что негативно повлияет на их состояние и обеспечит появление артефактов. В результате, качество медицинской информации влияет на качество диагностики заболеваний и, исходя из этого, на требуемое лечение. Поэтому для улучшения качества медицинских данных производят их вторичную обработку.

Обработка медицинских изображений представляет собой технологию выявления внутренних скрытых элементов изображения, которые являются практически невидимыми без процесса проведения каких-либо манипуляций с изображением, с целью его изменения. Данная процедура не должна искажать исходные данные, а обязана выявить тонкие структуры органов при разных видах исследований, специально визуализированных и усиленных для качественной диагностики [22].

1.1 Обзор методов обработки изображений

Изображение можно определить как двумерную функцию $f(x, y)$, где x и y координаты в пространстве (на плоскости). Значение f в любой точке, которая задается данными координатами, называется интенсивностью или уровнем серого (яркостью) изображения в этой точке [8].

Если величины f, x, y принимают конечное число дискретных значений, то изображение называется цифровым. Обработку таких изображений производят с помощью компьютеров [18].

1.1.1 Типы цифровых изображений

Цифровое изображение состоит из пикселей – наименьших элементов изображения, каждый из которых расположен в конкретном месте и принимает определенное значение. В зависимости от способа описания изображения бывают следующих типов : двухуровневое, полутоновое, цветное, непрерывно-тоновое, дискретно-тоновое, а также изображение, подобное мультфильмам [8].

Двухуровневое (монохроматическое) изображение – все пиксели могут иметь только два значения: двоичная 1 и двоичный 0 (черный и белый цвета). Каждый пиксел такого изображения представляется всего одним битом.

Полутоновое изображение – каждый пиксель такого изображения может иметь $2n$ значений от 0 до $2n-1$, обозначающих одну из $2n$ градаций серого цвета.

Цветное изображение – существует несколько методов задания цвета, но в каждом из них участвуют три или четыре параметра. Обычно цветной пиксель состоит из 3 байтов (по 1 байту на каждый параметр).

Непрерывно-тоновое изображение – этот тип изображений может иметь много похожих цветов или полутонов. Такие изображения могут содержать области, в которых цвет кажется глазу непрерывно меняющимся. В этом случае пиксель представляется или большим числом (в полутоновом случае) или тремя

компонентами (в случае цветного образа). Обычно это естественное изображение. Пример: фото природы, рентгенограмма.

Дискретно-тоновое изображение (синтетическое) – обычно это изображение получается искусственным путем. В нем может быть всего несколько цветов или много цветов, но в нем нет шумов или пятен естественного изображения. Примеры: фото искусственных объектов, скан страницы текста, карты, томограммы. Объекты и фон на дискретно-тоновых изображениях резко контрастируют.

Изображения, подобные мультфильмам – это цветные изображения, в которых присутствуют большие области одного цвета. При этом соприкасающиеся области могут сильно различаться по своему цвету.

Зрение является наиболее совершенным из органов чувств человека и максимальное количество информации человек воспринимает глазами. Однако, в отличие от людей, способных воспринимать электромагнитное излучение лишь в видимом диапазоне, машинная обработка изображений охватывает практически весь электромагнитный спектр от гамма-излучения до радиоволн [8,9,16].

Во всем диапазоне от обработки изображения до машинного зрения можно различить компьютерные процессы низкого, среднего и высокого уровней [8].

Процессы низкого уровня касаются только примитивных операций типа предобработки с целью уменьшения шума, повышения контраста или улучшения резкости изображений. На входе и на выходе присутствуют изображения;

Процессы среднего уровня охватывают такие задачи, как сегментация (разделение изображений на области или выделение на нем объектов), описание объектов и сжатие их в удобную для компьютерной обработки форму, а также классификация (распознавание) отдельных объектов. На входе изображение, на выходе – атрибуты, извлекаемые из изображения (например, границы областей, линии контуров);

Процессы высокого уровня включают в себя «осмысление» набора распознанных объектов, как это делается в анализе изображений.

В настоящее время в области обработки и анализа изображений выделяют пять основных классов методов и алгоритмов [8].

Улучшение качества – данный класс состоит из методов, которые используются для уменьшения шумов и удаления артефактов, а также повышения контраста области интереса на изображениях. Возможность использования тех или иных процедур улучшения качества существенно зависит от того, как будет проводиться последующий анализ – визуально или с помощью соответствующих компьютеризированных методов. В случае компьютерного анализа, количество применяемых алгоритмов улучшения визуального качества изображений должно быть сведено к минимуму.

Сегментация изображений – отделение анализируемого объекта, структуры или области от окружающего фона. Сегментация принадлежит к числу базовых шагов, качество выполнения которых во многом определяет точность, а порой даже саму возможность дальнейшего компьютеризированного анализа изображений. Методы сегментации базируются на яркостной, градиентной и текстурной информации изображения [21].

Количественный анализ применяется к отсегментированным объектам изображения с целью выделения существенной диагностической информации: размер, форма, текстура объектов и т.д. Данный класс методов используется для совмещения двух цифровых изображений одной и той же части человеческого тела. Данный процесс является целесообразным, если полученная в результате совмещения карта может быть использована для последующей обработки и анализа изображения. Совмещаемыми изображениями могут быть снимки одного и того же пациента в разных ситуациях или в разное время. Также в процессе диагностики часто возникает необходимость в совмещении изображения пациента с изображением здорового человека. Получаемая карта

соответствия может использоваться для попиксельного сравнения изображений, мониторинга изменения формы и роста новообразований и т.п.

Следующий класс методов используется для сжатия, архивирования, хранения, а также поиска в базах данных. Роль этого направления постоянно возрастает, так как за последние годы значительно увеличилось количество и размеры снимаемых цифровых диагностических изображений [4]. Ежедневно делается огромное количество медицинских снимков, над которыми необходимо проводить различного рода манипуляции, что подразумевает их последующее хранение, передачу, поиск и т.д.

Визуализация и виртуальная реальность. Используемые здесь методы и алгоритмы развиваются на стыке трехмерной компьютерной графики, систем компьютеризированной диагностики, а также различного рода тренажеров и образовательных систем, базирующихся на концепции погружения в виртуальную реальность.

На сегодняшний день развитие новых технологий и цифровой техники привело к появлению большого количества новых методов обработки, которые можно разделить на две категории: пространственные и частотные [14, 17, 26].

1.1.2 Пространственные методы обработки изображений

Категория методов, которые применяются в пространственной области, объединяет подходы, основанные на прямом манипулировании пикселями изображения. Частотные, в свою очередь, основываются на модификации сигнала, формируемого путем применения преобразования Фурье к обрабатываемому изображению [8].

Процессы пространственной области описываются уравнением:

$$g(x,y)=T(f(x,y)), \qquad (1)$$

где $f(x,y)$ – входное изображение; $g(x,y)$ – обработанное изображение;
T – оператор над f.

Методы, применяемые в данной области делятся на четыре категории.

Первая категория – градационные преобразования. Включает в себя преобразование изображения в негатив, логарифмическое преобразование, степенное преобразование, кусочно-линейные функции преобразования [15]. Функция градационного преобразования имеет следующий вид:

$$s=T(r), \tag{2}$$

где r и s – значения яркостей входного и выходного изображений в каждой точке (x,y).

Негатив – этот тип обработки, в большей мере подходит для усиления белых или серых деталей на фоне темных областей изображения. Преобразование изображения в негатив с яркостями в диапазоне $[0,L-1]$ представлено на рисунке 1 и осуществляется с использованием негативного преобразования, определяемого выражением определяется выражением [8]:

$$s=L-1-r \tag{3}$$

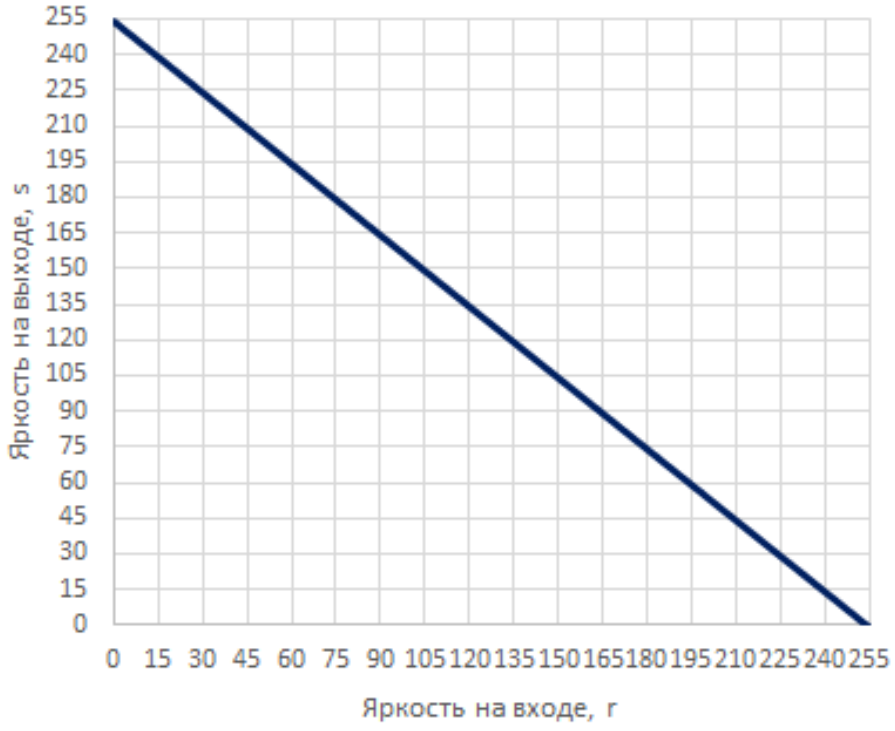


Рисунок 1 – Преобразование яркости

Пример преобразования изображения в негатив представлен на рисунке 2.

а) б)

Рисунок 2 – Преобразование изображения в негатив: а) исходное изображение; б) негатив

Логарифмическое преобразование отображает узкий диапазон малых значений яркостей на исходном изображении в более широкий диапазон выходных значений и наоборот для больших значений входного сигнала (рисунок 3,а). Общий вид логарифмического преобразования имеет вид [8]:

$$s=c\log(1+r), \quad (4)$$

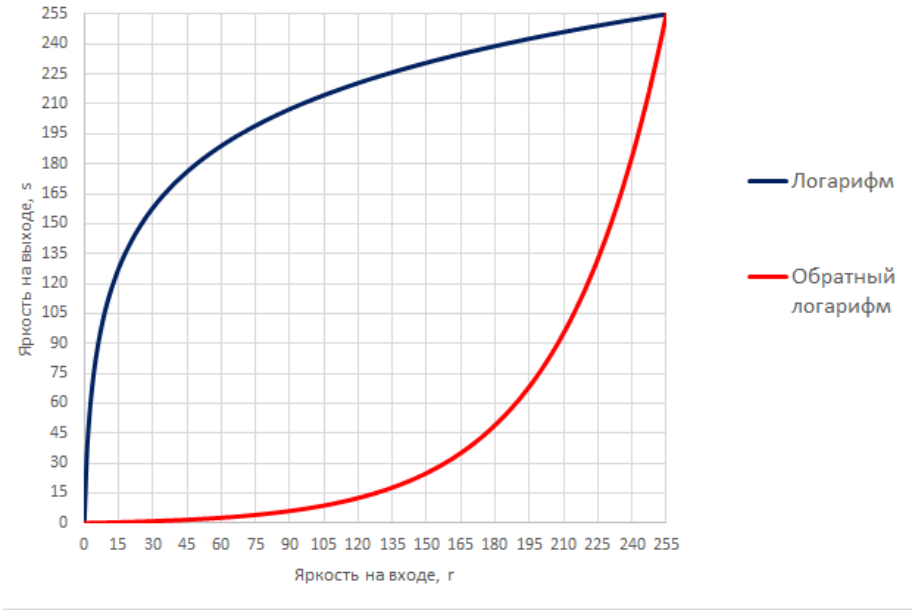
где c – постоянная величина, задаваемая пользователем, $r \geq 0$.

Используется для растяжения диапазона значений темных пикселей на изображении с одновременным сжатием диапазона значений ярких пикселей.

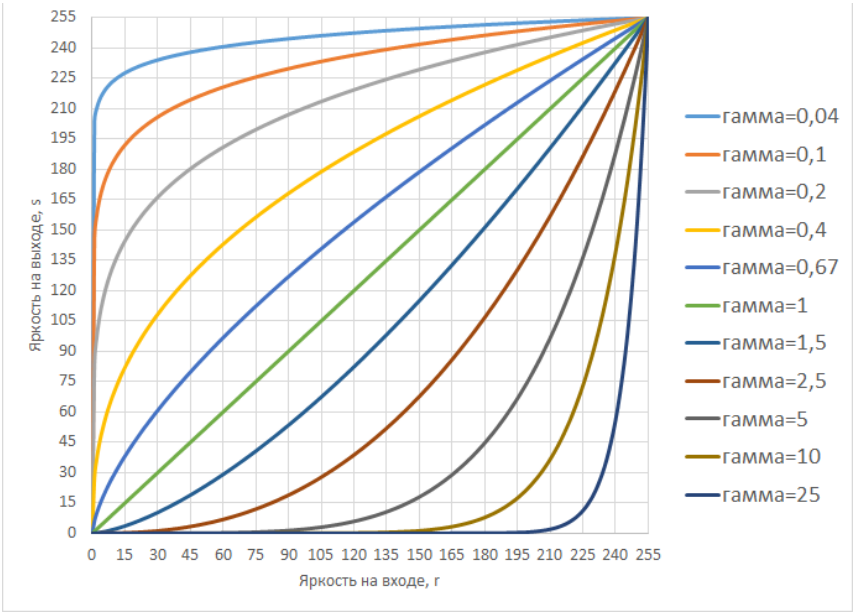
Степенное преобразование подобно по действию логарифмического преобразования, но имеет более широкий диапазон преобразования изображений (рисунок 3,б). Используется также для управления контрастом изображения и задается выражением [8]:

$$S=cr^{y}, \quad (5)$$

где r – яркость исходного изображения; s – яркость преобразованного изображения; c, γ – положительные константы.



а)



б)

Рисунок 3 – Пространственные методы повышения контраста:
а) логарифмическое преобразование; б) степенное преобразование

Для реализации степенного преобразования возможен вариант автоматического расчета значения гаммы, при котором по гистограмме интенсивности r определяется глобальный (I^0_{max}) и k локальных (I^k_{max})

максимумов интенсивностей и для каждого локального максимума гистограммы производился расчет коэффициентов α_k [24,25]:

$$\alpha_k = \frac{I_{max}^k}{I_{min}^k + 1}, \quad (6)$$

где I_{min}^k – значение k-го локального максимума; I_{min}^k – значение локального минимума в диапазоне между глобальным максимумом и рассматриваемым локальным.

Тот из локальных максимумов, для которого значение α_k окажется наибольшим, выбирается в качестве второго основного максимума гистограммы. Далее выбирается значение локального минимума I_{min}^k, расположенное между глобальным и найденным локальным максимумами. Расчет значения γ осуществляется по следующей формуле:

$$\gamma = \frac{128}{256 - I_{min}^k} \quad (7)$$

Пример использования степенного преобразования представлен на рисунке 4.

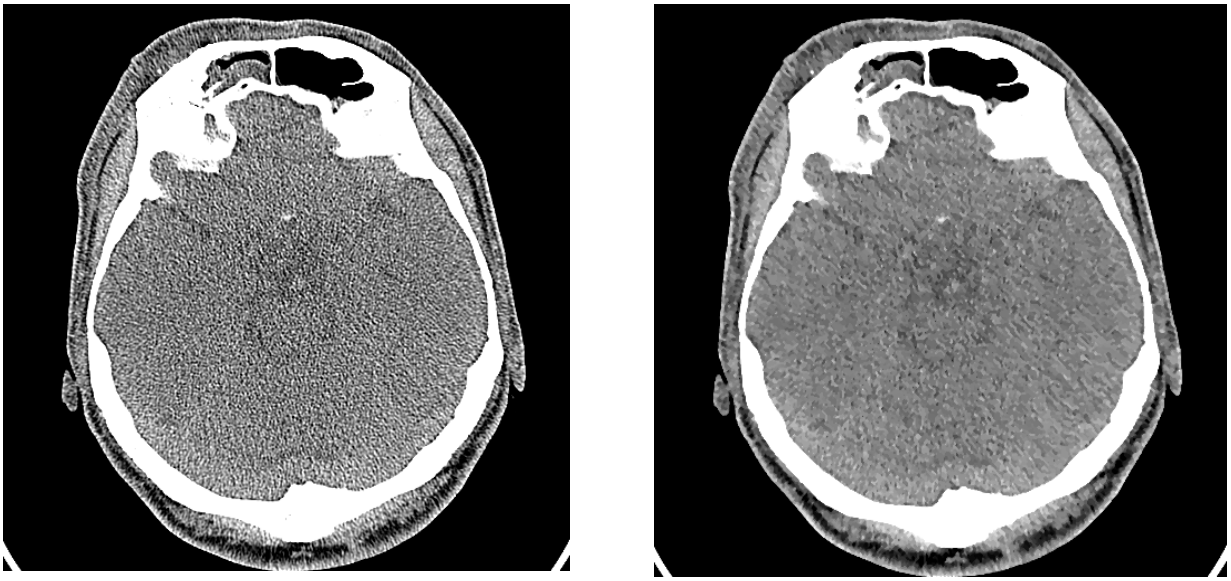
а) б)

Рисунок 4 – Степенное преобразование: а) исходное изображение; б) результат применения степенного преобразования

Вторая категория – сглаживающие пространственные фильтры, которые используются для расфокусировки изображения, а также подавления шума и делятся на линейные и нелинейные [13].

Результатом применения линейной фильтрации, является замена исходных значений элементов изображения средним по маске фильтра, что способствует уменьшению резких перепадов уровней яркости [8].

Недостатком использования данного вида фильтрации является то, что при устранении шумовых элементов, характеризующихся резким перепадом яркости, происходит расфокусировка контуров изображения, так как они тоже имеют резкий перепад яркости [11]. Пример использования линейной фильтрации представлен на рисунке 5.



а) б)

Рисунок 5 – Линейная фильтрация: а) исходное изображение; б) результат применения линейной фильтрации

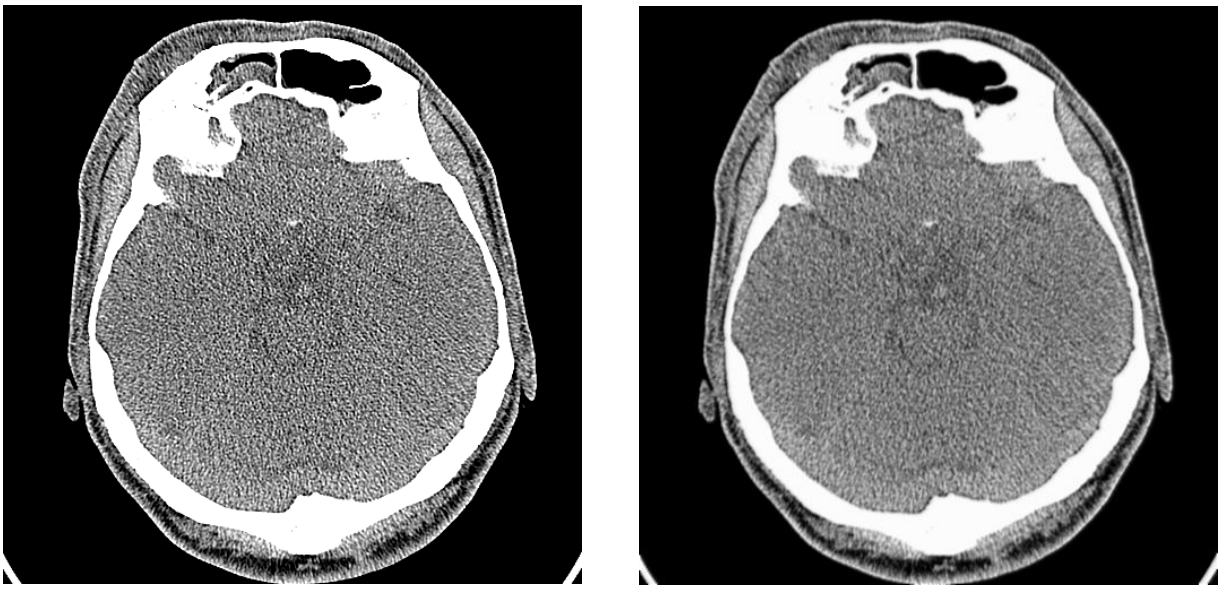
Процесс нелинейной пространственной фильтрации также происходит по окрестности, однако нелинейно зависит от значений элементов этой окрестности. Наиболее известен медианный фильтр, который заменяет значение пикселя на значение медианы распределения всех пикселей в

окрестности. Особенностью данного фильтра является то, что он без проблем справляется с устранением таких шумов, как «соль и перец», представляющих собой случайно возникающие черные и белые пиксели [8].

$$\hat{f}(x,y) = \underset{(s,t)\in S_{xy}}{\text{med}}\{g(s,t)\}, \tag{8}$$

где $f(x,y)$ – некоторое приближение исходного изображения по заданному (искаженному) изображению g(x,y); g(x,y) – искаженное шумом изображение; S_{xy} – окрестность размером $n \times n$.

Результат применения медианного фильтра, используя маску размером 5x5, изображен на рисунке 6.



а) б)

Рисунок 6 – Медианный фильтр: а) исходное изображение; б) результат применения медианного фильтра

Фильтр максимума, который является полезным при обнаружении ярких точек на изображении, основан на выборе максимального значения и задается выражением:

$$\hat{f}(x,y) = \max_{(s,t)\in S_{xy}} \{g(s,t)\}, \tag{9}$$

Фильтр минимума служит для выявления темных точек на изображении, при этом основан на выборе минимального значения:

$$\hat{f}(x,y) = \min_{(s,t)\in S_{xy}} \{g(s,t)\}, \tag{10}$$

Также существует фильтр срединной точки, применение которого заключается в вычислении среднего значения между максимальным и минимальным в определенной окрестности изображения:

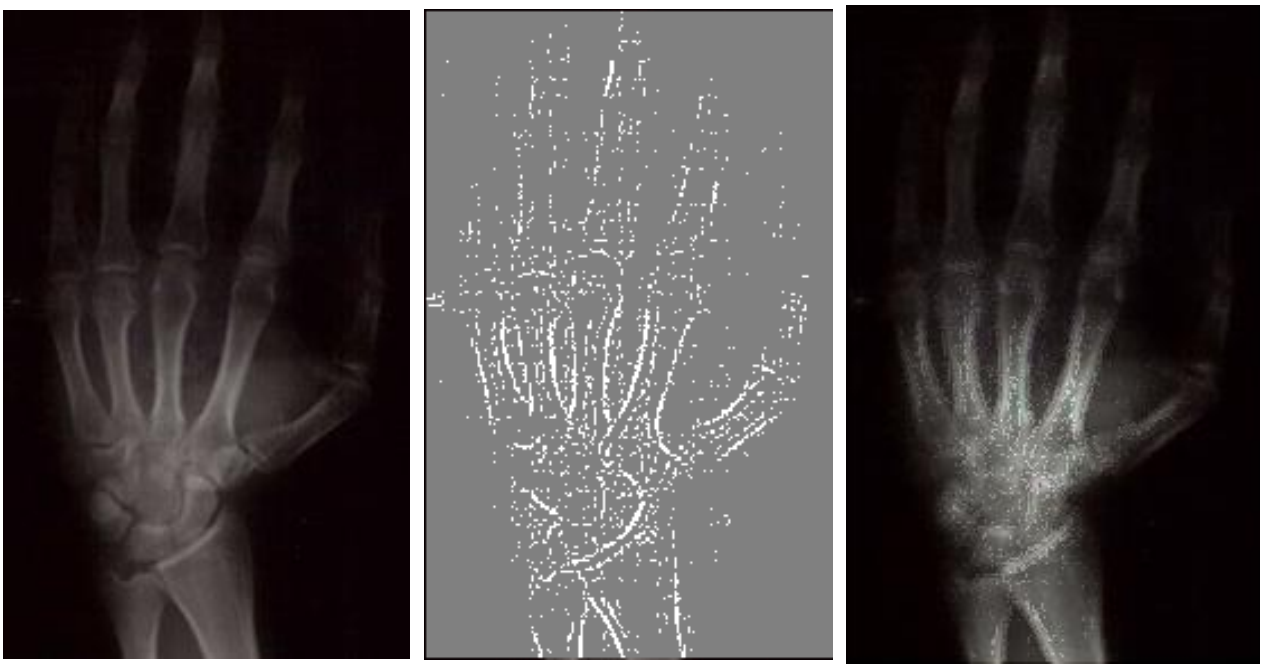
$$\hat{f}(x,y) = \frac{1}{2}\left(\max_{(s,t)\in S_{xy}} \{q(s,t)\} + \min_{(s,t)\in S_{xy}} \{q(s,t\} \right) \tag{11}$$

Третья категория – пространственные фильтры повышения резкости. Основной целью применения фильтров данной категории является подчеркивание деталей изображения, а также улучшение тех частей изображения, которые были лишены фокусировки в результате некорректного метода съемки [8].

Одним из известных методов пространственной фильтрации с целью повышения резкости является лапласиан (улучшение изображения с использованием вторых производных), который подчеркивает разрывы уровней яркости на изображении, а также подавляет области со слабыми изменениями яркости. Обобщенный алгоритм данного метода представлен выражением:

$$q(x,y) = \begin{cases} f(x,y) - \nabla^2 f(x,y), \text{если } w(0,0) < 0 \\ f(x,y) - \nabla^2 f(x,y), \text{если } w(0,0) \geq 0, \end{cases} \tag{12}$$

где w (0,0) – значение центрального коэффициента маски лапласиана. Результат применения лапласиана представлен на рисунке 7.

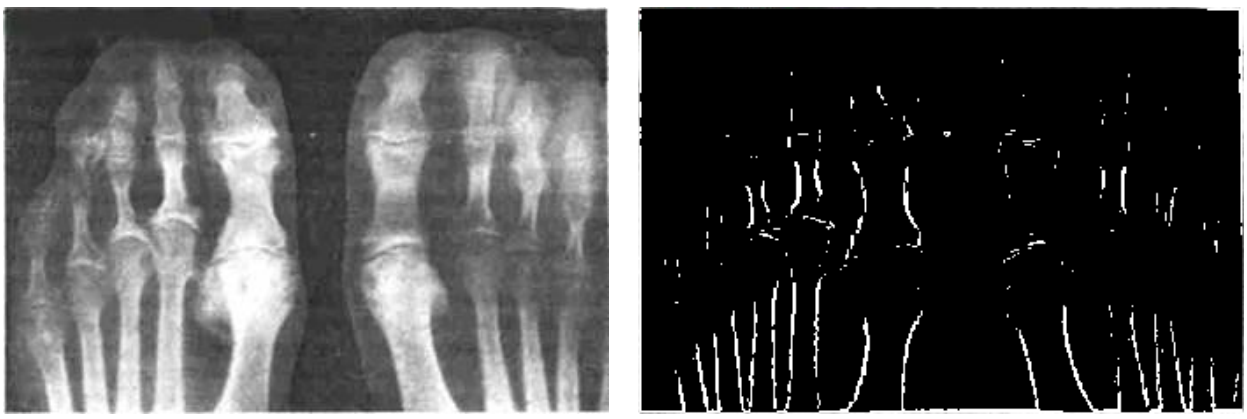


а) б) в)

Рисунок 7 – Результат улучшения визуального качества изображения с помощью лапласиан а) исходное изображение; б) лапласиан; в) исходное изображение умноженное на лапласиан

Улучшение изображений с использованием первых производных (градиент). Используется для обнаружения дефектов в техническом контроле или как предварительная обработка при автоматизированном контроле [19].

Выделение контуров с помощью градиента Собеля представлен на рисунке 8.

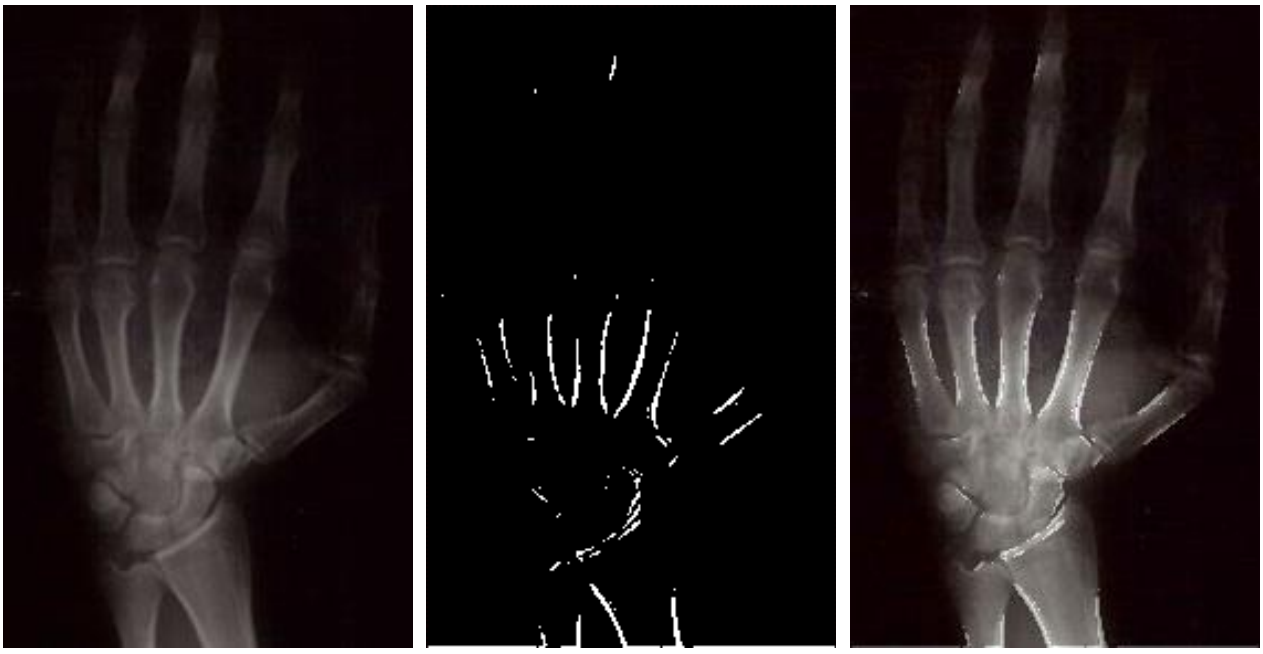


а) б)

в)

Рисунок 8 – Результат улучшения визуального качества изображения с помощью градиента Собеля: а) исходное изображение; б) градиент Собеля; в) градиент Собеля умноженный на исходное изображение

Выделение контуров с помощью градиента Робертса представлен на рисунке 9.



а) б) в)

Рисунок 9 – Результат улучшение визуального качества изображения с помощью градиента Робертса: а) исходное изображение; б) градиент Робертса; в) градиент Робертса наложенный на исходное изображение

Четвертая категория – пороговая фильтрация. Осуществляется по принципу замены центральной точки фильтра средним значением, если ее

величина превышает определенное наперед заданное значение. Значение порога может быть как константой, так и функционально зависимым от величины центральной точки.

В общем случае пороговое преобразование может рассматриваться как операция, при которой производится сравнение с функцией T, имеющей вид:

$$T = T(x, y, p(x,y), f), \tag{13}$$

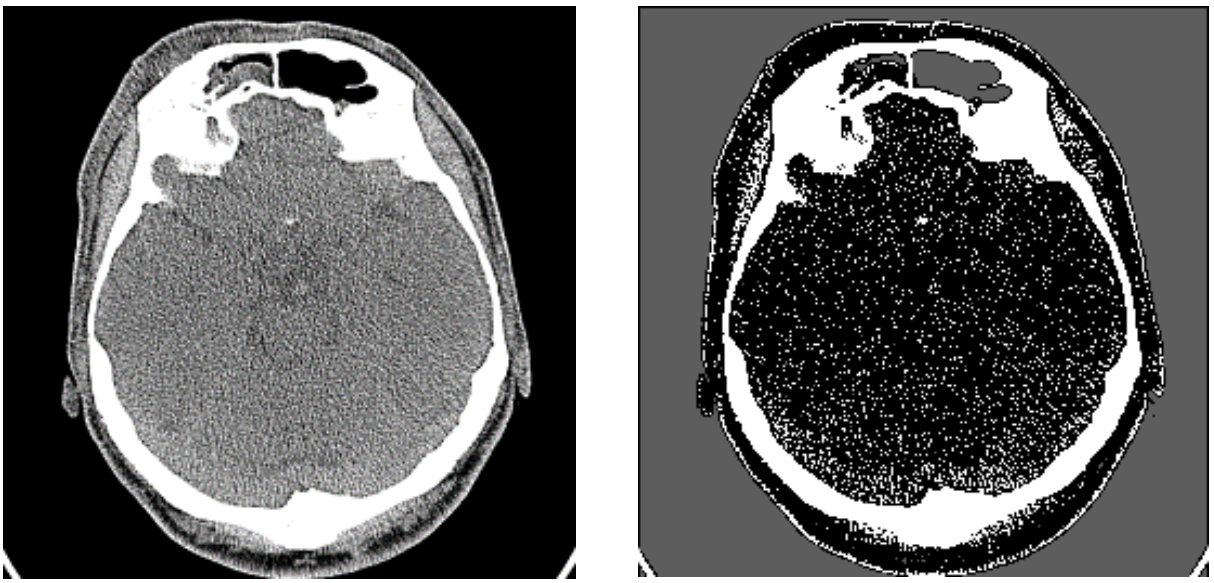
где f – некоторое изображение, содержащее светлые объекты на темном фоне, $p(x, y)$ обозначает некоторую локальную характеристику точки (x,y) изображения.

Для выбора порога используется метод, основанный на нахождении основных ее пиков путем сравнения локальных максимумов. Для этого определяется глобальный (в области высоких интенсивностей) максимум и k локальных максимумов M_k гистограммы. Затем для каждого локального максимума гистограммы, используя формулу (13) производился расчет коэффициентов α_k.

$$\alpha_k = \frac{M_k}{m_k + 1}, \tag{14}$$

где M_k – значение k-го локального максимума; m_k – значение локального минимума в диапазоне между глобальным максимумом и рассматриваемым локальным.

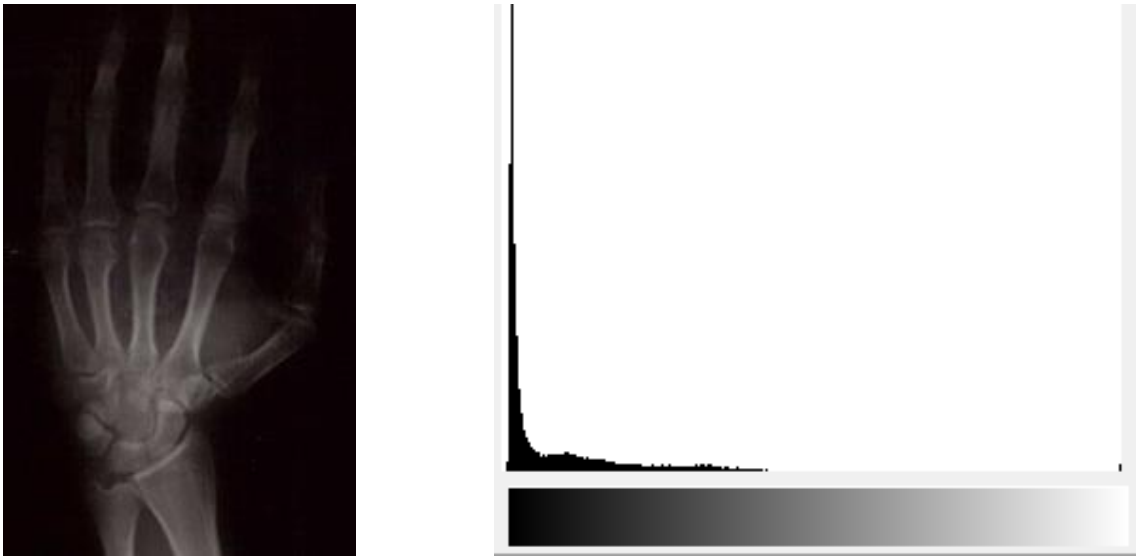
Тот из локальных максимумов, для которого значение α_k окажется наибольшим, выбирается в качестве второго основного максимума гистограммы. При этом порог бинаризации T выбирается равным уровню интенсивности, соответствующего минимуму m_k, расположенному между глобальным и найденным локальным максимумами [3]. Результат применения порогового фильтра представлен на рисунке 10 (значение порога 173).



а) б)

Рисунок 10 – Результат применения порогового фильтра: а) исходное изображение; б) результат применения порогового фильтра

Пороговое значение определяют по гистограмме изображения, которая представляет собой график статистического распределения элементов цифрового изображения с различной яркостью. Также по гистограмме изображения можно получить такую информацию о снимке, как правильность экспозиции при получении фотографии, контраст и цветовое насыщение снимка и т.д. Гистограмма рентгенограммы кисти руки представлена на рисунке 11.



а) б)

Рисунок 11 – Результат построения гистограммы: а) рентгенограмма кисти руки; б)гистограмма рентгенограммы

Одним из эффективных методов улучшения визуального качества изображений, таких как рентгенограмма, флюорограмма, ангиограмма, имеющих суженный диапазон яркостей, является эквализация гистограммы.

Эквализация гистограммы – преобразование гистограммы, в результате которой расширяется динамический диапазон изображения, что позволяет увидеть ранее не замеченные детали [10].

Данный процесс задается уравнением:

$$S_k = \sum_{j=0}^{k} p_r(r_j) = \sum_{j=0}^{k} \frac{n_j}{n} \quad (15)$$

Результат эквализации гистограммы исходного изображения (рисунок 11) представлен на рисунке 12.

а) б)

Рисунок 12 – Результат эквализации гистограммы: а) гистограмма преобразованная в результате эквализации; б) рентгенограмма после эквализации гистограммы

Морфологический фильтр используется для извлечения некоторых свойств изображения, полезных для его представления и описания (контуров, выпуклых оболочек) [14].

Морфологические операции, основными среди которых являются: AND (логическое умножение), OR (логическое сложение), NOT (отрицание), применяются в основном к двоичным изображениям. Любая другая логическая операция может быть получена путем комбинации этих операций (рисунок 13).

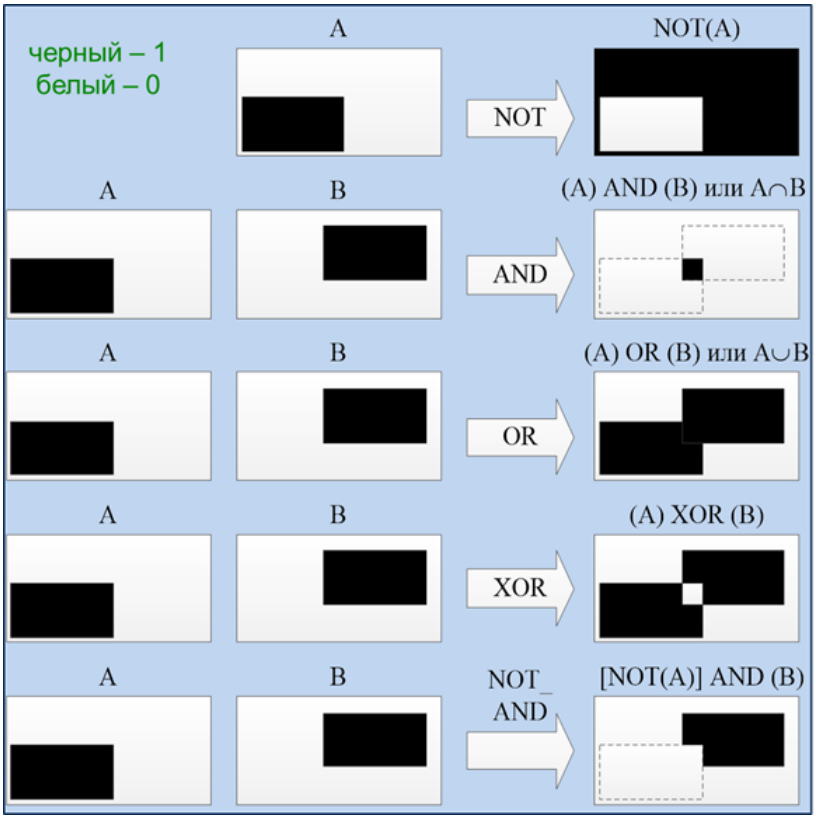


Рисунок 13– Логические операции над изображением

Логические операции выполняются поэлементно над пикселями двух и более изображений (за исключением операции NOT, применяемой к элементам одного изображения) [7].

Входными данными являются два изображения: обрабатываемое и специальное, зависящее от вида операции и решаемой задачи. Такое специальное изображения принято называть примитивом или структурным элементом.

Структурный элемент намного меньше обрабатываемого изображения и может быть любой формы

Результат морфологической обработки зависит как от размера и конфигурации исходного изображения, так и от структурного примитива. Наиболее часто используемые размеры примитива (3×3, 4×4 или 5×5 пикселей) представлены на рисунке 14.

Рисунок 14 – Наиболее распространенные формы структурного элемента

В результате морфологической обработки можно подчеркнуть или, наоборот, удалить детали определенной формы и размера на всем изображении.

Одно из основных преимуществ морфологической обработки – ее простота. В основе морфологической обработки изображений лежат следующие операции.

Дилатация (морфологическое расширение) – увеличивает область изображения, «наращивает» или «утолщает» объекты на двумерных изображениях. Используется примитив квадратной формы, где центром является ведущий элемент.

Дилатация множества A по множеству B обозначается A⊕B и определяется как:

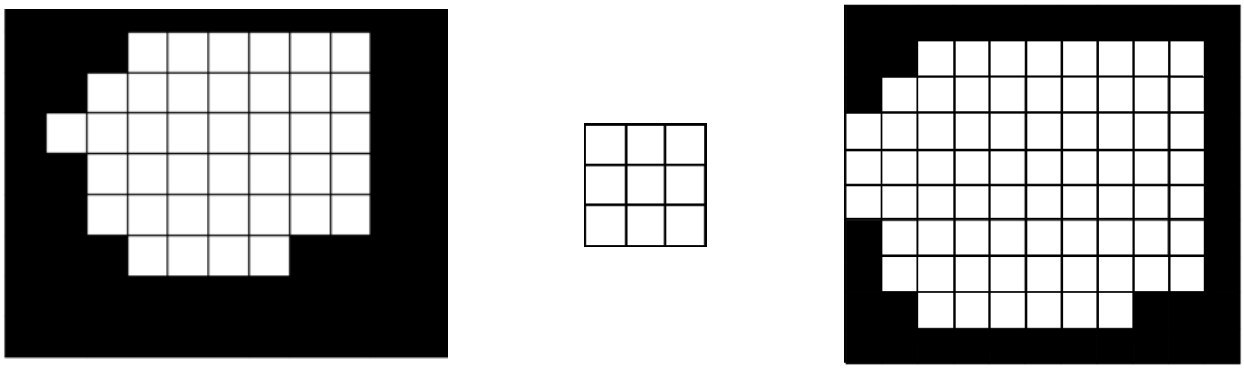
$$A \oplus B = (z|(B)_z \cap A \neq \emptyset), \qquad (16)$$

где A и B – множество из пространства Z^2, B – центральное отражение

$$B = \{w|w = -b, b \in B\}$$

Примитив применяется ко всем пикселям бинарного изображения. Каждый раз, когда ведущий элемент примитива совмещается с единичным бинарным пикселем, ко всему примитиву применяется перенос и последующее логическое сложение (OR) с соответствующими пикселями бинарного изображения.

Результат применения дилатации представлен на рисунке 15.



а) б) в)

Рисунок 15 – Результат применения дилатации: а) исходное изображение; б) примитив; в) результат применения дилатации

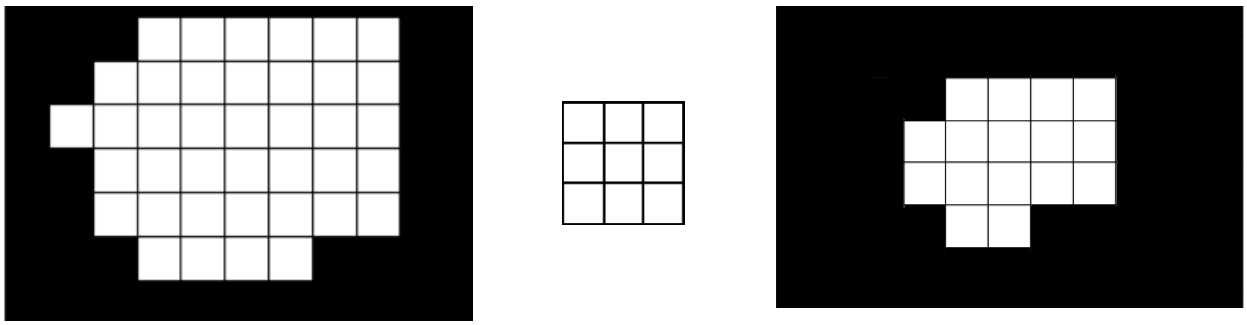
Эрозия (морфологическое сужение) – уменьшает область изображения, «ужимает» или «утончает» объекты на двумерных изображениях.

Эрозия множества A по множеству B обозначается A⊖B и определяется как:

$$A \ominus B = (z|(B)_z \subseteq A), \quad (17)$$

Эрозия множества A по B – это множество всех таких точек z, при сдвиге в которые множество B целиком содержится в A.

При выполнении операции эрозии примитив проходит по всем пикселам изображения. Если в некоторой позиции каждый единичный пиксель примитива совпадет с единичным пикселем бинарного изображения, то выполняется логическое умножение (AND) центрального пикселя примитива с соответствующим пикселем выходного изображения. Результат применения эрозии представлен на рисунке 16.

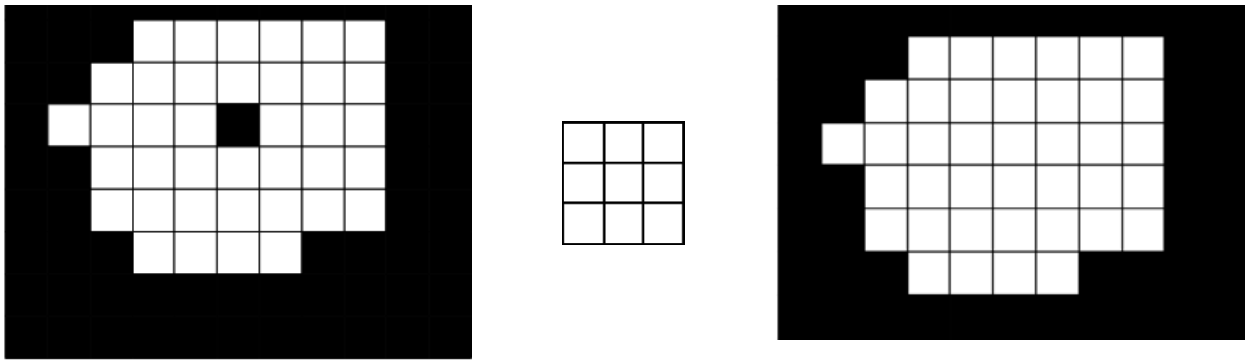


а) б) в)

Рисунок 16 – Результат применения эрозии: а) исходное изображение; б) примитив; в) результат применения эрозии

В результате применения эрозии все объекты, меньшие чем структурный элемент, стираются, объекты, соединённые тонкими линиями становятся разъединёнными и размеры всех объектов уменьшаются.

Замыкание – сглаживает контуры объекта, «заливает» узкие разрывы и длинные углубления малой ширины, ликвидирует небольшие отверстия и заполняет промежутки контура. Результат применения операции «замыкание» представлен на рисунке 17.



а) б) в)

Рисунок 17 – Результат применения операции «замыкание»: а) исходное изображение; б) примитив; в) результат применения операции «замыкание»

Замыкание множества A по примитиву B обозначается A•B и определяется как:

$$A \bullet B = (A \oplus B) \ominus B) \quad (18)$$

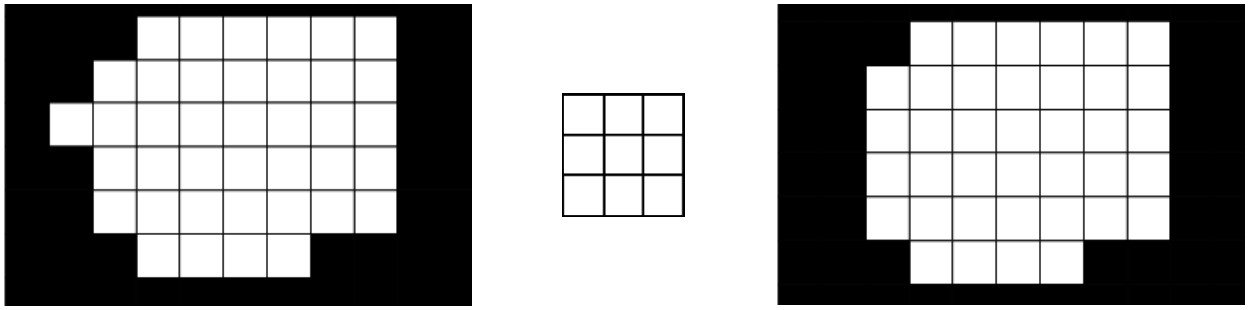
Если к изображению применить сначала операцию дилатации, то можно избавиться от малых дыр и щелей, но при этом произойдёт увеличение контура объекта. Избежать этого увеличения позволяет операция «эрозия», выполненная сразу после дилатации с тем же примитивом [8].

Размыкание – сглаживает контуры объекта, обрывает узкие перешейки и ликвидирует выступы небольшой ширины, помогает избавиться от маленьких фрагментов, выступающих наружу области вблизи ее границы.

Размыкание множества A по примитиву B обозначается A∘B и определяется как:

$$A \circ B = (A \ominus B) \oplus B), \quad (19)$$

Размыкание множества A по примитиву B строится как эрозия A по B, результат которой затем подвергается дилатации по тому же примитиву. Результат применения операции «размыкание» представлен на рисунке 18.



А б В

Рисунок 18 – Результат применения операции «размыкание»: а)исходное изображение; б)примитив; в) результат применения операции «размыкание»

Размыкание отсеивает все объекты, меньшие чем примитив, но при этом помогает избежать сильного уменьшения размера объектов. Также размыкание подходит для удаления линий, толщина которых меньше, чем диаметр

примитива. После операции размыкания контуры объектов становятся более гладкими.

1.1.3 Частотные методы обработки изображений

Методы обработки в частотной области основываются на модификации сигнала, формируемого путем применения к изображению преобразования Фурье в виде интеграла от синусов и/или косинусов, умноженных на некоторую весовую функцию. Функция, заданная рядом или преобразованием Фурье может быть полностью, без потери информации, восстановлена при помощи некоторой процедуры обращения [5].

Базовая модель фильтрации в частотной области задается равенством:

$$G(u,v)=H(u,v)\,F(u,v), \tag{20}$$

где $F(u,v)$ – Фурье-образ изображения, которое подлежит сглаживанию.

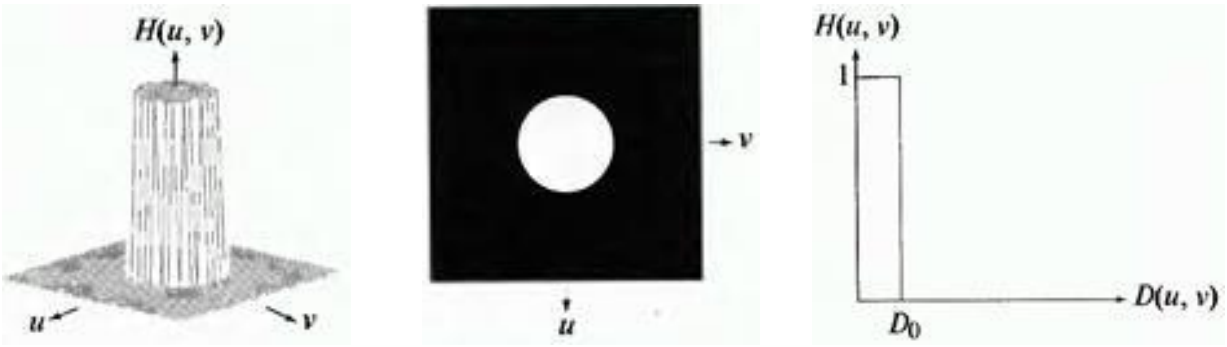
Цель состоит в выборе передаточной функции $H(u,v)$, которая ослабит высокочастотные компоненты $F(u,v)$ и сформирует функцию $G(u,v)$.

Фильтры нижних частот (ФНЧ). Данный тип покрывает диапазон очень резких фильтров до очень гладких: идеальный фильтр; фильтр Баттерворта; гауссов фильтр и т.д.

Идеальный фильтр низких частот обрезает все высокочастотные составляющие фурье-образа и имеет следующую передаточную функцию:

$$H(u,v)=\begin{cases}1 \text{ при } D(u,v)\le D_0\\ 0 \text{ при } D(u,v)>D_0\end{cases} \tag{21}$$

где D_0 – заданная неотрицательная величина; $D(u,v)$ – расстояние от точки (u,v) до начала координат (центра частотного прямоугольника). Передаточная функция ИФНЧ представлена на рисунке 19.



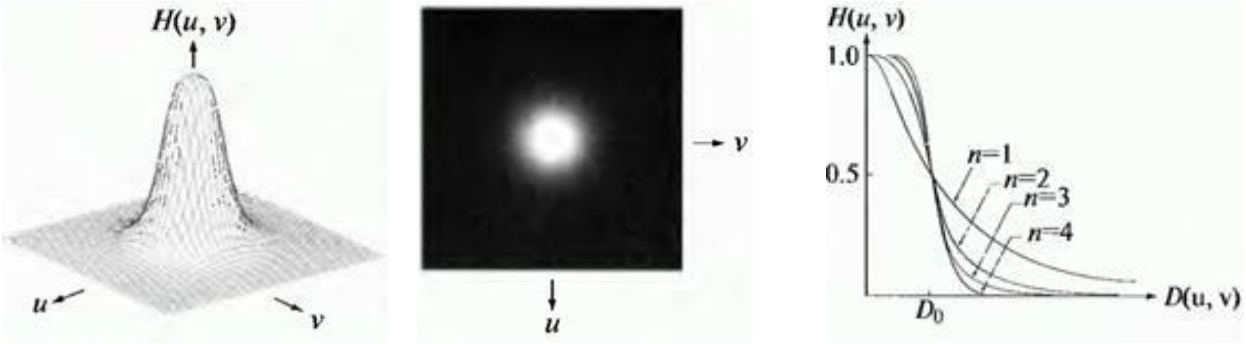
а) б) в)

Рисунок 19 – ИФНЧ: а) Изображение в перспективе графика передаточной функции ИФНЧ; б) Представление фильтра в виде изображения; в) радиальный профиль фильтра

Передаточная функция низкочастотного фильтра Баттерворта (БФНЧ) порядка n с частотой среза на расстоянии D_0 от начала координат задается выражением:

$$H(u,v) = \frac{1}{1 + \left(\frac{D(u,v)}{D_0}\right)^{2n}} \qquad (22)$$

В отличие от ИФНЧ, передаточная функция БФНЧ (рисунок 20) не имеет разрыва, который устанавливает точную границу между пропускаемыми и обрезаемыми частотами.



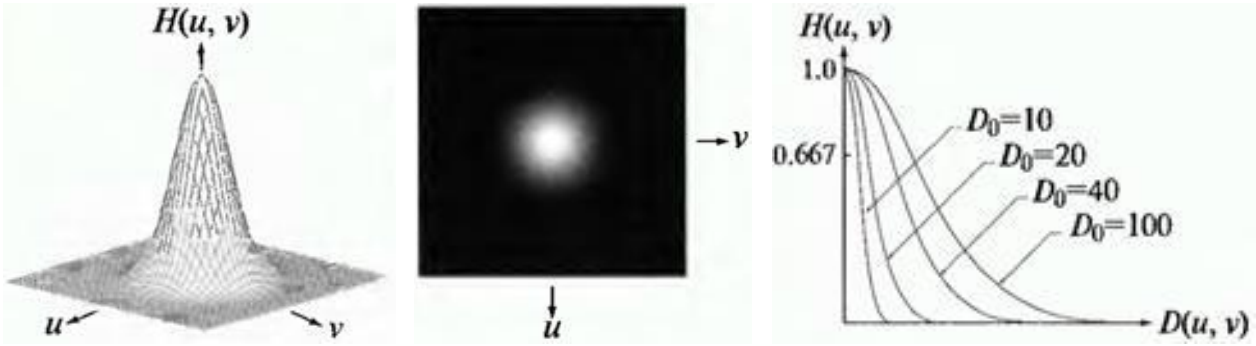
а) б) в)

Рисунок 20 – БФНЧ: а) перспективное изображение передаточной функции НЧ фильтра Баттерворта; б) полутоновое изображение фильтра; в) радиальные профили фильтров порядка от 1 до 4

Гауссовы фильтры низких частот (ГФНЧ) в двумерном случае задаются выражением:

$$H(u,v) = e^{\frac{-D^2(u,v)}{2D_0^2}} \tag{23}$$

Изображение может быть сглажено путем подавления высокочастотных составляющих его фурье-преобразования. Так как контуры и другие скачкообразные изменения яркости связаны с высокочастотными составляющими, повышение резкости изображения может быть достигнуто в частотной области при помощи процедуры высокочастотной фильтрации (ВЧФ), которая наоборот, подавляет низкочастотные составляющие и не затрагивает высокие частоты фурье-преобразования [2]. Передаточная функция ГФНЧ представлена на рисунке 21.



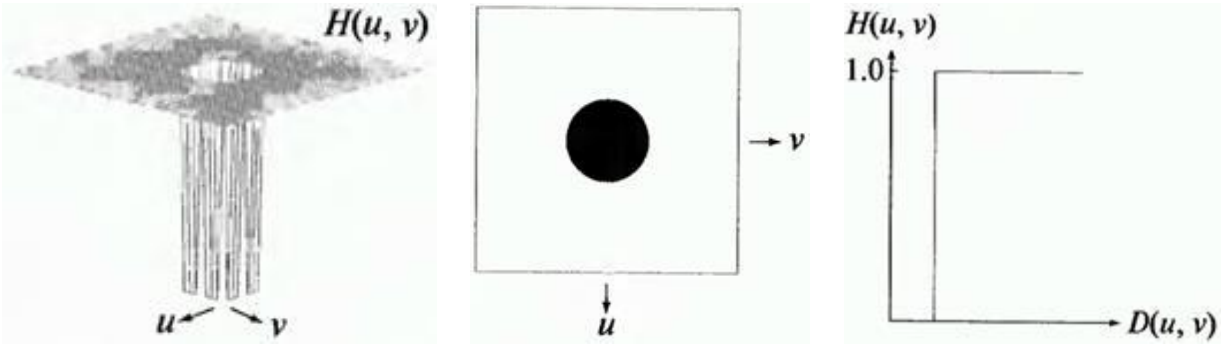
а) б) в)

Рисунок 21 – ГФНЧ: а) перспективное изображение передаточной функции ГФНЧ; б) полутоновое изображение фильтра; в) радиальные профили фильтров для различных значений D_0

Передаточная функция ВЧ фильтров может быть получена с помощью выражения:

$$H_{hp}(u,v)=1-H_{lp}(u,v) \tag{24}$$

Идеальный фильтр высоких частот обнуляет все частоты, попадающие внутрь круга радиуса частоты среза, одновременно пропуская без ослабления все частоты, лежащие вне круга. Передаточная функция ИФВЧ представлена на рисунке 22.

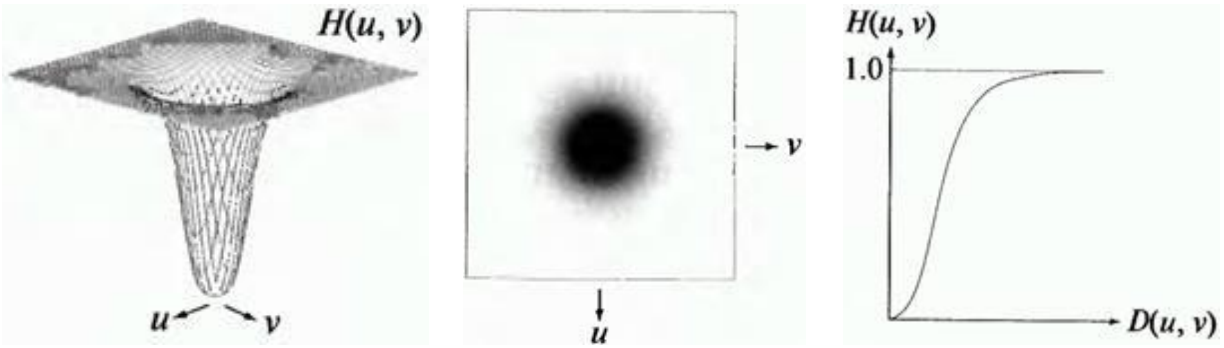


а) б) в)

Рисунок 22 – ИФВЧ: а) перспективное изображение передаточной функции ИФВЧ; б) полутоновое изображение фильтра; в) профиль ИФВЧ

Передаточная функция высокочастотного фильтра Баттерворта (БФВЧ) порядка n с частотой среза на расстоянии D_0 от начала координат представлена на рисунке 23 и задается выражением:

$$H(u,v) = \frac{1}{1 + \left(\frac{D_0}{D(u,v)}\right)^{2n}} \qquad (25)$$

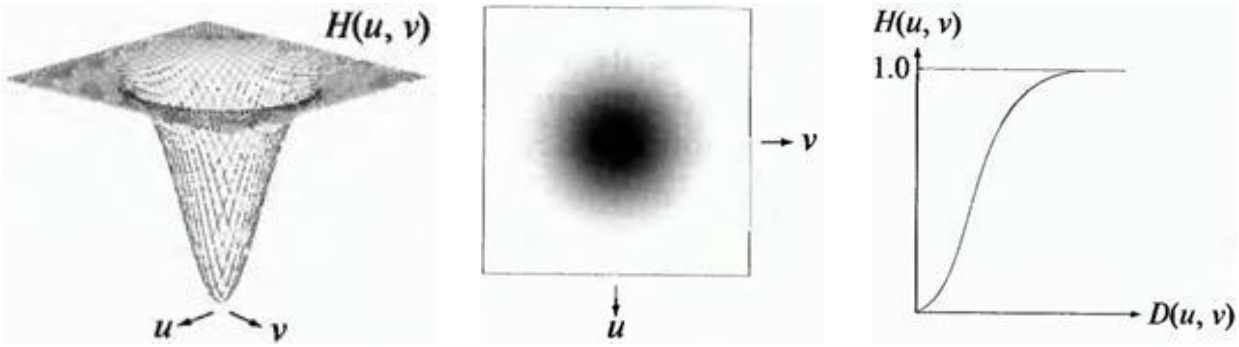


а) б) в)

Рисунок 23 – БФВЧ: а) перспективное изображение передаточной функции БФВЧ; б) полутоновое изображение фильтра; в) профиль БФВЧ

Передаточная функция гауссова фильтра высоких частот (ГФВЧ) с частотой среза, расположенной на расстоянии D_0 от начала координат представлена на рисунке 24 и задается выражением:

$$H(u,v) = 1 - e^{\frac{-D^2(u,v)}{2D_0^2}} \tag{26}$$



а) б) в)

Рисунок 24 – ГФВЧ: а) перспективное изображение передаточной функции ГФВЧ; б) полутоновое изображение фильтра; в) профиль ГФВЧ

Иногда при обработке изображения выгодно усилить его высокочастотную составляющую. В этом случае необходимо умножить передаточную функцию ВЧ фильтра на некоторую константу и добавить другую константу с тем, чтобы фильтрация не приводила к уничтожению нулевой частотной компоненты.

Такая процедура фильтрации называется фильтрацией с усилением высоких частот. Передаточная функция соответствующего фильтра задается выражением:

$$H_{hfe}(u,v) = a + bH_{hp}(u,v) \tag{27}$$

где $a \geq 0$ и $b > a$. Характерные значения a находятся в диапазоне от 0,25 до 0,50, а характерные значения b – в диапазоне от 1,5 до 2,5.

1.2 Особенности обработки медицинских изображений

Интерес к методам цифровой обработки медицинских изображений происходит из двух основных областей ее применения: повышение качества медицинских снимков для улучшения их визуального восприятия человеком и обработка изображений для их передачи и хранения [20].

Обработка медицинских снимков представляет собой процесс обнаружения некоторых элементов изображения, которых визуально трудно увидеть, не проводя различного рода манипуляции с медицинским снимком.

Как было выяснено в результате анализа, существует огромное количество методов обработки изображений, но лишь некоторые из них можно применить к медицинским снимкам [27]. Методы, которые применяются для обработки медицинских изображений не должны приводить к потере информации: данный фактор очень важен, так как именно благодаря качественному снимку врач будет способен установить более точный диагноз.

Проведя анализ рассмотренных методов обработки в пространственной и частотной областях с учетом требований к полному сохранению информации, можно сделать вывод, что следующие методы можно применять для улучшения визуального качества медицинских изображений и дальнейшего выделения интересующих объектов на этих изображениях.

Медианный фильтр – обладает свойствами подавления шума, при этом характеризуется очень малым эффектом расфокусировки.

Линейный фильтр – отлично справляется с подавлением шума, однако имеет небольшой эффект расфокусировки контуров изображения.

Морфологический фильтр – используется для извлечения некоторых свойств изображения, полезных для его представления и описания.

Градиент – используется для обнаружения дефектов на снимках, отлично справляются с задачей оконтурирования объектов, тем самым придавая изображению большую четкость.

Лапласиан – данный метод используется в обработке с целью выделения границ, при этом процесс происходит без потери необходимой для более точной постановки диагноза информации.

Степенное преобразование – также подходит для обработки медицинских снимков, так как без потерь и появления дефектов справляется с улучшением контраста изображения.

Эквализация гистограммы – данный процесс позволяет отобразить ранее не замеченные детали медицинского снимка, расширив его динамический диапазон.

Преобразование изображения в негатив – этот тип обработки применяется в случае необходимости усиления белых либо серых деталей изображения.

Пороговый фильтр – применяется на этапе сегментации изображений.

1.3 Постановка задачи проектирования

На сегодняшний день большинство изображений, в том числе медицинских, хранятся и используются в цифровом виде. При этом достаточно часто возникает такая проблема, как присутствие шумов, недостаточная четкость изображений, связанные с некорректной настройкой аппаратуры, наведенными помехами электросети и т.д. В настоящее время решение проблемы повышения визуального качества медицинских изображений без потери диагностической информации является актуальным.

Одним из решений является разработка графического редактора для обработки медицинских изображений. Такой графический редактор предоставит возможность медицинскому работнику без особых усилий и знания специфики методов обработки изображений визуально улучшить качество различных медицинских снимков. С целью облегчения реализации обработки, графический редактор включает только те из методов обработки, которые подходят для медицинских изображений. Это исключит вероятность потери диагностической информации вследствие выбора некорректного метода или метода, заведомо непригодного для обработки медицинских данных.

Вывод по первому разделу

В первом разделе был проведен анализ методов цифровой обработки изображений, выявлены особенности обработки медицинских изображений.

2 Проектирование графического редактора для обработки медицинских изображений

2.1 Разработка концептуальной модели

Для проектирования графического редактора медицинских изображений была создана концептуальная модель, описание которой начинается с построения контекстной диаграммы, представленной на рисунке 25 [1,12].

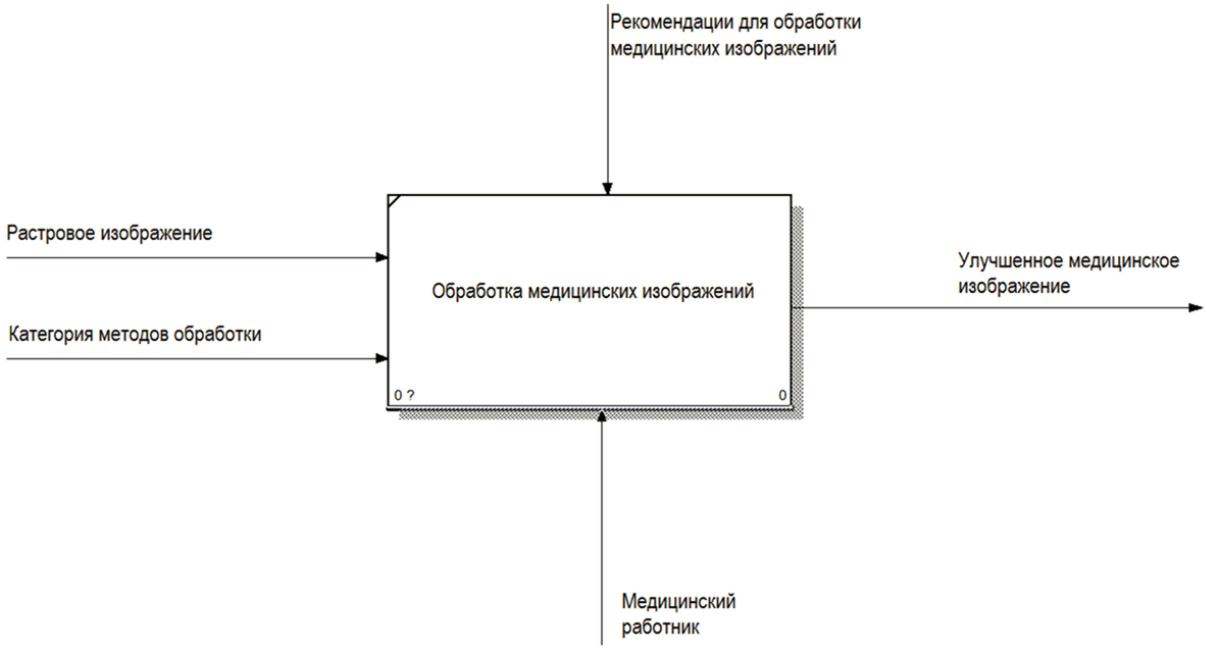


Рисунок 25 – Контекстная диаграмма системы обработки

Данная схема описывает процесс обработки медицинских изображений.

Первым входным параметром является цифровое растровое изображение, полученное в результате заранее проведенной диагностической процедуры на медицинском оборудовании. Изображение может быть сразу получено на медицинском оборудовании в цифровом виде или быть результатом оцифровки, например, пленочных рентгенограмм.

Вторым входным параметром является категория методов, которую выбирает медицинский работник перед проведением обработки изображения:

выделение границ, повышение контраста, удаление шумов и т.д. Категория методов обработки выбирается исходя из типа растрового изображения, а также целей, которые преследует медицинский работник.

Выходным параметром является улучшенное медицинское изображение, то есть обработанный снимок, который после оценки путем сравнения с исходным, оказывается тем, который требовался для постановки диагноза.

В качестве механизма управления выступает медицинский работник, руководствующийся рекомендациями для обработки медицинских изображений, находящиеся в инструкции пользователя.

Для более детального представления данного процесса была построена декомпозиция концептуальной схемы (рисунок 26).

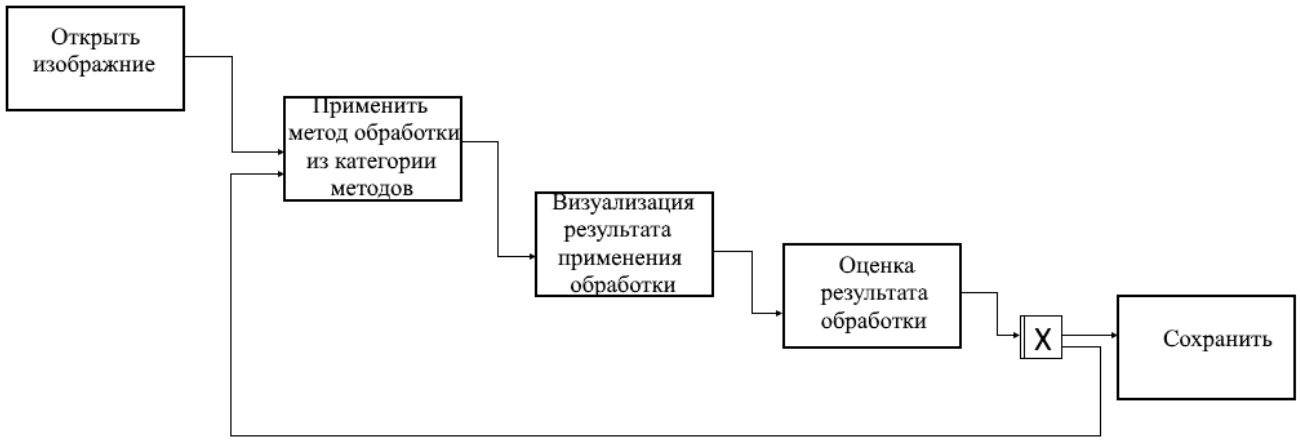


Рисунок 26 – Декомпозиция концептуальной диаграммы

В результате декомпозиции концептуальной диаграммы процесс был разбит на следующие этапы:

– открытие изображения;

– применение метода обработки из категории методов, которые представлены на UML-диаграмме использования (рисунок 27);

– визуализация результата применения обработки;

– оценка результата обработки;

– сохранение результата обработки изображения.

В процессе оценки результата обработки выполняется, во-первых, визуальное сравнение исходного изображения с обработанным снимком (субъективная оценка врача), во-вторых, вычисление пикового отношения сигнал-шум и среднеквадратичного отклонения данных обработанного снимка от исходного (объективные показатели оценки).

Следует учесть, что не всегда применение одного из методов обработки приведет к достижению требуемой цели, поэтому в схеме, представленной на рисунке 27 было реализовано два варианта продолжения работ после оценки [23].

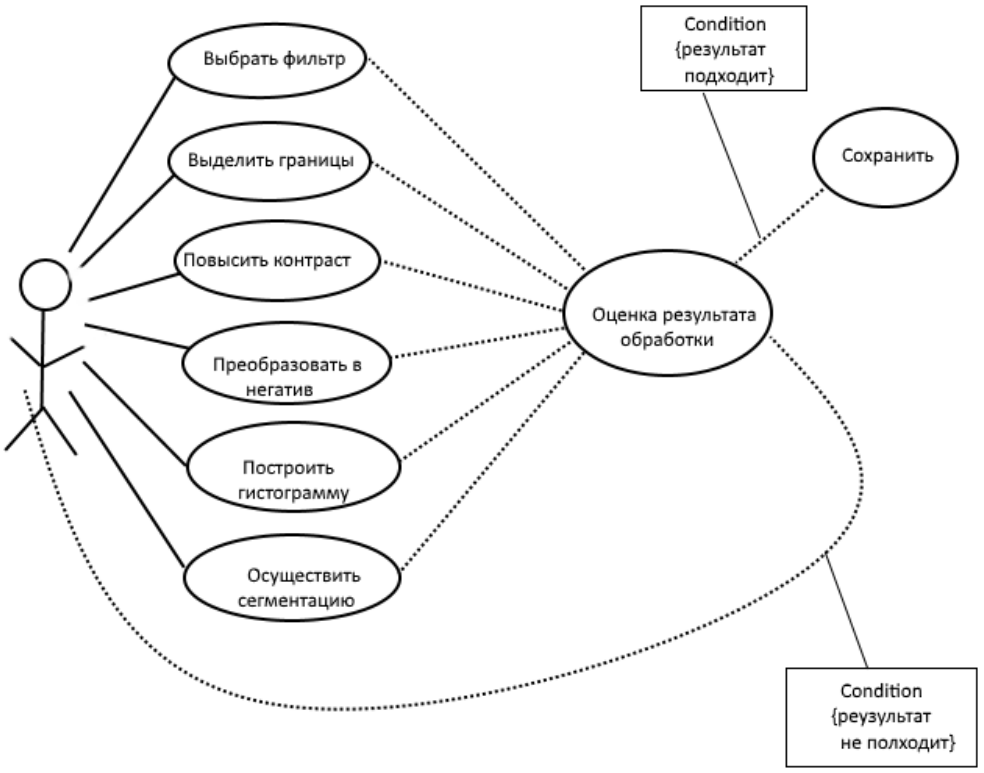


Рисунок 27 –UML-диаграмма вариантов использования

Первый представляет собой благополучное завершение работы со снимком и его последующее сохранение, а второй – требует применения

другого метода обработки вместо примененного либо еще дополнительного метода к уже примененному.

2.2 Разработка UML-диаграммы компонентов

С целью отображения структуры системы была разработана UML-диаграмма компонентов, которая позволяет определить архитектуру разрабатываемой системы. Данная схема представлена на рисунке 28.

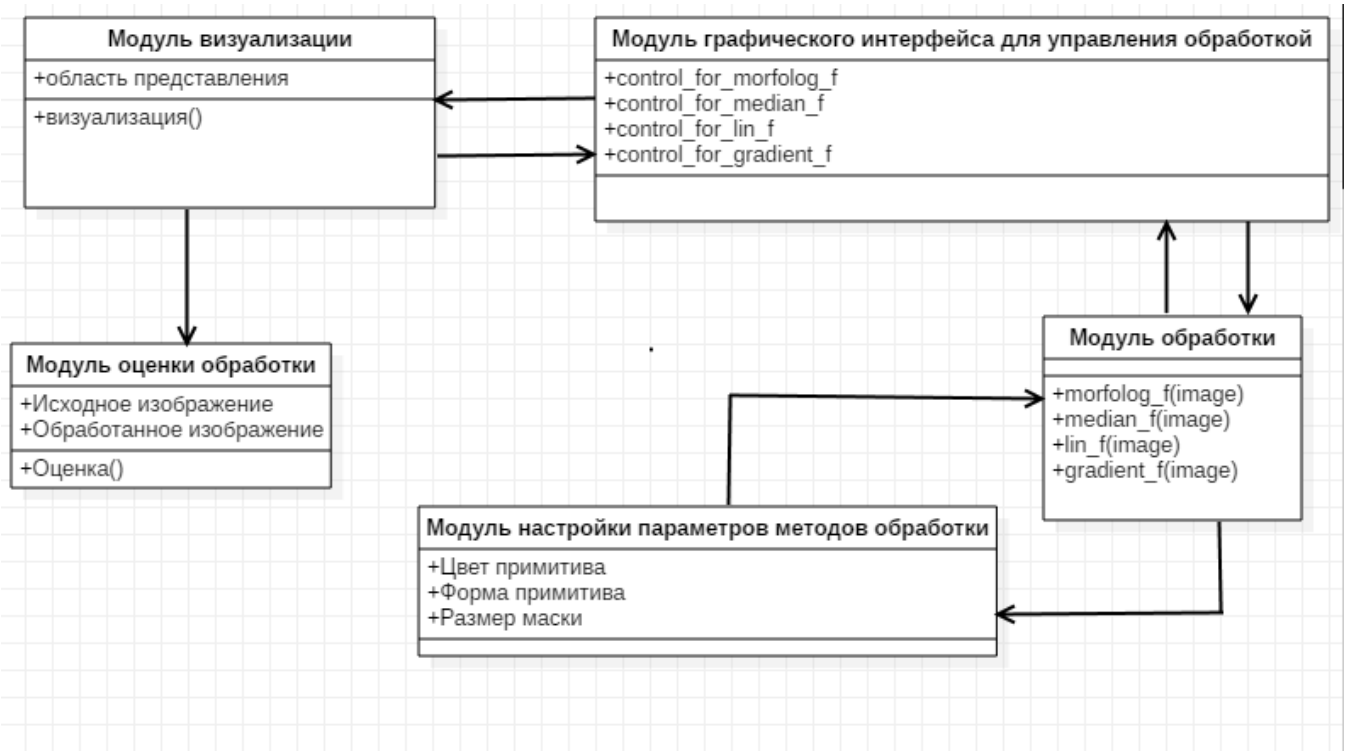


Рисунок 28 – UML-диаграмма компонентов

На данной диаграмме изображены пять компонентов, отвечающих за выполнение соответствующих функций. Некоторые модули имеют атрибуты, которые также носят название «глобальные переменные» и используются для обмена данными между внутренними процедурами

Модуль визуализации предназначен для отображения в окне графического редактора исходного растрового изображения, а также уже

обработанного снимка. Также данный модуль отвечает за визуализацию всех манипуляций в процессе редактирования.

Модуль обработки служит для проведения необходимых действий с медицинским изображением с целью его улучшения.

Следующий модуль необходим для настройки параметров методов обработки: выбор цвета примитива и его формы, размера маски, значений коэффициентов и т.д.

Модуль оценки обработки предназначен для проведения сравнения обработанного изображения с исходным.

Для разработки информационной модели системы целесообразно использовать нотацию DFD, которая способна продемонстрировать как каждый компонент преобразует свои входные данные в выходные, а также показать отношения между этими процессами [6].

Результат построения схемы потока данных, используя методологию DFD, представлен на рисунке 29.

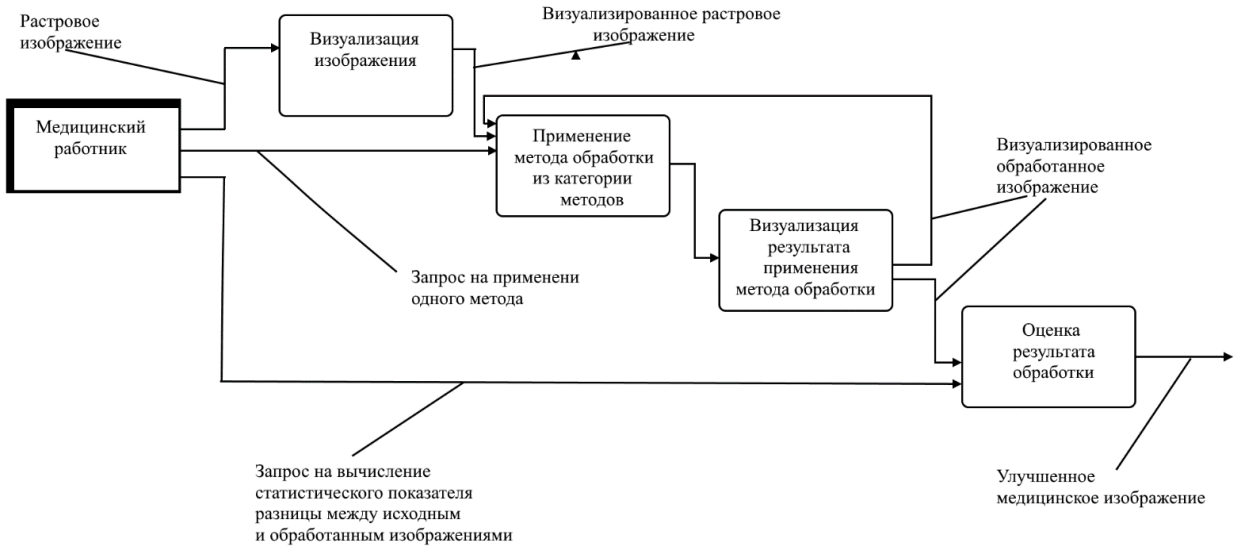


Рисунок 29 – DFD-диаграмма потока данных

Из диаграммы можно заметить, что медицинский работник, который является механизмом управления, представлен в виде блока сущности и

выполняет всю последовательность действий: открывает растровое изображение; применяет к визуализированному отображенному изображению метод обработки; визуализирует результат применения метода; проводит оценку результата обработки, в результате чего выходным параметром является улучшенное медицинское изображения, руководствуясь которым врач будет ставить диагноз.

2.3 Разработка графического интерфейса

Разрабатываемый графический интерфейс должен максимально реализовывать возможности редактора, но вместе с тем не перегружать пользователя обилием меню, кнопок, изображений и текста. Схемы графического интерфейса системы представлены на рисунке А.1 и рисунке А.2.

Для отображения реализации комплексных методов из пункта главного меню «Специальная обработка» были разработаны алгоритмы, которые изображены на рисунке А.3.

Вывод по второму разделу

Во втором разделе была создана концептуальная модель графического редактора, контекстная диаграмма и ее декомпозиция. Также был создан графический интерфейс редактора для улучшения визуального качества медицинских снимков.

3 Программная реализация графического редактора для медицинских изображений

3.1 Программная реализация графического редактора

Графический редактор был реализован в среде программирования C++ Builder на языке программирования C++.

При запуске графического редактора врач увидит окно, которое представлено на рисунке 30.

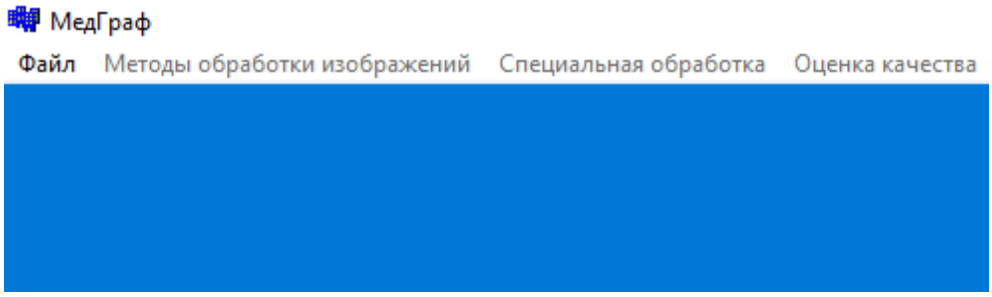


Рисунок 30 – Главное окно графического редактора

Главное меню содержит такие пункты, как «Файл», «Методы обработки изображений», «Специальная обработка», а также «Оценка качества».

Для начала обработки изображения врачу необходимо его открыть. Данную процедуру он сможет выполнить, выбрав пункт меню «Открыть», находящийся в пункте главного меню «Файл» (рисунок 31).

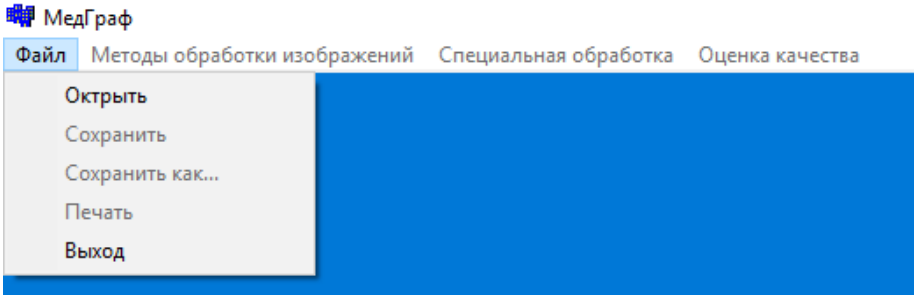


Рисунок 31 – Главное окно графического редактора

Также пункт «Файл» содержит такие подпункты: сохранить (данная функция отвечает за сохранение обработанного изображения, при этом заменяя

исходное); сохранить как (позволяет сохранить улучшенный снимок в формате bmp с имененем, заданным пользователем в любом для себя удобном месте); печать (обеспечивает печать обработанного снимка в случае необходимости); выход (при нажатии этой кнопки графический редактор остановит свою работу в случае выполнения каких-либо операций и закроется).

Следующий пункт главного меню дает возможность врачу выбрать метод, которым он будет пользоваться при обработке. Для удобства использования редактора медицинским персоналом, методы обработки были разделены на следующие категории действия: «Фильтры»; «Выделение границ»; «Повышение контраста»; «Негатив»; «Пороговый фильтр» (рисунок 32).

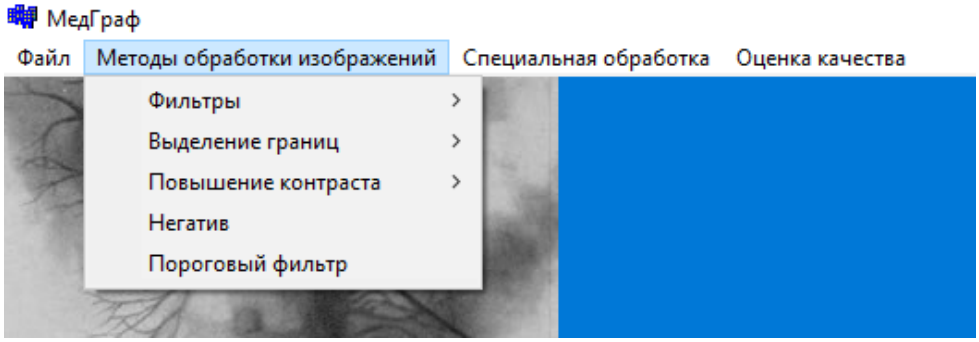


Рисунок 32 – Главное окно графического редактора

Следует отметить то, что для отображения составляющих какого либо пункта меню, достаточно навести на него стрелкой мыши.

Некоторые из перечисленных категорий делятся на подпункты. Например, категория «Фильтры» состоит из таких фильтров, как «Медианный», «Линейный», а также «Морфологический», который в свою очередь состоит из процессов: «Эрозия», «Дилатация», «Замыкание», «Размыкание» (рисунок 33).

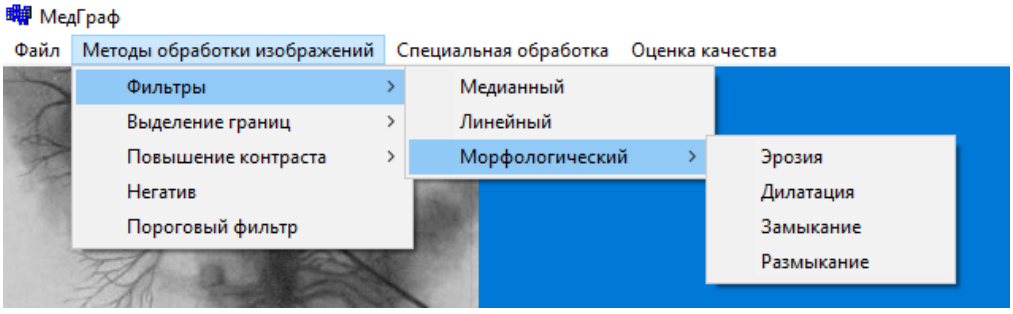


Рисунок 33 – Главное окно графического редактора

Категория «Выделение границ» состоит из таких процессов, как «Морфологический фильтр», «Лапласиан» и «Градиент», содержащий «Градиент Робертса», «Градиент Собеля» и «Градиент Щарра» (рисунок 34).

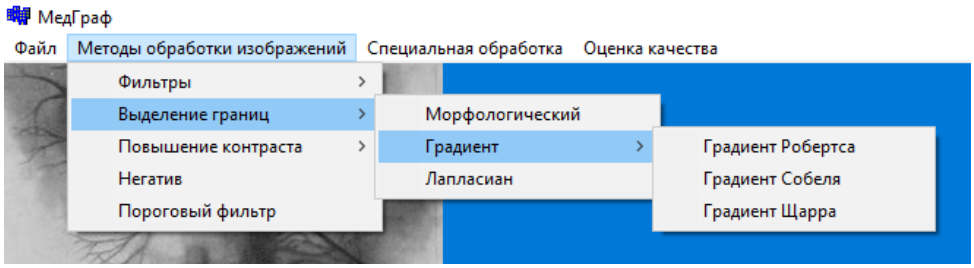


Рисунок 34 – Главное окно графического редактора

Категория «Повышение контраста» включает в себя «Степенное преобразование» и «Эквализацию гистограммы» (рисунок 35).

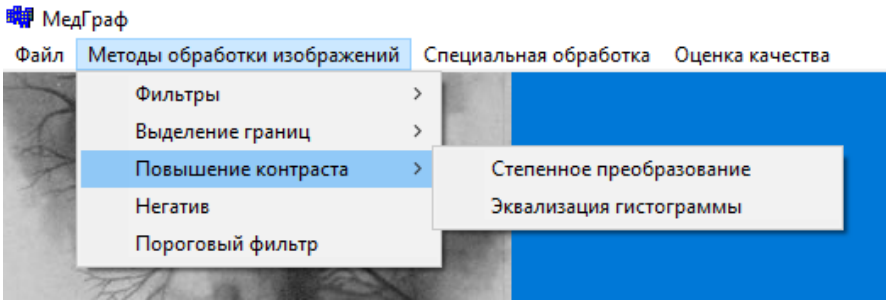


Рисунок 35 – Главное окно графического редактора

Следующий пункт главного меню – «Специальная обработка». Предоставляет возможность специалисту применить комплекс методов для улучшения медицинского снимка. Данный пункт содержит: «Улучшение качеста РГ (оператор Щарра)»; «Улучшение качества РГ (оператор Собеля)»; «Выделение констных структур на КТ» (рисунок 36).

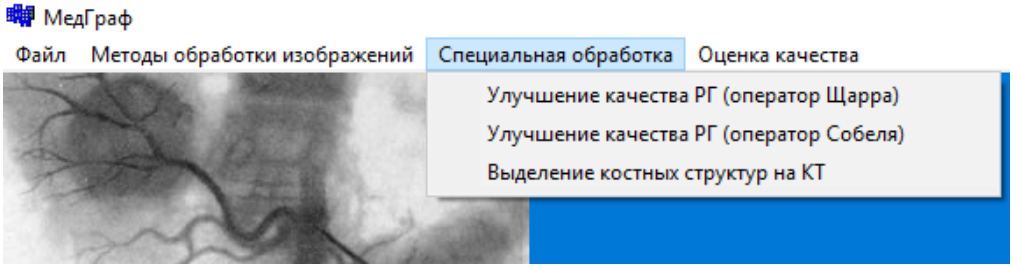


Рисунок 36 – Главное окно графического редактора

Последний пункт главного меню носит название «Оценка качества» и состоит из таких подпунктов, как «PSNR» и «СКО» (рисунок 37).

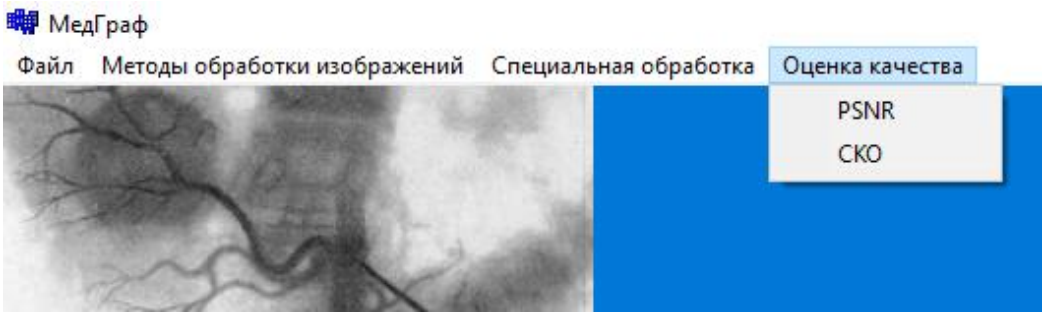


Рисунок 37 – Главное окно графического редактора

После открытия снимка врач видит перед собой главное окно, пример которого представлен на рисунке 38.

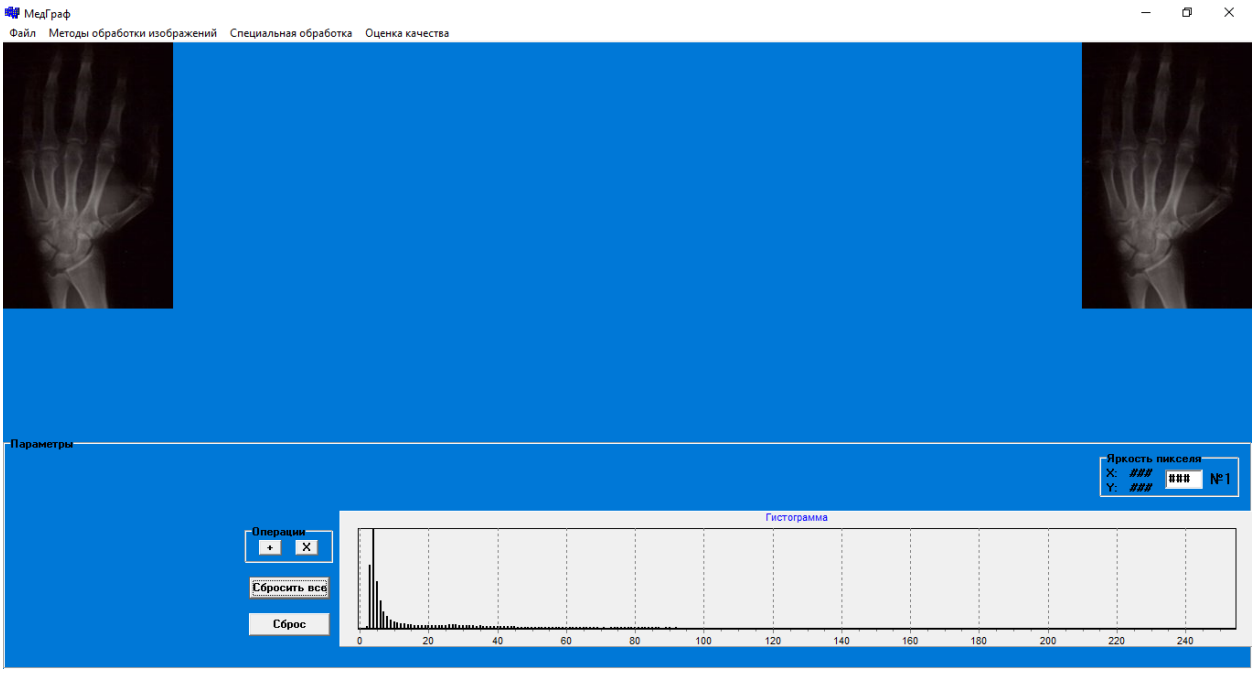


Рисунок 38 – Главное окно графического редактора

Из рисунка 38 можно заметить, что главное окно разделено горизонтальной полосой на две составляющих. Верхняя часть содержит окно с исходным изображением (находится в левом верхнем углу главного окна), а также окно, в котором будет отображаться обработанный снимок (правый верхний угол главного окна). Нижняя часть, в свою очередь, отображает

некоторые параметры, такие как «Яркость пикселя» (рисунок 39), а также параметры, которые необходимы для реализации определенных методов (ячейки для заполнения параметров появляются в левом нижнем углу автоматически при выборе соответствующего метода).

а) б)

Рисунок 39 – Значение яркости пикселя: а) исходного изображения; б) обработанного изображения

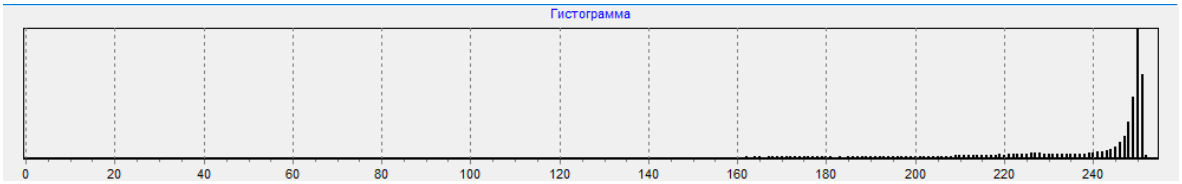
На рисунке 39,а представлены параметры выбранного пикселя: X и Y обозначают положение пикселя на изображении, цифра 13 в данном случае отображает значение яркости пикселя, на который кликом левой кнопки мыши указал пользователь, а №1 значит то, что рассматриваемый пиксель находится на первом изображении, то есть исходном.

В случае рассмотрения пикселя на обработанном изображении номер изменится на 2 (рисунок 39,б).

Также в главном окне автоматически строится гистограмма открытого изображения, представляющая собой график распределения элементов цифрового изображения (пикселей), в котором по горизонтальной оси отображается яркость пикселей, а по вертикальной – относительное количество этих пикселей с соответствующим значением яркости.

После применения какого-либо метода, гистограмма исходного изображения заменяется гистограммой обработанного (рисунок 40). Фрагмент программы построения гистограммы приведен в листинге 1 приложения Б.

а)



б)

Рисунок 40 – Гистограмма: а) исходного изображения; б) после применения негатива

Одной из дополнительных функций графического редактора является возможность наложения исходного изображения на обработанное (рисунок 41).

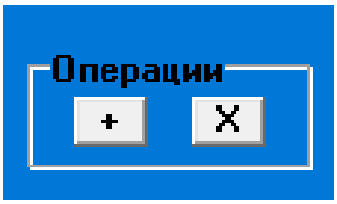


Рисунок 41 – Операции наложения снимков друг на друга

Наложение осуществляется либо путем попиксельного сложения, либо путем попиксельного умножения изображений, чем достигается усиление эффекта обработки. Например, выделением границ градиентными методами с последующим сложением с исходным изображением достигается усиление контраста и границ на рентегнограмме. Программный код данной функци приведен в листинге 2 приложения Б.

В случае, если врача не устроит результат, полученный после обработки выбранным им методом, пользователю предоставляется возможность сбросить крайний метод (кнопка «Сброс») или же полностью отменить все применяемые методы (кнопка «Сбросить все») (рисунок 42).

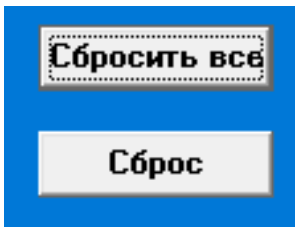
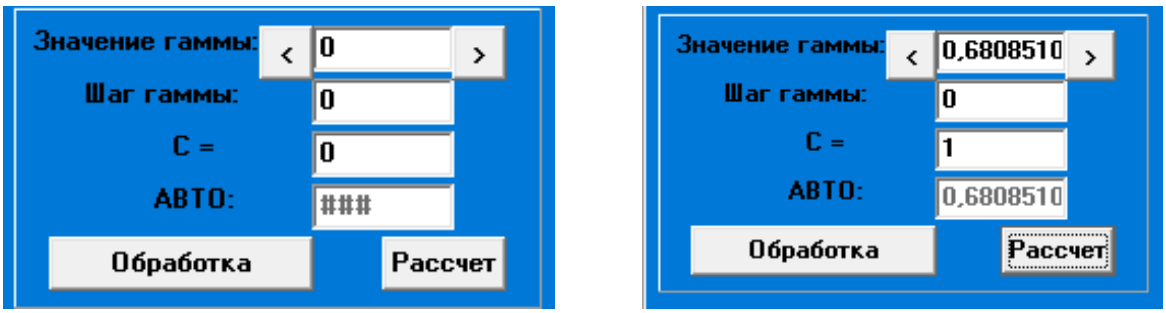


Рисунок 42 – Кнопка сброса фильтра

Еще одной дополнительной возможностью графического редактора является то, что на любом этапе работы врачу предоставляется возможность открыть сразу два снимка, с которыми он будет проводить различного рода манипуляции. Для этого ему рекомендуется нажать клавишу «пробел» на клавиатуре и выбрать второй необходимый для последующей работы снимок. После чего загруженный файл откроется во втором окне, где обычно отображается обработанное медицинское изображение. Данная функция полезна в том случае, когда медицинскому работнику требуется, например, наложить два изображения друг на друга.

С целью демонстрации работоспособности графического редактора были обработаны некоторые снимки с применением различных методов.

Итак, для улучшения качества рентгенограммы кисти руки, было проведено «Степенное преобразование» из категории «Повышение контраста». Подпрограмма, реализующая данную обработку приведена в листинге 3 приложения Б. Перед запуском данного процесса , как упоминалось ранее, врачу предлагается два варианта заполнения необходимых параметров: ввести некоторые данные вручную или же использовать функцию автоматического рассчета, программная реализация которой приведена в листинге 4 приложения Б. Данная функция была создана с целью облегчения и ускорения применения данного метода (рисунок 43). При автоматическом рассчете значение параметра гамма будет рассчитано по формуле (7) после чего результат сразу же отобразиться в поле «Значение гаммы».



а) б)

Рисунок 43 – Настройка параметров степенного преобразования: а) начальное окно; б) заполненное окно с автоматическим расчетом параметра гамма

Результат степенного преобразования представлен на рисунке 44.

Рисунок 44– Результат применения степенного преобразования

Еще одним из методов обработки медицинских изображений, с целью повышения контраста, является «Эквализация гистограммы». Данный метод не требует ручного рассчета и ввода каких-либо дополнительный параметров. Результат применения «Эквализации гистограммы» представлен на рисунке 45. Код программы приведен в листинге 5 приложения Б.

Рисунок 45 – Результат применения эквализации гистограммы

Категория «Фильтры», как упоминалось ранее, состоит из линейного, медианного и морфологических фильтров, программная реализация которых приведена в приложении Б.

При выборе линейного или медианного фильтров врачу предлагается определить размер маски (линейный: 3x3,4x4, 5x5; медианный: 3x3, 5x5). Результат применения медианного фильтра представлен на рисунке 46.

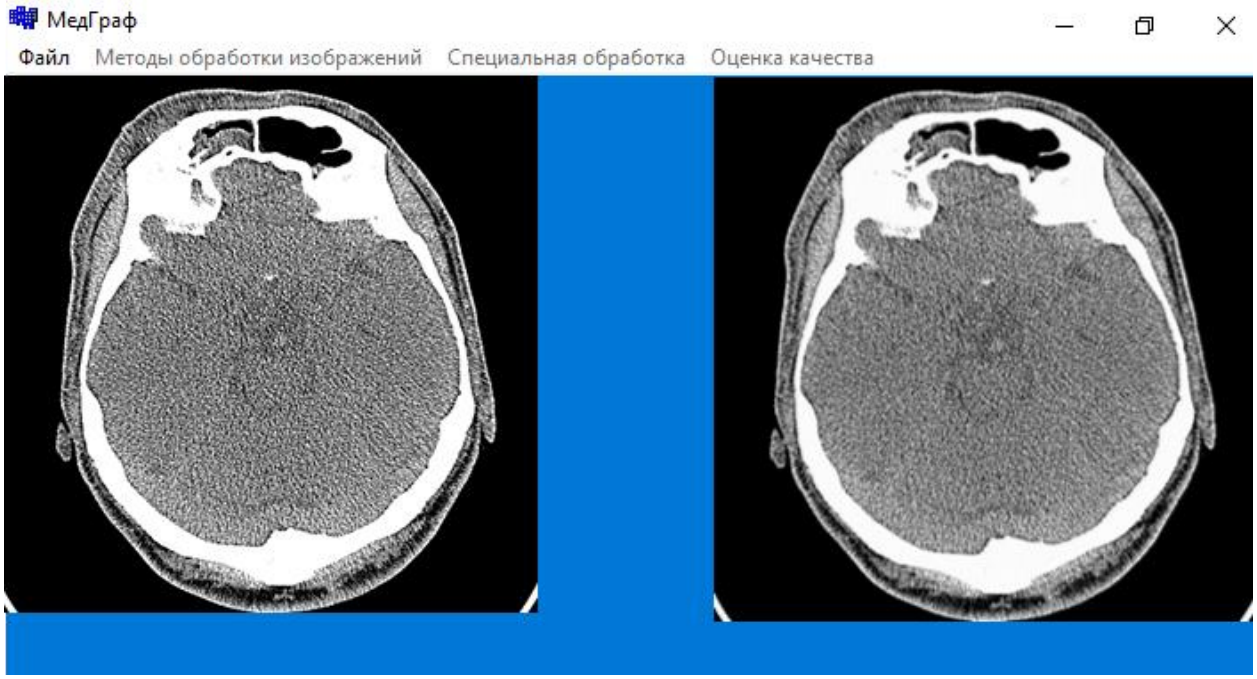


Рисунок 46 – Результат применения медианный фильтра

При выборе морфологического фильтра, необходимо выбрать метод математической морфологии, среди которых дилатация, эрозия, замыкание, размыкание. Фрагменты реализации данных методов приведены в приложении Б. После выбора фильтра необходимо определиться с формой и цветом примитива (рисунок 47).

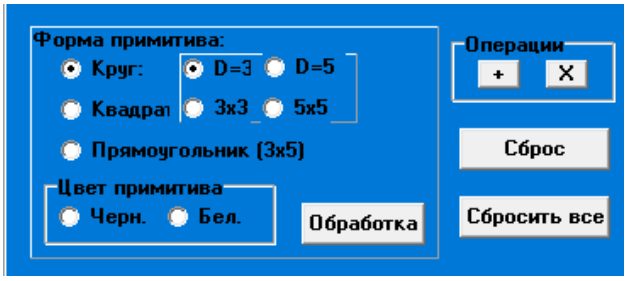


Рисунок 47 – Настройки параметров для морфологической обработки

Также необходимо учесть, что реализация морфологической обработки в графическом редакторе осуществляется только после применения порогового фильтра. Данная обработка используется для удаления артефактов после пороговой обработки для выделения костных структур. Результат работы морфологии изображен на рисунке 48.

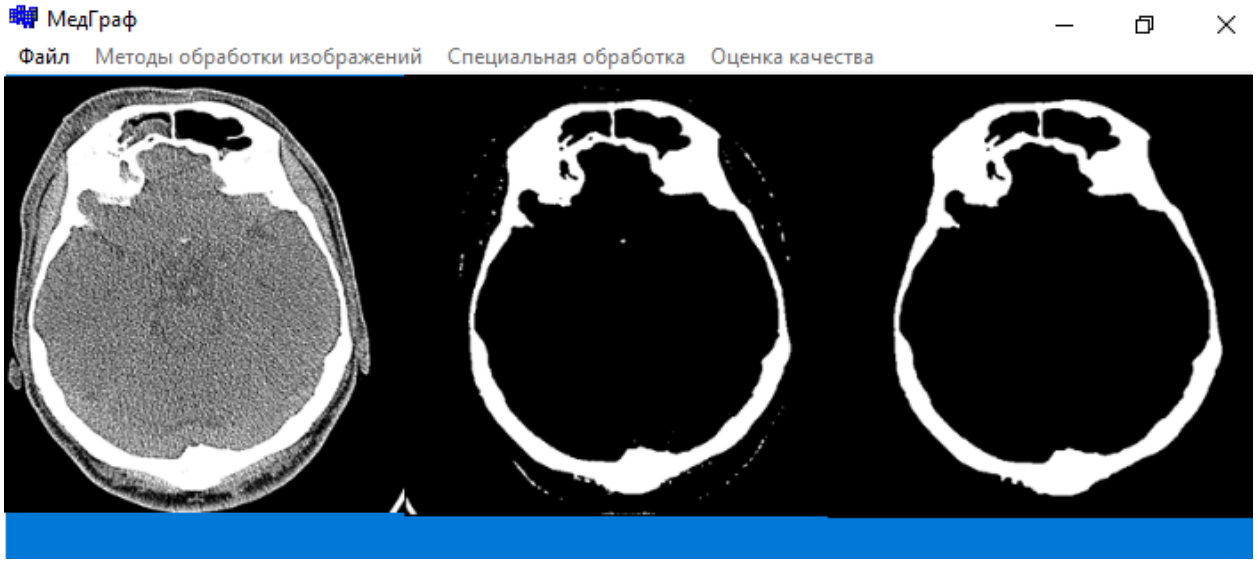


Рисунок 48 – Результат применения морфологического фильтра

При необходимости выделения границ на медицинском изображении рекомендуется использовать такие фильтры, как градиент Щарра, Собеля,

Робертса, лапласиан, морфологический, программная реализация которых приведена в приложении Б. Результат применения лапласиана представлен на рисунке 49.

Рисунок 49 – Результат применения лапласиана

С целью облегчения работы врачу были реализованы комплексные методы обработки наиболее распространенных медицинских изображений: рентгенограммы и томограммы.

Комплексный метод для улучшения визуального качества рентгенограмм представляет собой подбор фильтров и их параметров, которые запускаются последовательно в автоматическом режиме при нажатии на соответствующий пункт меню.

Графические схемы алгоритмов работы комплексного методы повышения визуального качества рентгенограмм приведена на рисунке А.3,а в случае использования одного из методов в комплексе – градиента Собеля, и на рисунке А.3,б – в случае использования градиента Щарра.

Текст программы, реализующей данные комплексы, приведен в листинге 16 приложения Б. Результат применения комплексного метода для улучшения визуального качества рентгенограмм представлен на рисунке 50.

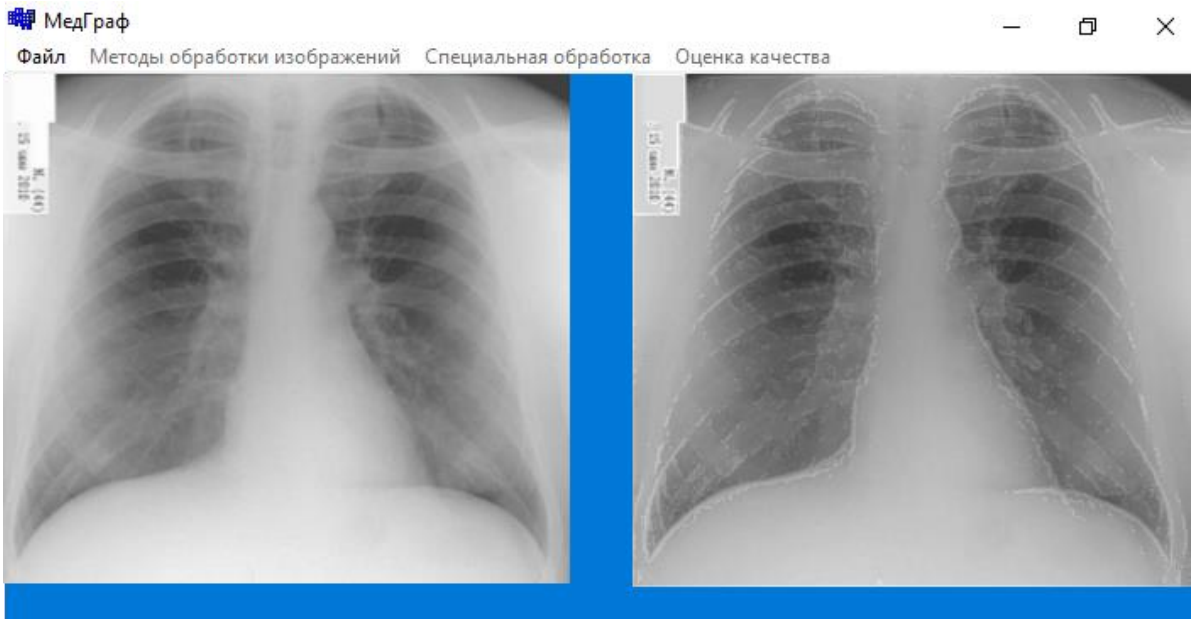


Рисунок 50 – Результат применения комплексной обработки для повышения визуального качества рентгенограмм

Результат применения комплексного метода с целью выделения костных структур на КТ представлен на рисунке 51. Для удобства печати результат работы взят в негативе.

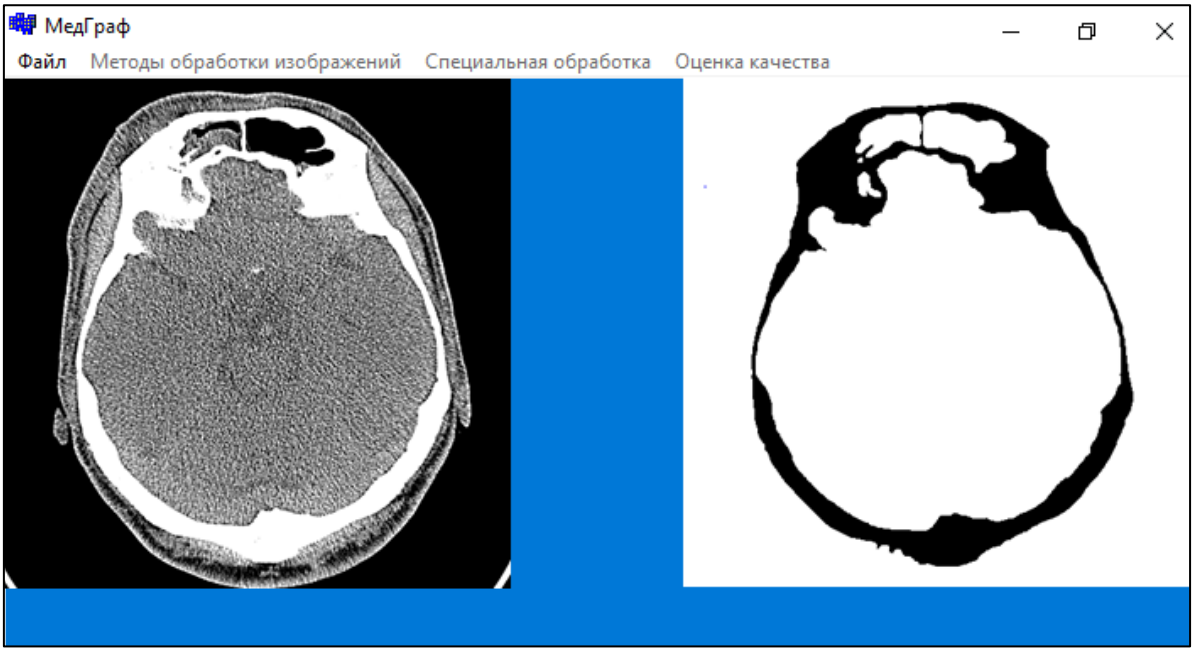


Рисунок 51 – Результат применения комплексной обработки для выделения костных структур на КТ-снимках

Комплексный метод выделения костных структур на КТ-данных включает в себя ряд методов и их параметров, позволяющих в автоматическом режиме удалить с томограммы все данные, не являющиеся костными структурами. Графическая схема алгоритма работы данного комплекса приведена на рисунке А.3,в. Текст программы представлен в листинге 17 приложения Б.

3.2 Оценка качества

С целью объективной оценки качества применения методов повышения визуального качества медицинских изображений были использованы следующие критерии: PSNR и СКО.

Среднеквадратичное отклонение (СКО) определяется по формуле:

$$CKO = \sqrt{\frac{1}{N * M} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (\tilde{f}(x,y) - f(x,y))^2}, \quad (28)$$

где$f(x,y)$ – входное изображение размером M×N; $\tilde{f}(x,y)$ – обработанное изображение.

Пиковое отношение сигнала к шуму (PSNR) определяется по формуле:

$$PSNR = 10 \lg\left(\frac{G^2}{CKO}\right) = 20 \lg\left(\frac{G}{\sqrt{CKO}}\right), \quad (29)$$

где G – максимальное значение яркости пикселей изображения; СКО – среднеквадратическое отклонение разности выходного и входного изображений. Типичное значение параметра $PSNR$ для обработанного изображения, при котором визуально не воспринимается потеря информации составляет 30-40дБ.

Текст программы, реализующей расчет объективных критериев качества обработки изображений представлен в листинге 18 приложения Б.

Данные критерии целесообразно применять только к методам фильтрации, повышающим визуальное качество изображений, такие как медианный фильтр, степенное преобразование, эквализация гистограммы, комплексный метод для повышения визуального качества рентгенограмм.

Для методов, с помощью которых производится сегментация изображения, выделение костных структур и т.д. применение таких критериев не имеет смысла, так как в этом случае происходит кардинальное изменение изображений. Результаты оценки качества представлены в таблице 1.

Таблица 1 – Результаты оценки качества обработки

Название метода	Значение критерия PSNR
комплексный метод для повышения визуального качества РГ	32.39
медианный фильтр	35.16
степенное преобразование	33.74
эквализация гистограммы	36.12

В результате проведения оценки качества можно заметить, что полученные цифры полностью соответствуют ранее установленным нормам, а именно 30-40 дБ. Соответственно можно предполагать, что в дальнейшем это обеспечит обработку медицинских изображений, при которой визуально не будет восприниматься потеря информации.

3.3 Технические требования

Для оптимального использования разработанного графического редактора рекомендуется следующая конфигурация компьютера:

- процессор с частотой не менее 1 ГГц или система на кристалле SoC ;

- ОЗУ – 1 ГБ для 32-разрядных систем или 2 ГБ для 64-разрядных систем;

- жесткий диск – 1 ГБ для 32-разрядных систем или 1 ГБ для 64-разрядных систем;

- видеоадаптер – DirectX 9 или более поздней версии с драйвером WDDM 1.0.

3.4 SWOT-анализ

Таблица 2 – SWOT-анализ

Сильные стороны	**Слабые стороны**
а) Наличие только тех методов, которые способны обрабатывать медицинские изображения без потери необходимой информации б) Наличие комплексных методов обработки для конкретных медицинских изображений в) Разделение методов обработки на категории по действию г) Возможность проведения оценки качества д) Возможность автоматического рассчета необходимых параметров	а) Отсутствие распознавания медицинских изображений б) Узкий выбор комплексных методов в) Отсутствие взаимодействия с некоторыми форматами медицинских изображений
Возможности	**Угрозы**
а) Возможность добавления комплексных методов может увеличить потребность в использовании графического редактора для медицинских изображений б) Наличие шумов и неблагоприятных артефактов на изображении вызывает потребность в использовании графического редактора для более точной постановки диагноза.	а) Появление ложных контуров и других артефактов при неправильном подборе параметров в ручном режиме

В таблице 2 приведен результат проведения swot-анализа.

SWOT-анализ показывает слабые и сильные стороны проекта, как с внутренней точки зрения, так и с внешней.

Сильные и слабые стороны представляют собой внутренние факторы, находящиеся под контролем разработчика, которые можно изменить с течением времени.

Возможности и угрозы являются внешними факторами, которые находятся вне контроля разработчика. Их можно попробовать спланировать и повлиять на положительные изменения и дальнейший ход событий, но это не гарантирует тотального контроля над ситуацией.

Сильные и слабые стороны позволяют увидеть настоящее течение дел, тогда как возможности и угрозы сосредотачиваются на будущем.

Вывод по третьему разделу

В третьем разделе выпускной квалификационной работы была проведена демонстрация работы некоторых методов обработки, работоспособность которых была доказана последующей оценкой качества обработки. Также были рассмотрены основные технические требования для оптимального использования редактора.

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы была достигнута поставленная цель повышения визуального качества вторичной обработки медицинских изображений персоналом больницы за счет разработки графического редактора для медицинских изображений, что подтверждается оценкой качества обработанных изображений.

Данный проект, являясь адаптированным для работы с различными типами медицинских изображений, предоставляет возможность медицинскому персоналу облегчить обработку медицинских снимков.

Графический редактор медицинских изображений для его использования медицинским персоналом не требует знания специфики работы методов обработки изображений за счет группировки методов по категориям действий. Также графический редактор включает в себя только те методы, которые можно использовать для обработки медицинских изображений, что избавит от вероятности потери диагностической информации вследствие использования недопустимых методов.

Преимуществом разработанного графического редактора является реализация комплексных методов повышения визуального качества рентгенограмм и выделения костных структур на томограммах, позволяющих в автоматическом режиме подобрать параметры и запустить обработку соответствующих изображений. В результате работы данных комплексов по нажатию одной кнопки осуществляется полная обработка изображения всем набором методов, заложенных в соответствующую подпрограмму.

Перспективой разработки является добавление в графический редактор различных комплексных методов, позволяющих обрабатывать, например, данные ультразвуковых исследований, проводить в автоматическом режиме распознавание форменных элементов крови и другие задачи по обработке медицинских изображений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Балашов, Е. П. Проектирование информационно-управляющих систем / Е.П. Балашов, Д.В. Пузанков. - М.: Радио и связь, 1987. – 256 с.

2. Беликова Т.П. Моделирование линейных фильтров для обработки рентгеновских изображений в задачах медицинской диагностики // Цифровая оптика. Обработка изображений и полей в экспериментальных исследованиях / Под ред. В.И. Сифорова и Л.П. Ярославского. – М.: Наука, 1990. – 176 с.

3. Беликова Т.П. Цифровая обработка томограмм и измерение статистических признаков в задаче ранней дифференциальной диагностики шаровидных образований легких // Цифровая оптика в медицинской интроскопии/ Н.И. Яшунская, Е.А. Коган – М.: Ин–т проблем передачи информ., 1992. – С. 73–88.

4. Бьемон Ж. Итерационные методы улучшения изображений/ Р.Л. Лагендейк, Р.М. Марсеро – ТИИЭР, 1990. – Т.78, № 5. – С. 58–84.

5. Виллевальде А. Ю. Система анализа и обработки медицинских изображений с малоконтрастными объектами: диссертация кандидата технических наук: 05.11.17 / Виллевальде Анна Юрьевна; [Место защиты: С.-Петерб. гос. электротехн. ун-т (ЛЭТИ)].– Санкт-Петербург, 2008. – 143 с.: ил. РГБ ОД, 61 09-5/297.

6. Вичугова А. А. Методы и средства концептуального проектирования информационных систем: сравнительный анализ структурного и объектно-ориентированного подходов / А.А. Вичугова. - М.: Синергия, 2014. - 631 с.

7. Голд Б., Цифровая обработка сигналов: Пер. с англ. / Под ред. А.М. Трахтмана. – М.: Советское радио, 1973. – 368 с.

8. Гонсалес, Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс; пер. с англ. – М. : Техносфера, 2005. – 1072 с..

9. Гранрат Д.Дж. Роль моделей зрения человека в обработке изображений // ТИИЭР. – Т. 69. – № 5. – 1981. – С. 65 – 77.

10. Гуров А.А. Вопросы оценки контрастности сюжетных изображений // Труды ГОИ им. С.И. Вавилова / А.А. Гуров, Н.Н. Порфирьева – т. 44, вып. 178.– Л. – 1979. – С. 31 – 34.

11. Гуров А.А. Обработка изображений на ЭВМ методами линейной фильтрации // Труды ГОИ им. С.И. Вавилова / А.А. Гуров, Н.Н. Порфирьева – Л, 1982. – Вып. 185. – С. 33–50.

12. Ипатова Э. Р. Методологии и технологии системного проектирования информационных систем. Учебник: моногр. / Э.Р. Ипатова. - М.: Флинта, 2016. - 300 с.

13. Калинкина Д. Проблема подавления шума на изображениях и видео и различные подходы к ее решению — Компьютерная графика и мультимедиа: журнал / Д. Калинкина, Д. Ватолин — № 3(2) — 2005

14. Лабинская Д.Е., Исследование методов предобработки изображений рентгенограмм / Д.Е. Лабинская, Т.В. Мартыненко — Донецк, ДонНТУ – 2012, с. 506–511.

15. Литван Р.И.,. Оптимальное градационное преобразование изображений. // Техника кино и телевидения / Р.И. Литван, Ю.И. Аверьянов, Ф.С. Быковская – 1979. – №2. – С. 38 – 41.

16. Нестерук В.Ф., Порфирьева Н.Н. Информационная оценка процесса зрительного восприятия // Оптика и спектроскопия / В.Ф. Нестерук, Н.Н. Порфирьева – 1978. – Т. 44, вып. 4. – С. 801 – 803.

17. Нестерук В.Ф. Преобразование оптических изображений и оценка их качества // Успехи научной фотографии. – М.: Наука. – 1985. – Т. 23. – С. 93 – 102.

18. Прэтт У. Цифровая обработка изображений – М.: Мир, 1982. – 790 с.

19. Рамбо Дж., UML 2.0. Объектно-ориентированное моделирование и разработка. 2-е изд.: Пер. с англ. – СПб.: Питер, 2007

20. Розенфельд А. Распознавание и обработка изображений. – М.: Мир, 1972. – 230 с.

21. Смирнов А.Я. Критерии качества дискретизированных изображений // Труды ГОИ им. С.И. Вавилова. – Т. 57. – Вып. 191. – Л. – 1984.

22. Сойфер В.А. Компьютерная обработка изображений — Соросовский образовательный журнал — №3 — 1996

23. Цуканова О. А. Методология и инструментарий моделирования бизнеспроцессов: учебное пособие – СПб.: Университет ИТМО, 2015.– 100 с.

24. Шамраева, Е.О. Метод компьютерной обработки цифровых аэрофотоснимков [Электронный ресурс] / А.А. Шамраев, Е.О. Шамраева, О.Б. Дудинова // Системи обробки інформації. – Вып.7 (123). – 2014. С.109-112. – Режим доступа: http://www.irbis-nbuv.gov.ua/cgi-bin/irbis_nbuv/cgiirbis_64.exe?C21COM=2&I21DBN=UJRN&P21DBN=UJRN&IMAGE_FILE_DOWNLOAD=1&Image_file_name=PDF/soi_2014_7_24.pdf. – Дата доступа: 18.04.2019. – Загл. с экрана

25. Шлихт Г.Ю. Цифровая обработка цветных изображений. – М., Издательство ЭКОМ, 1997. – 336 с.

26. Ярославский Л.П. Введение в цифровую обработку изображений. – М.: Сов. радио, 1979. – 312 с.

27. Ярославский Л.П. Обработка изображений в медицинской интроскопии / Цифровая оптика в медицинской интроскопии. – М.:ИППИ РАН, 1992. – С. 4–17.

ПРИЛОЖЕНИЕ А

Макет графического интерфейса редактора. Алгоритмы комплекса методов обработки медицинских изображений

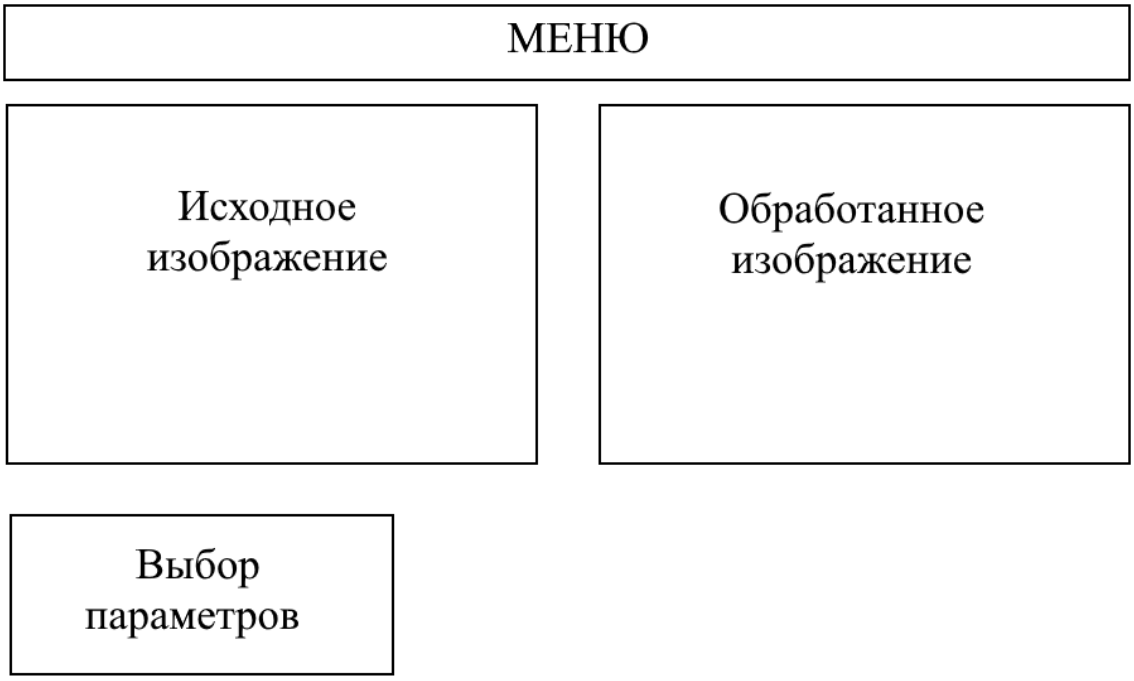


Рисунок А.1 – Макет графического интерфейса

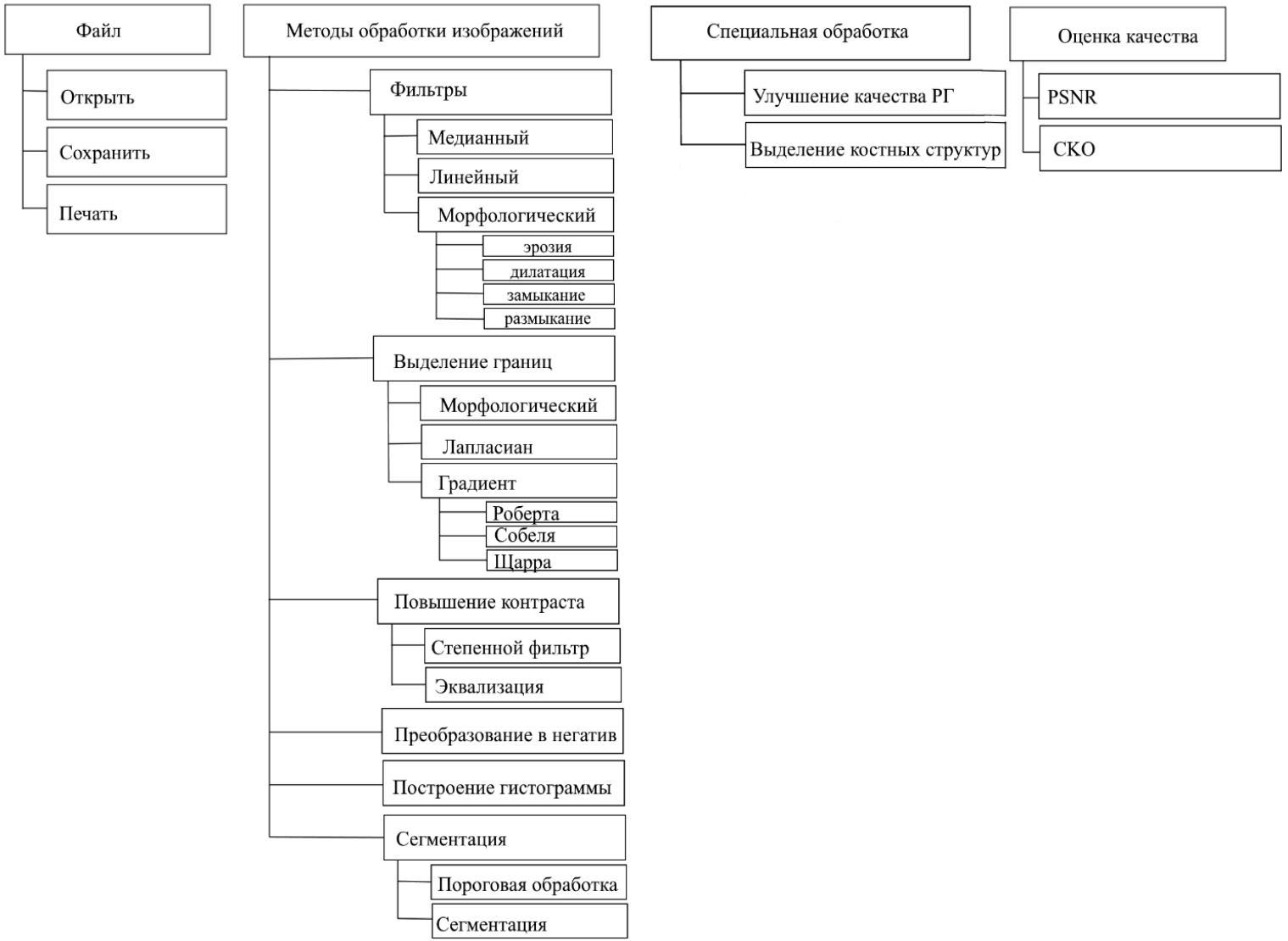
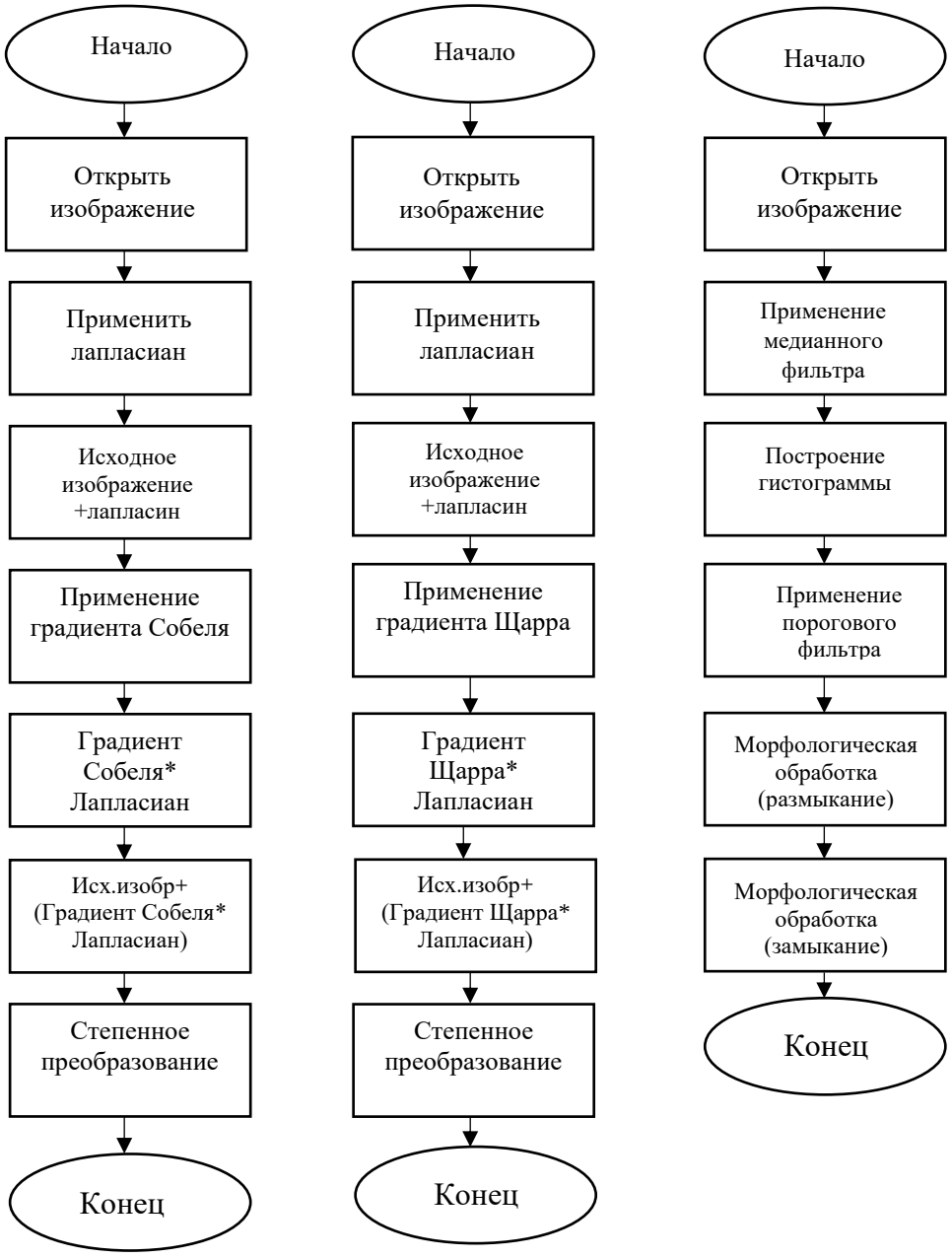


Рисунок А.2 – Структура меню графического редактора



а) б) в)

Рисунок А.3 – Алгоритмы работы комплекса методов: а) для улучшения качества РГ (оператор Собеля); б) для улучшение качества РГ (оператор Щарра); в) для выделения костных структур на КТ

ПРИЛОЖЕНИЕ Б

Исходный код программной реализации методов обработки медицинских изображений

Листинг Б.1 – Реализация построения гистограммы

```
void makeBrightGraph(int n_series) {
MainForm -> Chart1  -> Series[n_series] -> Clear();
std::vector<int> brightness; brightness.resize(256);
for (int i = 0; i < 256; i++) brightness[i] = 0;
Graphics::TBitmap *nBitmap = MainForm -> Image2 -> Picture -> Bitmap;
int w = nBitmap -> Width; int h = nBitmap -> Height;
for (int x = 0; x < w; x++) {
for (int y = 0; y < h; y++) {
int pixB = GetRValue(nBitmap -> Canvas -> Pixels[x][y]) * 0.3 +
GetGValue(nBitmap -> Canvas -> Pixels[x][y]) * 0.59 + GetBValue(nBitmap -
> Canvas -> Pixels[x][y]) * 0.11;
for (int b = 0; b < 256; b++) {
//rgb_index[0] * 0.3 + rgb_index[1] * 0.59 + rgb_index[2] * 0.11)
if (b == pixB) {
brightness[b]++;
break;
}
}
}
}
for (int i = 0; i < 255; i++) MainForm -> Chart1 -> Series[n_series] ->
AddY(brightness[i]);
}
```

Листинг Б.2 – Реализации процессов наложения изображений

```
vector< vector<TColor> > add_image(vector< vector<TColor> > x, vector<
vector<TColor> > y) {
  vector<int> R_values;
  vector<int> G_values;
  vector<int> B_values;
  vector< vector< vector<int> > > result;
  result.resize(x.size());
  for (int k = 0; k < result.size(); k++) result[k].resize(x[0].size());

  for (int i = 0; i < y.size(); i++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int j = 0; j < y[i].size(); j++) {
      int Ry = GetRValue(y[i][j]); int Rx = GetRValue(x[i][j]) + Ry;
      int Gy = GetGValue(y[i][j]); int Gx = GetGValue(x[i][j]) + Gy;
      int By = GetBValue(y[i][j]); int Bx = GetBValue(x[i][j]) + By;

      R_values.push_back(Rx);
      B_values.push_back(Gx);
      G_values.push_back(Bx);

      result[i][j].push_back(Rx);
      result[i][j].push_back(Gx);
```

```
      result[i][j].push_back(Bx);
    }
  }

  int Rmax = return_max(R_values);
  int Gmax = return_max(G_values);
  int Bmax = return_max(B_values);

  for (int i = 0; i < result.size(); i++) {
    for (int j = 0; j < result[i].size(); j++) {
      int Rx = result[i][j][0]; int Gx = result[i][j][1]; int Bx =
result[i][j][2];
      Rx = scale256(Rx, Rmax);
      Bx = scale256(Bx, Bmax);
      Gx = scale256(Gx, Gmax);
      x[i][j] = (TColor) RGB((Byte) Rx, (Byte) Gx, (Byte) Bx);
    }
  }
  return x;
}

vector< vector<TColor> > multiply_image(vector< vector<TColor> > x,
vector< vector<TColor> > y) {
  vector<int> R_values;
  vector<int> G_values;
  vector<int> B_values;
  vector< vector< vector<int> > > result;
  result.resize(x.size());
  for (int k = 0; k < result.size(); k++) result[k].resize(x[0].size());

  for (int i = 0; i < y.size(); i++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int j = 0; j < y[i].size(); j++) {
      int Ry = GetRValue(y[i][j]); int Rx = GetRValue(x[i][j]) * Ry;
      int Gy = GetGValue(y[i][j]); int Gx = GetGValue(x[i][j]) * Gy;
      int By = GetBValue(y[i][j]); int Bx = GetBValue(x[i][j]) * By;

      R_values.push_back(Rx);
      B_values.push_back(Gx);
      G_values.push_back(Bx);

      result[i][j].push_back(Rx);
      result[i][j].push_back(Gx);
      result[i][j].push_back(Bx);
    }
  }

  int Rmax = return_max(R_values);
  int Gmax = return_max(G_values);
  int Bmax = return_max(B_values);

  for (int i = 0; i < result.size(); i++) {
    for (int j = 0; j < result[i].size(); j++) {
      int Rx = result[i][j][0]; int Gx = result[i][j][1]; int Bx =
result[i][j][2];
      Rx = scale256(Rx, Rmax); Gx = scale256(Gx, Gmax); Bx = scale256(Bx,
Bmax);

      x[i][j] = (TColor) RGB((Byte) Rx, (Byte) Gx, (Byte) Bx);
    }
  }
  return x;
}
```

Листинг Б.3 – Реализация степенного преобразования

```
vector< vector<TColor> > power_processing_scaled(vector< vector<TColor> >
pixels, float C, float gamma) {
  vector< vector <TColor> > new_pixels; new_pixels = pixels;
  vector<int> R_values; vector<int> G_values; vector<int> B_values;

  vector< vector< vector<int> > > result;
  result.resize(pixels.size());
  for (int k = 0; k < result.size(); k++)
result[k].resize(pixels[0].size());

  MainForm -> Panel1 -> Visible = true;

  for (int i = 0; i < pixels.size(); i++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int j = 0; j < pixels[i].size(); j++) {
      int Rx = C * pow(GetRValue(pixels[i][j]), gamma);
      int Gx = C * pow(GetGValue(pixels[i][j]), gamma);
      int Bx = C * pow(GetBValue(pixels[i][j]), gamma);

      R_values.push_back(Rx);
      B_values.push_back(Gx);
      G_values.push_back(Bx);

      result[i][j].push_back(Rx);
      result[i][j].push_back(Gx);
      result[i][j].push_back(Bx);
    }
  }

  int Rmax = return_max(R_values);
  int Gmax = return_max(G_values);
  int Bmax = return_max(B_values);

  for (int i = 0; i < result.size(); i++) {
    for (int j = 0; j < result[i].size(); j++) {
      int Req = result[i][j][0]; int Geq = result[i][j][1]; int Beq =
result[i][j][2];
      Req = scale256(Req, Rmax);
      Beq = scale256(Beq, Bmax);
      Geq = scale256(Geq, Gmax);
      new_pixels[i][j] = (TColor) RGB((Byte) Req, (Byte) Geq, (Byte)
Beq);
    }
  }
  return new_pixels;
}
void __fastcall TMainForm::Button1Click(TObject *Sender)
{
 float gamma = StrToFloat(MainForm -> Edit1 -> Text);
 float C = StrToFloat(MainForm -> Edit2 -> Text);

 Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
 bufferBitmap -> Assign(pBitmap);
 std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);

 MainForm -> Panel1 -> Visible = true;
 MainForm -> ProgressBar1 -> Max = pixels.size();

 /*for (int i = 0; i < pixels.size(); i++) {
   MainForm -> ProgressBar1 -> StepIt();
```

```
   for (int j = 0; j < pixels[i].size(); j++) pixels[i][j] =
power_processing(pixels[i][j], gamma, C);
 }*/

 pixels = power_processing_scaled(pixels, C, gamma);

 setNewBitmap(pixels);
 MainForm -> Panel1 -> Visible = false;
 makeBrightGraph(0);
}
```

Листинг Б.4 – Реализация автоматического расчёта параметра гамма для степенного преобразования

```
void __fastcall TMainForm::Button8Click(TObject *Sender)
{
  Edit5 -> Text = 128.0 / (256.0 - autoTreshold());
  Edit2 -> Text = 1;
}
Листинг 3.5 – Реализация эквализации гистограммы
vector< vector<TColor> > eq(vector< vector<TColor> > pixels) {
  vector< vector <TColor> > new_pixels; new_pixels = pixels;
  vector<int> R_values; vector<int> G_values; vector<int> B_values;

  vector< vector< vector<int> > > result;
  result.resize(pixels.size());
  for (int k = 0; k < result.size(); k++)
result[k].resize(pixels[0].size());

  MainForm -> Panel1 -> Visible = true;

  for (int i = 0; i < pixels.size(); i++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int j = 0; j < pixels[i].size(); j++) {
      int Req = GetRValue(pixels[i][j]) * 2;
      int Geq = GetGValue(pixels[i][j]) * 2;
      int Beq = GetBValue(pixels[i][j]) * 2;

      R_values.push_back(Req);
      B_values.push_back(Geq);
      G_values.push_back(Beq);

      result[i][j].push_back(Req);
      result[i][j].push_back(Geq);
      result[i][j].push_back(Beq);

    }
  }

  int Rmax = return_max(R_values);
  int Gmax = return_max(G_values);
  int Bmax = return_max(B_values);

  for (int i = 0; i < result.size(); i++) {
    for (int j = 0; j < result[i].size(); j++) {
      int Rx = result[i][j][0]; int Gx = result[i][j][1]; int Bx =
result[i][j][2];
      Rx = scale256(Rx, Rmax);
      Bx = scale256(Bx, Bmax);
      Gx = scale256(Gx, Gmax);
      new_pixels[i][j] = (TColor) RGB((Byte) Rx, (Byte) Gx, (Byte) Bx);
    }
  }
```

```
  return new_pixels;
}

void __fastcall TMainForm::N27Click(TObject *Sender)
{
  Graphics::TBitmap *firstBitmap = MainForm -> Image1 -> Picture ->
Bitmap;
  Graphics::TBitmap *secondBitmap = MainForm -> Image2 -> Picture ->
Bitmap;

  bufferBitmap -> Assign(secondBitmap);

  std::vector< std::vector<TColor> > pixels =
getPixelsMatrix(secondBitmap);
  vector< vector <TColor> > new_pixels; new_pixels = pixels;
  MainForm -> Panel1 -> Visible = true;

  new_pixels = eq(pixels);

  setNewBitmap(new_pixels);
  MainForm -> Panel1 -> Visible = false;
  makeBrightGraph(0);
}
```

Листинг Б.6 – Реализация линейного фильтра

```
TColor getAVGMask(std::vector<TColor> mask) {
 int b = 0; std::vector<int> rgb; rgb.resize(3);
  for (int i = 0; i < mask.size(); i++) {
    rgb[0] = GetRValue(mask[i]);
    rgb[1] = GetGValue(mask[i]);
    rgb[2] = GetBValue(mask[i]);
    b += getBright(rgb);
  }
  b /= mask.size();
  return b;
}

TColor brightAVG3x3(std::vector< std::vector<TColor> > pixels, int x, int
y) {
  std::vector<TColor> mask; mask.resize(9); TColor avg_bright;
  mask[0] = pixels[x - 1][y - 1];
  mask[1] = pixels[x - 1][y];
  mask[2] = pixels[x - 1][y + 1];
  mask[3] = pixels[x][y - 1];
  mask[4] = pixels[x][y];
  mask[5] = pixels[x][y + 1];
  mask[6] = pixels[x + 1][y - 1];
  mask[7] = pixels[x + 1][y];
  mask[8] = pixels[x + 1][y + 1];

  avg_bright = RGB(getAVGMask(mask), getAVGMask(mask), getAVGMask(mask));

  return avg_bright;
}

TColor brightAVG4x4(std::vector< std::vector<TColor> > pixels, int x, int
y) {
  std::vector<TColor> mask; mask.resize(16); TColor avg_bright;
  mask[0] = pixels[x - 2][y - 2];
  mask[1] = pixels[x - 2][y - 1];
  mask[2] = pixels[x - 2][y];
  mask[3] = pixels[x - 2][y + 1];
```

```
  mask[4] = pixels[x - 2][y + 2];

  mask[5] = pixels[x - 1][y - 2];
  mask[6] = pixels[x - 1][y - 1];
  mask[7] = pixels[x - 1][y];
  mask[8] = pixels[x - 1][y + 1];
  mask[9] = pixels[x - 1][y + 2];

  mask[10] = pixels[x][y - 2];
  mask[11] = pixels[x][y - 1];
  mask[12] = pixels[x][y];
  mask[13] = pixels[x][y + 1];
  mask[14] = pixels[x][y + 2];

  mask[15] = pixels[x + 1][y - 2];
  mask[16] = pixels[x + 1][y - 1];

  avg_bright = RGB(getAVGMask(mask), getAVGMask(mask), getAVGMask(mask));

  return avg_bright;
}

TColor brightAVG5x5(std::vector< std::vector<TColor> > pixels, int x, int
y) {
  std::vector<TColor> mask; mask.resize(25); TColor avg_bright;
  mask[0] = pixels[x - 2][y - 2];
  mask[1] = pixels[x - 2][y - 1];
  mask[2] = pixels[x - 2][y];
  mask[3] = pixels[x - 2][y + 1];
  mask[4] = pixels[x - 2][y + 2];

  mask[5] = pixels[x - 1][y - 2];
  mask[6] = pixels[x - 1][y - 1];
  mask[7] = pixels[x - 1][y];
  mask[8] = pixels[x - 1][y + 1];
  mask[9] = pixels[x - 1][y + 2];

  mask[10] = pixels[x][y - 2];
  mask[11] = pixels[x][y - 1];
  mask[12] = pixels[x][y];
  mask[13] = pixels[x][y + 1];
  mask[14] = pixels[x][y + 2];

  mask[15] = pixels[x + 1][y - 2];
  mask[16] = pixels[x + 1][y - 1];
  mask[17] = pixels[x + 1][y];
  mask[18] = pixels[x + 1][y + 1];
  mask[19] = pixels[x + 1][y + 2];

  mask[20] = pixels[x + 2][y - 2];
  mask[21] = pixels[x + 2][y - 1];
  mask[22] = pixels[x + 2][y];
  mask[23] = pixels[x + 2][y + 1];
  mask[24] = pixels[x + 2][y + 2];

  avg_bright = RGB(getAVGMask(mask), getAVGMask(mask), getAVGMask(mask));

  return avg_bright;
}

void __fastcall TMainForm::Button3Click(TObject *Sender)
{
 Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
 bufferBitmap -> Assign(pBitmap);
```

```
 std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);

 int slide;
 TColor (*filter) (std::vector< std::vector<TColor> >, int, int);

 if (RadioButton3 -> Checked) {
   slide = 1;
   filter = brightAVG3x3;
 }
 if (RadioButton4 -> Checked) {
   slide = 2;
   filter = brightAVG4x4;
 }
 if (RadioButton5 -> Checked) {
   slide = 2;
   filter = brightAVG5x5;
 }
 MainForm -> Panel1 -> Visible = true;
 MainForm -> ProgressBar1 -> Max = pixels.size() - slide;

 for (int i = slide; i < pixels.size() - slide; i++) {
   MainForm -> ProgressBar1 -> StepIt();
   for (int j = slide; j < pixels[i].size() - slide; j++) pixels[i][j] =
filter(pixels, i, j);
 }
 setNewBitmap(pixels);
 MainForm -> Panel1 -> Visible = false;
 makeBrightGraph(0);
}
```

Листинг Б.7 – Реализация медианного фильтра

```
TColor brightSort3x3(std::vector< std::vector<TColor> > pixels, int x,
int y) {
  std::vector<TColor> mask; mask.resize(9);
  mask[0] = pixels[x - 1][y - 1];
  mask[1] = pixels[x - 1][y];
  mask[2] = pixels[x - 1][y + 1];
  mask[3] = pixels[x][y - 1];
  mask[4] = pixels[x][y];
  mask[5] = pixels[x][y + 1];
  mask[6] = pixels[x + 1][y - 1];
  mask[7] = pixels[x + 1][y];
  mask[8] = pixels[x + 1][y + 1];

  std::sort(mask.begin(), mask.end());

  return mask[4];
}

TColor brightSort5x5(std::vector< std::vector<TColor> > pixels, int x,
int y) {
  std::vector<TColor> mask; mask.resize(25);
  mask[0] = pixels[x - 2][y - 2];
  mask[1] = pixels[x - 2][y - 1];
  mask[2] = pixels[x - 2][y];
  mask[3] = pixels[x - 2][y + 1];
  mask[4] = pixels[x - 2][y + 2];

  mask[5] = pixels[x - 1][y - 2];
  mask[6] = pixels[x - 1][y - 1];
  mask[7] = pixels[x - 1][y];
  mask[8] = pixels[x - 1][y + 1];
```

```
  mask[9] = pixels[x - 1][y + 2];

  mask[10] = pixels[x][y - 2];
  mask[11] = pixels[x][y - 1];
  mask[12] = pixels[x][y];
  mask[13] = pixels[x][y + 1];
  mask[14] = pixels[x][y + 2];

  mask[15] = pixels[x + 1][y - 2];
  mask[16] = pixels[x + 1][y - 1];
  mask[17] = pixels[x + 1][y];
  mask[18] = pixels[x + 1][y + 1];
  mask[19] = pixels[x + 1][y + 2];

  mask[20] = pixels[x + 2][y - 2];
  mask[21] = pixels[x + 2][y - 1];
  mask[22] = pixels[x + 2][y];
  mask[23] = pixels[x + 2][y + 1];
  mask[24] = pixels[x + 2][y + 2];

  std::sort(mask.begin(), mask.end());

  return mask[13];
}

void __fastcall TMainForm::Button5Click(TObject *Sender)
{
 Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
 bufferBitmap -> Assign(pBitmap);
 std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);

 int slide;
 TColor (*filter) (std::vector< std::vector<TColor> >, int, int);

 if (RadioButton1 -> Checked) {
   slide = 1;
   filter = brightSort3x3;
 }
 if (RadioButton2 -> Checked) {
   slide = 2;
   filter = brightSort5x5;
 }
 MainForm -> Panel1 -> Visible = true;
 MainForm -> ProgressBar1 -> Max = pixels.size() - slide;

 for (int i = slide; i < pixels.size() - slide; i++) {
   MainForm -> ProgressBar1 -> StepIt();
   for (int j = slide; j < pixels[i].size() - slide; j++) pixels[i][j] =
filter(pixels, i, j);
 }
 setNewBitmap(pixels);
 MainForm -> Panel1 -> Visible = false;
 makeBrightGraph(0);
}
```

Листинг Б.8 – Реализация методов математической морфологии (дилатация)

```
std::vector< std::vector<TColor> > dilCircle3x3(std::vector<
std::vector<TColor> > pixels)
{
  std::vector<TColor> mask; mask.resize(4);
  std::vector<int> b_mask; b_mask.resize(4);
```

```
  std::vector< std::vector<TColor> > new_pixels = pixels;
  int s;

  for (unsigned x = 1; x < pixels.size() - 1; x++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (unsigned y = 1; y < pixels[x].size() - 1; y++) {
      if (pixels[x][y] == 0) {
        new_pixels[x - 1][y] = (TColor) 0;
        new_pixels[x + 1][y] = (TColor) 0;
        new_pixels[x][y - 1] = (TColor) 0;
        new_pixels[x][y + 1] = (TColor) 0;
      }
    }
  }
  return new_pixels;
}

std::vector< std::vector<TColor> > dilCircle5x5(std::vector<
std::vector<TColor> > pixels)
{
  std::vector<TColor> mask; mask.resize(10);
  std::vector<int> b_mask; b_mask.resize(10);
  std::vector< std::vector<TColor> > new_pixels = pixels;
  int s;

  for (unsigned x = 2; x < pixels.size() - 2; x++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (unsigned y = 2; y < pixels[x].size() - 2; y++) {
      if (pixels[x][y] == 0) {
        new_pixels[x - 2][y] = (TColor) 0;
        new_pixels[x - 1][y] = (TColor) 0;
        new_pixels[x + 1][y] = (TColor) 0;
        new_pixels[x + 2][y] = (TColor) 0;
        new_pixels[x][y - 1] = (TColor) 0;
        new_pixels[x][y - 2] = (TColor) 0;
        new_pixels[x][y + 1] = (TColor) 0;
        new_pixels[x][y + 2] = (TColor) 0;
      }
    }
  }
  return new_pixels;
}

std::vector< std::vector<TColor> > dilSquare3x3(std::vector<
std::vector<TColor> > pixels)
{
  std::vector<TColor> mask; mask.resize(9);
  std::vector<int> b_mask; b_mask.resize(9);
  std::vector< std::vector<TColor> > new_pixels = pixels;
  int s;

  for (unsigned x = 1; x < pixels.size() - 1; x++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (unsigned y = 1; y < pixels[x].size() - 1; y++) {
      if (pixels[x][y] == 0) {
        new_pixels[x - 1][y - 1] = (TColor) RGB(0, 0, 0);
        new_pixels[x - 1][y] = (TColor) RGB(0, 0, 0);
        new_pixels[x - 1][y + 1] = (TColor)RGB(0, 0, 0);
        new_pixels[x][y - 1] = (TColor)RGB(0, 0, 0);
        new_pixels[x][y + 1] = (TColor)RGB(0, 0, 0);
        new_pixels[x + 1][y - 1] = (TColor)RGB(0, 0, 0);
        new_pixels[x + 1][y] = (TColor)RGB(0, 0, 0);
        new_pixels[x + 1][y + 1] = (TColor)RGB(0, 0, 0);
      }
```

```
    }
  }
  return new_pixels;
}

std::vector< std::vector<TColor> > dilSquare5x5(std::vector<
std::vector<TColor> > pixels)
{
  std::vector<TColor> mask; mask.resize(25);
  std::vector<int> b_mask; b_mask.resize(25);
  std::vector< std::vector<TColor> > new_pixels = pixels;
  int s;

  for (unsigned x = 2; x < pixels.size() - 2; x++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (unsigned y = 2; y < pixels[x].size() - 2; y++) {
      if (pixels[x][y] == 0) {
        new_pixels[x - 2][y - 2] = (TColor) 0;
        new_pixels[x - 2][y - 1] = (TColor) 0;
        new_pixels[x - 2][y] = (TColor) 0;
        new_pixels[x - 2][y + 1] = (TColor) 0;
        new_pixels[x - 2][y + 2] = (TColor) 0;

        new_pixels[x - 1][y - 2] = (TColor) 0;
        new_pixels[x - 1][y - 1] = (TColor) 0;
        new_pixels[x - 1][y] = (TColor) 0;
        new_pixels[x - 1][y + 1] = (TColor) 0;
        new_pixels[x - 1][y + 2] = (TColor) 0;

        new_pixels[x][y - 2] = (TColor) 0;
        new_pixels[x][y - 1] = (TColor) 0;
        new_pixels[x][y] = (TColor) 0;
        new_pixels[x][y + 1] = (TColor) 0;
        new_pixels[x][y + 2] = (TColor) 0;

        new_pixels[x + 1][y - 2] = (TColor) 0;
        new_pixels[x + 1][y - 1] = (TColor) 0;
        new_pixels[x + 1][y] = (TColor) 0;
        new_pixels[x + 1][y + 1] = (TColor) 0;
        new_pixels[x + 1][y + 2] = (TColor) 0;

        new_pixels[x + 2][y - 2] = (TColor) 0;
        new_pixels[x + 2][y - 1] = (TColor) 0;
        new_pixels[x + 2][y] = (TColor) 0;
        new_pixels[x + 2][y + 1] = (TColor) 0;
        new_pixels[x + 2][y + 2] = (TColor) 0;
      }
    }
  }
  return new_pixels;
}

std::vector< std::vector<TColor> > dilBigSquare(std::vector<
std::vector<TColor> > pixels)
{
  std::vector<TColor> mask; mask.resize(15);
  std::vector<int> b_mask; b_mask.resize(15);
  std::vector< std::vector<TColor> > new_pixels = pixels;
  int s;

  for (unsigned x = 1; x < pixels.size() - 1; x++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (unsigned y = 2; y < pixels[x].size() - 2; y++) {
      if (pixels[x][y] == 0) {
```

```
        new_pixels[x - 1][y - 2] = (TColor) 0;
        new_pixels[x - 1][y - 1] = (TColor) 0;
        new_pixels[x - 1][y] = (TColor) 0;
        new_pixels[x - 1][y + 1] = (TColor) 0;
        new_pixels[x - 1][y + 2] = (TColor) 0;

        new_pixels[x][y - 2] = (TColor) 0;
        new_pixels[x][y - 1] = (TColor) 0;
        new_pixels[x][y] = (TColor) 0;
        new_pixels[x][y + 1] = (TColor) 0;
        new_pixels[x][y + 2] = (TColor) 0;

        new_pixels[x + 1][y - 2] = (TColor) 0;
        new_pixels[x + 1][y - 1] = (TColor) 0;
        new_pixels[x + 1][y] = (TColor) 0;
        new_pixels[x + 1][y + 1] = (TColor) 0;
        new_pixels[x + 1][y + 2] = (TColor) 0;

      }
    }
  }
  return new_pixels;
}

void __fastcall TMainForm::dilButtonClick(TObject *Sender)
{
  Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
  bufferBitmap -> Assign(pBitmap);
  std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);

  MainForm -> Panel1 -> Visible = true;

  if (RadioButton6 -> Checked) {
    if (RadioButton8 -> Checked) pixels = dilCircle3x3(pixels);
    if (RadioButton7 -> Checked) pixels = dilCircle5x5(pixels);
  }

  if (RadioButton9 -> Checked) {
    if (RadioButton10 -> Checked) pixels = dilSquare3x3(pixels);
    if (RadioButton11 -> Checked) pixels = dilSquare5x5(pixels);
  }

  if (RadioButton12 -> Checked) pixels = dilBigSquare(pixels);

  setNewBitmap(pixels);
  MainForm -> Panel1 -> Visible = false;
  makeBrightGraph(0);
}
```

Листинг Б.9 – Реализация методов математической морфологии (эрозия)

```
std::vector<TColor> getBrightMask5x5(std::vector< std::vector<TColor> >
pixels, int x, int y) {
  std::vector<TColor> mask; mask.resize(25); TColor avg_bright;
  mask[0] = pixels[x - 2][y - 2];
  mask[1] = pixels[x - 2][y - 1];
  mask[2] = pixels[x - 2][y];
  mask[3] = pixels[x - 2][y + 1];
  mask[4] = pixels[x - 2][y + 2];

  mask[5] = pixels[x - 1][y - 2];
  mask[6] = pixels[x - 1][y - 1];
  mask[7] = pixels[x - 1][y];
```

```
  mask[8] = pixels[x - 1][y + 1];
  mask[9] = pixels[x - 1][y + 2];

  mask[10] = pixels[x][y - 2];
  mask[11] = pixels[x][y - 1];
  mask[12] = pixels[x][y];
  mask[13] = pixels[x][y + 1];
  mask[14] = pixels[x][y + 2];

  mask[15] = pixels[x + 1][y - 2];
  mask[16] = pixels[x + 1][y - 1];
  mask[17] = pixels[x + 1][y];
  mask[18] = pixels[x + 1][y + 1];
  mask[19] = pixels[x + 1][y + 2];

  mask[20] = pixels[x + 2][y - 2];
  mask[21] = pixels[x + 2][y - 1];
  mask[22] = pixels[x + 2][y];
  mask[23] = pixels[x + 2][y + 1];
  mask[24] = pixels[x + 2][y + 2];

  return mask;
}


std::vector< std::vector<TColor> > erosedeSquare5x5(std::vector<
std::vector<TColor> > pixels)
{
  std::vector<TColor> mask; mask.resize(25);
  std::vector<int> b_mask; b_mask.resize(25);
  std::vector< std::vector<TColor> > new_pixels = pixels;
  int s;

  MainForm -> ProgressBar1 -> Max = pixels.size();

  for (int x = 2; x < pixels.size() - 2; x++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int y = 2; y < pixels[x].size() - 2; y++) {
      mask = getBrightMask5x5(pixels, x, y);
      b_mask = getBrights(mask);
      if (sum(b_mask) >= 255) {
        new_pixels[x - 2][y - 2] = RGB(255, 255, 255);
        new_pixels[x - 2][y - 1] = RGB(255, 255, 255);
        new_pixels[x - 2][y] = RGB(255, 255, 255);
        new_pixels[x - 2][y + 1] = RGB(255, 255, 255);
        new_pixels[x - 2][y + 2] = RGB(255, 255, 255);

        new_pixels[x - 1][y - 2] = RGB(255, 255, 255);
        new_pixels[x - 1][y - 1] = RGB(255, 255, 255);
        new_pixels[x - 1][y] = RGB(255, 255, 255);
        new_pixels[x - 1][y + 1] = RGB(255, 255, 255);
        new_pixels[x - 1][y + 2] = RGB(255, 255, 255);

        new_pixels[x][y - 2] = RGB(255, 255, 255);
        new_pixels[x][y - 1] = RGB(255, 255, 255);
        new_pixels[x][y] = RGB(255, 255, 255);
        new_pixels[x][y + 1] = RGB(255, 255, 255);
        new_pixels[x][y + 2] = RGB(255, 255, 255);

        new_pixels[x + 1][y - 2] = RGB(255, 255, 255);
        new_pixels[x + 1][y - 1] = RGB(255, 255, 255);
        new_pixels[x + 1][y] = RGB(255, 255, 255);
        new_pixels[x + 1][y + 1] = RGB(255, 255, 255);
        new_pixels[x + 1][y + 2] = RGB(255, 255, 255);
```

```
        new_pixels[x + 2][y - 2] = RGB(255, 255, 255);
        new_pixels[x + 2][y - 1] = RGB(255, 255, 255);
        new_pixels[x + 2][y] = RGB(255, 255, 255);
        new_pixels[x + 2][y + 1] = RGB(255, 255, 255);
        new_pixels[x + 2][y + 2] = RGB(255, 255, 255);
      }
    }
  }
  return new_pixels;
}

std::vector< std::vector<TColor> > erosedeSquare3x3(std::vector<
std::vector<TColor> > pixels)
{
  std::vector<TColor> mask; mask.resize(9);
  std::vector<int> b_mask; b_mask.resize(9);
  std::vector< std::vector<TColor> > new_pixels = pixels;
  int s;

  MainForm -> ProgressBar1 -> Max = pixels.size();

  for (int x = 1; x < pixels.size() - 1; x++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int y = 1; y < pixels[x].size() - 1; y++) {
      mask = getBrightMask(pixels, x, y);
      b_mask = getBrights(mask);
      if (sum(b_mask) >= 255) {
        new_pixels[x - 1][y - 1] = RGB(255, 255, 255);
        new_pixels[x - 1][y] = RGB(255, 255, 255);
        new_pixels[x - 1][y + 1] = RGB(255, 255, 255);
        new_pixels[x][y - 1] = RGB(255, 255, 255);
        new_pixels[x][y + 1] = RGB(255, 255, 255);
        new_pixels[x + 1][y - 1] = RGB(255, 255, 255);
        new_pixels[x + 1][y] = RGB(255, 255, 255);
        new_pixels[x + 1][y + 1] = RGB(255, 255, 255);
      }
    }
  }
  return new_pixels;
}

std::vector<TColor> getBrightMask3Circle(std::vector< std::vector<TColor>
> pixels, int x, int y) {
  std::vector<TColor> mask; mask.resize(5);
  mask[0] = pixels[x - 1][y];
  mask[1] = pixels[x + 1][y];
  mask[2] = pixels[x][y];
  mask[3] = pixels[x][y - 1];
  mask[4] = pixels[x][y + 1];

  return mask;
}

std::vector< std::vector<TColor> > erosedeCircle3x3(std::vector<
std::vector<TColor> > pixels)
{
  std::vector<TColor> mask; mask.resize(9);
  std::vector<int> b_mask; b_mask.resize(9);
  std::vector< std::vector<TColor> > new_pixels = pixels;
  int s;

  MainForm -> ProgressBar1 -> Max = pixels.size();
```

```
  for (int x = 1; x < pixels.size() - 1; x++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int y = 1; y < pixels[x].size() - 1; y++) {
      mask = getBrightMask(pixels, x, y);
      b_mask = getBrights(mask);
      if (sum(b_mask) >= 255) {
        new_pixels[x - 1][y] = RGB(255, 255, 255);
        new_pixels[x + 1][y] = RGB(255, 255, 255);
        new_pixels[x][y - 1] = RGB(255, 255, 255);
        new_pixels[x][y + 1] = RGB(255, 255, 255);
      }
    }
  }
  return new_pixels;
}

std::vector<TColor> getBrightMask5Circle(std::vector< std::vector<TColor>
> pixels, int x, int y) {
  std::vector<TColor> mask; mask.resize(9);
  mask[0] = pixels[x - 2][y];
  mask[1] = pixels[x - 1][y];
  mask[2] = pixels[x + 2][y];
  mask[3] = pixels[x + 1][y];
  mask[4] = pixels[x][y];
  mask[5] = pixels[x][y - 2];
  mask[6] = pixels[x][y - 1];
  mask[7] = pixels[x][y + 1];
  mask[8] = pixels[x][y + 2];

  return mask;
}

std::vector< std::vector<TColor> > erosedeCircle5x5(std::vector<
std::vector<TColor> > pixels)
{
  std::vector<TColor> mask; mask.resize(9);
  std::vector<int> b_mask; b_mask.resize(9);
  std::vector< std::vector<TColor> > new_pixels = pixels;
  int s;

  MainForm -> ProgressBar1 -> Max = pixels.size();

  for (int x = 2; x < pixels.size() - 2; x++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int y = 2; y < pixels[x].size() - 2; y++) {
      mask = getBrightMask(pixels, x, y);
      b_mask = getBrights(mask);
      if (sum(b_mask) >= 255) {
        new_pixels[x - 2][y] = RGB(255, 255, 255);
        new_pixels[x - 1][y] = RGB(255, 255, 255);
        new_pixels[x + 1][y] = RGB(255, 255, 255);
        new_pixels[x + 2][y] = RGB(255, 255, 255);
        new_pixels[x][y - 1] = RGB(255, 255, 255);
        new_pixels[x][y - 2] = RGB(255, 255, 255);
        new_pixels[x][y + 1] = RGB(255, 255, 255);
        new_pixels[x][y + 2] = RGB(255, 255, 255);
      }
    }
  }
  return new_pixels;
}

std::vector<TColor> getBrightMask3x5(std::vector< std::vector<TColor> >
pixels, int x, int y) {
```

```
  std::vector<TColor> mask; mask.resize(15); TColor avg_bright;
  mask[0] = pixels[x - 1][y - 2];
  mask[1] = pixels[x - 1][y - 1];
  mask[2] = pixels[x - 1][y];
  mask[3] = pixels[x - 1][y + 1];
  mask[4] = pixels[x - 1][y + 2];

  mask[5] = pixels[x][y - 2];
  mask[6] = pixels[x][y - 1];
  mask[7] = pixels[x][y];
  mask[8] = pixels[x][y + 1];
  mask[9] = pixels[x][y + 2];

  mask[10] = pixels[x + 1][y - 2];
  mask[11] = pixels[x + 1][y - 1];
  mask[12] = pixels[x + 1][y];
  mask[13] = pixels[x + 1][y + 1];
  mask[14] = pixels[x + 1][y + 2];

  return mask;
}

std::vector< std::vector<TColor> > erosedeBigSquare(std::vector<
std::vector<TColor> > pixels)
{
  std::vector<TColor> mask; mask.resize(15);
  std::vector<int> b_mask; b_mask.resize(15);
  std::vector< std::vector<TColor> > new_pixels = pixels;
  int s;

  MainForm -> ProgressBar1 -> Max = pixels.size();

  for (int x = 1; x < pixels.size() - 1; x++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int y = 2; y < pixels[x].size() - 2; y++) {
      mask = getBrightMask3x5(pixels, x, y);
      b_mask = getBrights(mask);
      if (sum(b_mask) >= 255) {

        new_pixels[x - 1][y - 2] = RGB(255, 255, 255);
        new_pixels[x - 1][y - 1] = RGB(255, 255, 255);
        new_pixels[x - 1][y] = RGB(255, 255, 255);
        new_pixels[x - 1][y + 1] = RGB(255, 255, 255);
        new_pixels[x - 1][y + 2] = RGB(255, 255, 255);

        new_pixels[x][y - 2] = RGB(255, 255, 255);
        new_pixels[x][y - 1] = RGB(255, 255, 255);
        new_pixels[x][y] = RGB(255, 255, 255);
        new_pixels[x][y + 1] = RGB(255, 255, 255);
        new_pixels[x][y + 2] = RGB(255, 255, 255);

        new_pixels[x + 1][y - 2] = RGB(255, 255, 255);
        new_pixels[x + 1][y - 1] = RGB(255, 255, 255);
        new_pixels[x + 1][y] = RGB(255, 255, 255);
        new_pixels[x + 1][y + 1] = RGB(255, 255, 255);
        new_pixels[x + 1][y + 2] = RGB(255, 255, 255);

      }
    }
  }
  return new_pixels;
}

void __fastcall TMainForm::Button4Click(TObject *Sender)
```

```
{
 Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
 bufferBitmap -> Assign(pBitmap);
 std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);

 MainForm -> Panel1 -> Visible = true;

 int figure; int d = 3;
 if (RadioButton6 -> Checked) {
   figure = 1;
   if (RadioButton8 -> Checked) pixels = erosedeCircle3x3(pixels);
   if (RadioButton7 -> Checked) pixels = erosedeCircle5x5(pixels);
 }

 if (RadioButton9 -> Checked) {
   figure = 2;
   if (RadioButton10 -> Checked) pixels = erosedeSquare3x3(pixels);
   if (RadioButton11 -> Checked) pixels = erosedeSquare5x5(pixels);
 }

 if (RadioButton12 -> Checked) pixels = erosedeBigSquare(pixels);

 setNewBitmap(pixels);
 MainForm -> Panel1 -> Visible = false;
 makeBrightGraph(0);
}
```

Листинг Б.10 – Реализация методов математической морфологии (замыкание)

```
void __fastcall TMainForm::erosedillClick(TObject *Sender)
{
  Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
  bufferBitmap -> Assign(pBitmap);
  std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);

  MainForm -> Panel1 -> Visible = true;

  if (RadioButton6 -> Checked) {
    if (RadioButton8 -> Checked) {
      pixels = erosedeCircle3x3(pixels);
      pixels = dilCircle3x3(pixels);
    }
    if (RadioButton7 -> Checked) {
      pixels = erosedeCircle5x5(pixels);
      pixels = dilCircle5x5(pixels);
    }
  }

  if (RadioButton9 -> Checked) {
    if (RadioButton10 -> Checked) {
      pixels = erosedeSquare3x3(pixels);
      pixels = dilSquare3x3(pixels);
    }
    if (RadioButton11 -> Checked) {
      pixels = erosedeSquare5x5(pixels);
      pixels = dilSquare5x5(pixels);
    }
  }

  if (RadioButton12 -> Checked) {
    pixels = erosedeBigSquare(pixels);
    pixels = dilBigSquare(pixels);
```

```
  }

  setNewBitmap(pixels);
  MainForm -> Panel1 -> Visible = false;
  makeBrightGraph(0);
}
```

Листинг Б.11 – Реализация методов математической морфологии (размыкание)

```
void __fastcall TMainForm::dilleroseClick(TObject *Sender)
{
  Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
  bufferBitmap -> Assign(pBitmap);
  std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);

  MainForm -> Panel1 -> Visible = true;

  if (RadioButton6 -> Checked) {
    if (RadioButton8 -> Checked) {
      pixels = dilCircle3x3(pixels);
      pixels = erosedeCircle3x3(pixels);
    }
    if (RadioButton7 -> Checked) {
      pixels = dilCircle5x5(pixels);
      pixels = erosedeCircle5x5(pixels);
    }
  }

  if (RadioButton9 -> Checked) {
    if (RadioButton10 -> Checked) {
      pixels = dilSquare3x3(pixels);
      pixels = erosedeSquare3x3(pixels);
    }
    if (RadioButton11 -> Checked) {
      pixels = dilSquare5x5(pixels);
      pixels = erosedeSquare5x5(pixels);
    }
  }

  if (RadioButton12 -> Checked) {
    pixels = dilBigSquare(pixels);
    pixels = erosedeBigSquare(pixels);
  }

  setNewBitmap(pixels);
  MainForm -> Panel1 -> Visible = false;
  makeBrightGraph(0);
}
```

Листинг Б.12 – Реализация градиента Щарра

```
std::vector< std::vector<TColor> > ScharGradient(std::vector<
std::vector<TColor> > pixels) {
  std::vector< std::vector<TColor> > new_pixels = pixels;
  std::vector<TColor> mask; mask.resize(9);
  std::vector<int> b_mask; b_mask.resize(9);
  TColor new_pixel;  TColor f = (TColor) 0;

  for (unsigned i = 1; i < pixels.size() - 1; i++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (unsigned j = 1; j < pixels[i].size(); j++) {
```

```
      mask = getBrightMask(pixels, i, j);
      b_mask = getBrights(mask);

      f = (TColor) abs((3 * b_mask[6] + 10 * b_mask[7] + 3 * b_mask[8]) -
(3 * b_mask[0] + 10 * b_mask[1] + 3 * b_mask[2])) + abs((3 * b_mask[2] +
10 * b_mask[5] + 3 * b_mask[8]) - (3 * b_mask[0] + 10 * b_mask[3] + 3 *
b_mask[6]));
      if (f >= 150) new_pixels[i][j] = RGB(255, 255, 255);
      else new_pixels[i][j] = RGB(0, 0, 0);
    }
  }
  return new_pixels;
}

void __fastcall TMainForm::N23Click(TObject *Sender)
{
  Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
  bufferBitmap -> Assign(pBitmap);
  std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);

  MainForm -> Panel1 -> Visible = true;

  pixels = ScharGradient(pixels);
  setNewBitmap(pixels);
  MainForm -> Panel1 -> Visible = false;
  makeBrightGraph(0);
}
```

Листинг Б.13 – Реализация градиента Собеля

```
std::vector< std::vector<TColor> > SobelsGradient(std::vector<
std::vector<TColor> > pixels) {
  std::vector< std::vector<TColor> > new_pixels = pixels;
  std::vector<TColor> mask; mask.resize(9);
  std::vector<int> b_mask; b_mask.resize(9);
  TColor new_pixel;  TColor f = 0;

  for (int i = 1; i < pixels.size() - 1; i++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int j = 1; j < pixels[i].size() - 1; j++) {
      mask = getBrightMask(pixels, i, j);
      b_mask = getBrights(mask);

      f = abs(b_mask[8] - b_mask[4]) + abs(b_mask[7] - b_mask[5]);

      if (f >= 64) new_pixels[i][j] = RGB(255, 255, 255);
      else new_pixels[i][j] = RGB(0, 0, 0);
    }
  }
  MainForm -> BitBtn1 -> Enabled = true;
  return new_pixels;
}

void __fastcall TMainForm::N22Click(TObject *Sender)
{
  Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
  bufferBitmap -> Assign(pBitmap);
  std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);

  MainForm -> Panel1 -> Visible = true;

  pixels = SobelsGradient(pixels);
  setNewBitmap(pixels);
```

```
  MainForm -> Panel1 -> Visible = false;
  makeBrightGraph(0);
}
```

Листинг Б.14 – Реализация градиента Робертса

```
std::vector< std::vector<TColor> > RobertsGradient(std::vector<
std::vector<TColor> > pixels) {
  std::vector< std::vector<TColor> > new_pixels = pixels;
  std::vector<TColor> mask; mask.resize(9);
  std::vector<int> b_mask; b_mask.resize(9);
  TColor new_pixel;  TColor f = (TColor) 0;

  for (unsigned i = 1; i < pixels.size() - 1; i++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (unsigned j = 1; j < pixels[i].size(); j++) {
      mask = getBrightMask(pixels, i, j);
      b_mask = getBrights(mask);

      f = (TColor) abs((b_mask[6] + 2 * b_mask[7] + b_mask[8]) -
(b_mask[0] + 2 * b_mask[1] + b_mask[2])) + abs((b_mask[2] + 2 * b_mask[5]
+ b_mask[8]) - (b_mask[0] + 2 * b_mask[3] + b_mask[6]));
      if (f >= 150) new_pixels[i][j] = RGB(255, 255, 255);
      else new_pixels[i][j] = RGB(0, 0, 0);
    }
  }
  return new_pixels;
}

void __fastcall TMainForm::N21Click(TObject *Sender)
{
  Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
  bufferBitmap -> Assign(pBitmap);
  std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);

  MainForm -> Panel1 -> Visible = true;

  pixels = RobertsGradient(pixels);
  setNewBitmap(pixels);
  MainForm -> Panel1 -> Visible = false;
  makeBrightGraph(0);
}
```

Листинг Б.15 – Реализация метода Лапласа

```
std::vector< std::vector<TColor> > Laplas(std::vector<
std::vector<TColor> > pixels, int background) {
  std::vector< std::vector<TColor> > new_pixels = pixels;
  std::vector<TColor> mask; mask.resize(5);
  std::vector<int> b_mask; b_mask.resize(5);
  int f;
  for (int i = 1; i < pixels.size() - 1; i++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int j = 1; j < pixels[i].size() - 1; j++) {
      mask = getBrightMask3Circle(pixels, i, j);
      b_mask = getBrights(mask);
      f = b_mask[0] + b_mask[1] + b_mask[3] + b_mask[4] - 4 * b_mask[2];
      if (f > 7) new_pixels[i][j] = RGB(255, 255, 255);
      else new_pixels[i][j] = RGB(background, background, background);
    }
  }
  return new_pixels;
}
```

```
void __fastcall TMainForm::N24Click(TObject *Sender)
{
  Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
  bufferBitmap -> Assign(pBitmap);
  std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);

  MainForm -> Panel1 -> Visible = true;
  pixels = Laplas(pixels, 128);
  setNewBitmap(pixels);
  MainForm -> Panel1 -> Visible = false;
  makeBrightGraph(0);
}
```

Листинг Б.16 – Реализация комплексной обработки для улучшения визуального качества РГ

```
vector< vector<TColor> > quality_imporvement(String gradient) {
  Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
  bufferBitmap -> Assign(pBitmap);
  std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);
  vector< vector<TColor> > new_pixels = pixels;

  MainForm -> Panel1 -> Visible = true;

  //Laplas(raw)
  new_pixels = Laplas(pixels, 32);

  //raw + Laplas
  new_pixels = add_image(new_pixels, pixels);

  //Scharr(raw)
  vector< vector<TColor> > schar_pixels = pixels;
  if (gradient == "Schar") schar_pixels = ScharGradient(pixels);
  else schar_pixels = SobelsGradient(pixels);

  //(Laplas(raw) + raw) * Scharr(raw)
  new_pixels = multiply_image(new_pixels, schar_pixels);

  //(Laplas(raw) + raw) + Schar(raw) + raw
  new_pixels = add_image(new_pixels, pixels);

  //gamma-correct or eq.
  if (define_treshold -> RadioButton1 -> Checked) {
    if (define_treshold -> CheckBox1 -> Checked) global_gamma_name =
128.0 / (256.0 - autoTreshold());
    else global_gamma_name = StrToFloat(define_treshold -> Edit1 ->
Text);
    new_pixels = power_processing_scaled(new_pixels, 1,
global_gamma_name);
  } else {
    new_pixels = eq(pixels);
  }


  MainForm -> Panel1 -> Visible = false;
  return new_pixels;
}

void __fastcall TMainForm::N31Click(TObject *Sender)
{
  define_treshold -> ShowModal();
  if (define_treshold -> ModalResult == mrOk) {
```

```
    vector< vector<TColor> > new_pixels = quality_imporvement("Scharr");
    setNewBitmap(new_pixels);
    makeBrightGraph(0);
  }
}

void __fastcall TMainForm::N29Click(TObject *Sender)
{
  define_treshold -> ShowModal();
  if (define_treshold -> ModalResult == mrOk) {
    vector< vector<TColor> > new_pixels = quality_imporvement("Sobel");
    setNewBitmap(new_pixels);
    makeBrightGraph(0);
  }
}
```

Листинг Б.17 – Реализация комплексной обработки для выделения костных структур на КТ

```
vector< vector<TColor> > bones_structure_selector() {
  Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
  bufferBitmap -> Assign(pBitmap);
  std::vector< std::vector<TColor> > pixels = getPixelsMatrix(pBitmap);
  vector< vector<TColor> > new_pixels = pixels;

  MainForm -> Panel1 -> Visible = true;

  //MedianFilter3x3(raw)
  new_pixels = median_filter_3x3(pixels);

  //eq(new_pixels)
  new_pixels = eq(new_pixels);

  //treshold filter(new_pixels)
  int treshold = StrToInt(treshold_value_KT -> Edit1 -> Text);
  for (int i = 1; i < new_pixels.size() - 1; i++) {
    MainForm -> ProgressBar1 -> StepIt();
    for (int j = 1; j < new_pixels[i].size(); j++) new_pixels[i][j] =
thresholdFilter(treshold, new_pixels, i, j);
  }

  //erosede & dillatation
  new_pixels = erosedeSquare3x3(new_pixels);
  new_pixels = dilCircle3x3(new_pixels);

  //dillatation & erosede
  new_pixels = dilCircle3x3(new_pixels);
  new_pixels = erosedeSquare3x3(new_pixels);

  MainForm -> Panel1 -> Visible = false;
  return new_pixels;
}

void __fastcall TMainForm::N32Click(TObject *Sender)
{
  treshold_value_KT -> ShowModal();
  if (treshold_value_KT -> ModalResult == mrOk) {
    vector< vector<TColor> > new_pixels = bones_structure_selector();
    setNewBitmap(new_pixels);
    makeBrightGraph(0);
  }
```

Листинг Б.18 – Реализация оценки качества

```
float getSKO() {
  Graphics::TBitmap *pBitmap = MainForm -> Image2 -> Picture -> Bitmap;
  std::vector< std::vector<TColor> > new_pixels =
getPixelsMatrix(pBitmap);

  Graphics::TBitmap *pBitmap1 = MainForm -> Image1 -> Picture -> Bitmap;
  std::vector< std::vector<TColor> > old_pixels =
getPixelsMatrix(pBitmap1);

  int MN = MainForm -> Image1 -> Width * MainForm -> Image1 -> Height;

  float SKO = 0; int i = 1; int j = 1;
  for (int i = 0; i < new_pixels.size(); i++) {
    for (int j = 0; j < new_pixels[i].size(); j++) SKO +=
(pow(getPixelBright(new_pixels[i][j]) - getPixelBright(old_pixels[i][j]),
2));
  }
  return sqrt(SKO / MN);
}

float getPSNR() {
  Graphics::TBitmap *nBitmap = MainForm -> Image2 -> Picture -> Bitmap;
  int w = nBitmap -> Width; int h = nBitmap -> Height;
  std::vector< std::vector<TColor> > pixels; pixels =
getPixelsMatrix(nBitmap);
  std::vector< std::vector<int> > pixels_brights;
pixels_brights.resize(pixels.size());
  for (int i = 0; i < pixels.size(); i++) {
    pixels_brights[i].resize(pixels[i].size());
    for (int j = 0; j < pixels[i].size(); j++) {
     pixels_brights[i][j] = getBrightOfPixel(pixels[i][j]);
    }
  }

  std::vector<int> brights; brights.resize(255);
  for (int i = 0; i < pixels_brights.size(); i++) {
    for (int j = 0; j < pixels_brights[i].size(); j++)
brights[pixels_brights[i][j]] += 1;
  }

  int globalMAX = brights[return_max_indx(brights)];

  float SKO = getSKO();
  float PSNR;
  if (SKO != 0) PSNR = 20 * log10(globalMAX / sqrt(SKO));
  else PSNR = 0;
  return PSNR;
}

void __fastcall TMainForm::CKO1Click(TObject *Sender)
{
  float SKO = getSKO();
  Application -> MessageBox(FloatToStr(SKO).c_str(), "ÑÊÎ", MB_OK);
}

void __fastcall TMainForm::PSNR1Click(TObject *Sender)
{
  float PSNR = getPSNR();
  Application -> MessageBox(FloatToStr(PSNR).c_str(), "PSNR", MB_OK);
}
```

Выпускная квалификационная работа выполнена мной совершенно самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

«___» _____ _____ г.

---------- ----------

(подпись)