МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
по дисциплине «Объектно-ориентированное проектирование и программирование»

на тему: «3*D*-реконструкция поясничного отдела позвоночника человека методом *dual contouring*»

Исполнитель: студент гр. ИТИ-22
Ковшаров Г. Ю.
Руководитель: доцент
Курочка К. С.

Дата проверки: _____
Дата допуска к защите: _____
Дата защиты: _____

Оценка работы: _____

Подписи членов комиссии
по защите курсовой работы: _____

Гомель 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире наука и медицина всё больше интересуются развитием информационных технологий, особенно в области 3*D*-реконструкции. На данный момент 3*D*-реконструкция органов и тканей является одной из наиболее актуальных и перспективных технологий в современной медицине. Она позволяет создавать трёхмерные модели органов и тканей с высокой точностью, что может быть полезно для планирования хирургических вмешательств, более точного создания имплантов, а также обучения студентов медицинских вузов.

Позвоночник, как и любой другой орган человека, может подвергаться различным видам травм, так как ежедневно он испытывает большие нагрузки. Исследовать патологии позвоночника можно при помощи рентгеновской компьютерной томографии. Однако в некоторых случаях медикаментозное лечение является не эффективным, и пациент нуждается в хирургическом вмешательстве. В подобных ситуациях в целях повышения информативности данных и грамотного хирургического планирования, целесообразно использовать 3*D*-реконструкцию позвоночника.

Также 3*D*-реконструкция позвоночника человека может быть необходима для иных различных целей, начиная от медицинской практики и заканчивая образованием.

Во-первых, 3*D*-реконструкция позволяет получить более наглядное и точное представление о структуре и особенностях позвоночника, что может быть важно для диагностики и лечения различных заболеваний и травм. Например, при планировании хирургического вмешательства 3*D* модель позволяет врачу более точно определить место и глубину разреза, а также предвидеть возможные осложнения.

Во-вторых, 3*D*-реконструкция может быть использована для моделирования и исследования биомеханики позвоночника, что может быть важно для разработки новых методов лечения и профилактики травм.

В-третьих, 3*D* модели могут быть использованы для обучения студентов и врачей, а также для информирования пациентов о структуре и функциях позвоночника.

Кроме того, 3*D*-реконструкция может быть необходима для различных прикладных задач, таких как разработка эргономичных кресел и подголовников, проектирование систем виртуальной реальности для физических упражнений и тому подобное.

Также в целях улучшения качества реконструкции в трёхмерную модель можно добавить пороговый фильтр, который позволяет очищать полученные при помощи компьютерной томографии (КТ) изображения от различных шумов. Данные действия приведут к улучшению точности и информативности

изображения, что в свою очередь облегчит применение алгоритмов 3*D*-реконструкции и улучшит качество результата.

При разработке программы рассматривается процесс 3*D* реконструкции с использованием алгоритма двойных контуров, также известный как *Dual contouring* и предварительной очистки изображения, получаемого при помощи компьютерной томографии поясничного отдела позвоночника человека, от шумов при помощи порогового фильтра.

После тщательного создания детализированной 3*D* модели, она становится неоценимым инструментом, который может быть использован для дальнейшего анализа и исследования позвоночника. Врачи и исследователи могут подробно изучить каждый элемент этой модели, чтобы выявить любые ненормальные участки или области, которые могут указывать на наличие потенциальных заболеваний или травм. Это может включать в себя переломы, деформации, опухоли, воспаления и другие возможные проблемы, которые могут требовать медицинского вмешательства.

Врачи также могут активно использовать эту 3*D*-модель для планирования и симуляции хирургических вмешательств. Это может быть особенно полезно при планировании сложных и чувствительных операций, таких как спинномозговая хирургия или ортопедическая хирургия. Используя точную 3*D*-модель позвоночника, хирурги могут заранее определить наилучший способ проведения операции, а также предвидеть возможные трудности или осложнения, что может значительно улучшить исход операции.

Кроме того, эта модель может быть использована для обучения и образования медицинских специалистов. Студенты и врачи могут изучать анатомию позвоночника и узнавать о различных заболеваниях и их лечении, используя детализированную и точную 3*D*-модель в качестве образовательного инструмента. Это может помочь им лучше понять патологии позвоночника и усовершенствовать свои навыки диагностики и лечения.

Основной целью данной работы является разработка программного обеспечения, которое будет способствовать автоматизации процессов, улучшению качества обработки и анализа полученных врачами компьютерно-томографических изображений. Это в свою очередь позволит повысить точность и скорость выявления патологий, что немаловажно для диагностики и выбора правильного лечения.

Таким образом, использование 3*D*-реконструкции в медицине имеет огромный потенциал и может привести к значительным изменениям в подходе к лечению и диагностике. В будущем, применение данной технологии в медицине может стать одним из ключевых направлений развития медицинской науки и технологий, открывая новые возможности для исследователей, врачей и пациентов.

1 АЛГОРИТМЫ И СРЕДСТВА РАЗРАБОТКИ ПРОГРАММЫ ДЛЯ 3D-РЕКОНСТРУКЦИИ

1.1 Введение в 3D-реконструкцию

Технология 3*D*-реконструкции представляет собой метод трансформации двухмерных изображений в трёхмерные модели. Этот процесс является чрезвычайно сложным и включает в себя детальное извлечение и анализ геометрической информации, такой как глубина, форма и текстура, из входных данных. Основная задача этого метода заключается в создании цифровой модели, которая бы максимально точно воспроизводила структуру и внешний вид исходного, реального объекта.

История 3*D*-реконструкции уходит корнями в эпоху зарождения областей компьютерного зрения и компьютерной графики. Уже в 1960-е годы учёные начали исследования в области извлечения трёхмерной информации из плоских изображений. С тех пор исследование методов 3*D*-реконструкции представляет собой одну из наиболее сложных и важных задач в данной области. 3*D*-реконструкция объектов стала ключевой задачей во многих научных и практических областях, включая медицинскую визуализацию, компьютерное геометрическое моделирование, компьютерную графику и анимацию, компьютерное зрение, вычислительную науку, виртуальную реальность, а также в сфере цифровых медиа.

Современные достижения в области 3*D*-реконструкции, развитие датчиков изображений и прогресс в алгоритмах реконструкции значительно продвинули данную технологию вперёд, расширяя границы возможного в этой области и делая её более точной.

В процессе 3*D*-реконструкции используются множество сложных и разнообразных алгоритмов. Эти алгоритмы по своей сущности уникальны, поскольку они позволяют трансформировать двухмерные изображения в трёхмерные модели. Это преобразование позволяет лучше понять структуру и форму объектов, представленных на изображениях, и предоставляет возможность изучать их в трёхмерном представлении.

1.2 Алгоритмы 3D-реконструкции

Существует множество различных алгоритмов трансформации двумерных изображений в трёхмерные модели, например: *Marching cubes, Marching tetrahedra, Asymptotic decider, Dual contourin.*

Далее будет рассмотрен подробно каждый алгоритм.

Marching cubes – это алгоритм для создания трёхмерных поверхностей из данных, представленных в виде трёхмерной скалярной функции [2]. Данный

алгоритм был разработан в 1987 году Эдвардом Лоренсом и Вейнди Бурком. Алгоритм *Marching cubes* был первым примером в области компьютерной графики, спровоцировавшим скандал в области патентования программного обеспечения. Он был запатентован несмотря на относительную очевидность решения задачи генерации поверхности.

Marching cubes работает с трёхмерной сеткой, разбивая её на кубические ячейки. Алгоритм проходит через скалярное поле, одновременно беря восемь соседних местоположений, формируя таким образом воображаемый куб, а затем определяя многоугольники, необходимые для представления части изоповерхности, проходящей через этот куб. Каждый куб содержит вершины, которые представляют значения функции f в соответствующих точках. Для каждой вершины куба вычисляется значение неявной функции. Отдельные полигоны затем объединяются в желаемую поверхность.

Это делается путем создания индекса для предварительно рассчитанного массива из 2^8 возможных конфигураций многоугольников внутри куба, обрабатывая каждое из восьми скалярных значений как бит в восьми-битном целом числе. Если значение скаляра выше, чем значение изоповерхностного уровня (то есть оно находится внутри поверхности), тогда соответствующий бит устанавливается в единицу, а если оно ниже (снаружи), он устанавливается в ноль. Окончательное значение после проверки всех восьми скаляров является фактическим индексом массива индексов полигонов.

Наконец, каждая вершина сгенерированных многоугольников помещается в соответствующую позицию вдоль края куба путём линейной интерполяции двух скалярных значений, соединенных этим краем.

Данный алгоритм имеет ряд недостатков, связанных с тем, что при аппроксимации изоповерхности могут возникать ошибки, которые влияют на точность представления поверхности.

Далее будут расмотрены некоторые из этих проблем:

Алгоритм *Marching cubes* работает с дискретными данными, разбивая пространство на кубы. Из-за этого полученные полигоны представляют приближённую аппроксимацию изоповерхности. В результате поверхность может быть недостаточно точной, особенно при низком разрешении данных;

В некоторых случаях алгоритм может столкнуться с неоднозначностью. Например, если предположить, что углы помечены положительно, если их значение больше, чем у изолинии, или отрицательно, если оно меньше, то либо положительные углы разделены двумя изолиниями, либо положительные углы находятся в основной части квадрата, а отрицательные углы разделены двумя изолиниями;

Малые изменения в значениях скалярного поля могут привести к большим изменениям в генерируемых полигонах. Шум в данных может существенно повлиять на точность;

Алгоритм может иметь трудности с представлением сложных геометрических форм.

Позднее был разработан похожий алгоритм, названный Marching *tetrahedra*, который для того, чтобы обойти патент *Marching cubes*, использует вместо кубов тетраэдры.

Marching tetrahedra – это алгоритм в области компьютерной графики для рендеринга неявных поверхностей. Первоначально он был представлен в 1991 году Акио Дои и Акио Койде.

В *Marching tetrahedra* каждый куб разбивается на шесть неправильных тетраэдров путём трёхкратного разрезания куба пополам, разрезая по диагонали каждую из трёх пар противоположных граней. Таким образом, все тетраэдры имеют общую одну из главных диагоналей куба. Вместо двенадцати рёбер куба теперь девятнадцать рёбер: исходные двенадцать, шесть диагоналей граней и главная диагональ. Так же, как и в *Marching cubes*, пересечения этих рёбер с изоповерхностью аппроксимируются линейной интерполяцией значений в узлах сетки.

Соседние кубики имеют общие рёбра соединяющейся грани, включая одну и ту же диагональ. Это важное свойство для предотвращения трещин на визуализированной поверхности, поскольку интерполяция двух различных диагоналей грани обычно даёт несколько разные точки пересечения. Дополнительным преимуществом является то, что при обработке соседнего куба можно повторно использовать до пяти вычисленных точек пересечения. Сюда входят вычисленные нормали поверхности и другие графические атрибуты в точках пересечения.

Каждый тетраэдр имеет шестнадцать возможных конфигураций, которые делятся на три класса: без пересечения, пересечение в одном треугольнике и пересечение в двух (соседних) треугольниках.

Asymptotic decider – это алгоритм, разработанный Нильсоном и Хаманном в 1991 году, который создаёт изоповерхности из заданного скалярного поля. Он был предложен как улучшение алгоритма *Marching cubes* и решение одной из его проблем – неоднозначности, из-за которой может создаваться «плохая» топология, но также данный алгоритм может считаться самостоятельным. *Marching cubes* гарантирует, что полученные многоугольники всегда будут топологически корректными. Это важно для правильного представления поверхностей. Он описывает метод разрешения неоднозначных конфигураций последовательно. Это позволяет избежать ошибок в топологии. Таким образом, *Asymptotic decider* предоставляет более надежное и эффективное решение для создания изоповерхности из скалярных полей, чем *Marching cubes*.

Алгоритм сначала разбивает скалярное поле на равные кубы. Это делается для упрощения обработки и представления данных. Затем на боковых (граничных) поверхностях каждого куба рисуются топологически корректные

контуры. Эти контуры представляют собой изолинии (линии постоянного значения скалярной функции). Контуры затем соединяются в многоугольники и триангулируются. Эти многоугольники могут быть треугольниками или другими простыми многоугольниками. Триангуляция многоугольников позволяет получить топологически корректные поверхности. В итоге все треугольники, полученные из каждого куба, формируют итоговую изоповерхность.

Однако, в процессе создания изоповерхностей возникают неоднозначные случаи, когда диагонально противоположные точки находятся с одной стороны изолинии, но с другой стороны относительно других точек в кубе. В этом случае алгоритм *Asymptotic decider* использует гиперболу для определения, как правильно разделить куб на две области. Благодаря этим действиям данный алгоритм как раз-таки и решает проблему неоднозначности алгоритма *Marching cubes*, из-за которой может создаваться плохая топология.

Dual contouring – это тоже метод создания трёхмерных поверхностей из данных, представленных в виде скалярной функции [1]. Данный алгоритм впервые был опубликован в 2002 году в сборнике работ конференции *SIGGRAPH*, авторами которого являются Тао Ю и Фрэнк Лосассо.

Dual contouring так же работает с трёхмерной сеткой, разбивая её на кубические ячейки. Для каждой ячейки воксельной сетки алгоритм вычисляет вершины сетки внутри ячейки. Эти вершины находятся на контурных линиях, которые соответствуют пересечениям поверхности с ячейкой. Алгоритм соединяет вершины, чтобы получить треугольники или квадраты. Это формирует поверхностную сетку. В отличие от *Marching cubes*, ячейки не могут быть оценены независимо [7]. Должны рассматриваться соседние, чтобы соединить точки и найти полную сетку. Но на самом деле это гораздо более простой алгоритм, чем *Marching cubes*, потому что здесь не так много отдельных случаев. Просто находится каждое ребро со сменой знака и соединяются вершины ячеек, соседних с этим ребром.

В *Marching cubes* нужно было просто вычислить функцию f на сетке. Теперь нам нужна информация о градиенте. Градиент функции f – это мера того, насколько быстро изменяется значение f в данной точке при движении в любом заданном направлении. Обычно он задаётся в виде пары чисел для каждой точки, указывая, насколько изменяется функция при перемещении по оси X или оси Y. Соединение точек через рёбра выполняется с изменением знака функции, используя информацию о градиенте. Так же для определения точки на сетке, в целях улучшения точности, используется функция квадратичной ошибки. Это позволяет *Dual contouring* более точно аппроксимировать поверхности, особенно в областях с резкими углами и краями, так как учитывается изменения функции в окрестности каждой вершины [1].

В рамках данного проекта будет использован алгоритм *Dual contouring*. Был выбран именно этот алгоритм, так как он решает некоторые проблемы алгоритма *Marching cubes* и позволяет получить более естественную модель.

Цель работы заключается в изучении принципов и методов для работы с 3*D*-реконструкцией. Также в работе предполагается создание порогового фильтра для очистки изображения, полученного при помощи компьютерной томографии, от шумов и дальнейшей его реконструкции в 3*D*-модель, при помощи алгоритма *Dual contouring*.

Реализация 3*D*-реконструкции может быть выполнена при помощи широкого спектра разнообразных инструментов и технологий. Это могут быть специализированные программы для обработки изображений, такие как *Adobe Photoshop* или *GIMP*, программное обеспечение для 3*D*-моделирования, например, *Blender* или 3*DS Max*, библиотеки и фреймворки для обработки изображений и 3*D*-моделирования, например, *OpenCV* или *TensorFlow*. Также могут быть использованы различные языки программирования, которые поддерживают работу с графическими данными и имеют библиотеки для работы с 3*D*-реконструкцией.

1.3 Обзор средств разработки программы для 3D-реконструкции

В настоящее время в мире существует несколько сотен реально используемых языков программирования. Для каждого есть своя область применения.

Любой алгоритм, как известно, есть последовательность предписаний, выполнив которые можно за конечное число шагов перейти от исходных данных к результату. В зависимости от степени детализации предписаний обычно определяется уровень языка программирования – чем меньше детализация, тем выше уровень языка.

По этому критерию можно выделить следующие уровни языков программирования:

– машинные языки программирования, воспринимаемые аппаратной частью компьютера (машинные коды);

– машинно-ориентированные языки программирования (ассемблеры);

– языки программирования, которые отражают структуру конкретного типа компьютера;

– машинно-независимые (языки высокого уровня).

Машинные языки и машинно-ориентированные языки – это языки низкого уровня, требующие указания мелких деталей процесса обработки данных. Языки же высокого уровня имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы.

До того, как компьютер сможет выполнить программу, написанную на языке высокого уровня, её приходится переводить на понятный компьютеру язык, то есть машинный код. Этот процесс перевода называют трансляцией, а программу-переводчик – транслятором. Трансляторы делятся на два класса: компиляторы и интерпретаторы.

Компиляция заключается в том, что программа в машинном коде (называемая компилятором) преобразует другую программу, написанную на языке программирования в машинный код. После этого полученный машинный код программы выполняется.

Интерпретация заключается в том, что программа в машинном коде (интерпретатор) записывает файл программы во внутреннюю память и начинает её построчно выполнять.

Языки высокого уровня делятся на несколько типов:

– процедурные;

– логические;

– объектно-ориентированные.

Процедурные языки программирования предоставляют возможность программисту определять каждый шаг в процессе решения задачи. Особенность таких языков программирования состоит в том, что задачи разбиваются на шаги и решаются шаг за шагом. Используя процедурный язык, программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов.

Логические языки программирования – языки программирования, позволяющие выполнить описание проблемы в терминах фактов и логических формулах, а собственно решение проблемы выполняет система с помощью механизмов логического вывода.

Объектно-ориентированные языки программирования, в основе которых лежит понятие объекта, сочетающего в себе данные и действия над ними. Программа на объектно-ориентированном языке, решая некоторую задачу, по сути, описывает часть мира, относящуюся к этой задаче. Описание действительности в форме системы взаимодействующих объектов естественнее, чем в форме взаимодействующих процедур.

В настоящее время для разработчиков предлагается широкое разнообразие прикладных программных продуктов, позволяющее автоматизировать различные задачи проектировщиков и разработчиков.

В целях создания порогового фильтра и реконструкции из двухмерного пространства в трёхмерное, также может быть использовано множество языков программирования разных уровней и типов. Ниже приведен обзор самых часто используемых языков программирования.

Python – это высокоуровневый, мультипарадигменный, интерпретируемый язык программирования с динамической типизацией и автоматическим управлением памятью, к преимуществам которого относят высокую производительность программных решений и структурированный, хорошо читаемый код. Синтаксис *Python* максимально облегчен, что позволяет выучить его за сравнительно короткое время.

Python широко используется для разработки веб-приложений, работы с графикой, создания искусственного интеллекта, научных вычислений, системного администрирования. Благодаря своей простоте и читаемости, *Python* особенно популярен среди начинающих программистов и в образовательной сфере.

Python обладает богатой стандартной библиотекой и большим количеством сторонних модулей, что позволяет легко интегрировать различные системы и обрабатывать данные разных форматов. Он поддерживает множество парадигм программирования, включая объектно-ориентированное, процедурное и функциональное программирование.

Программы на *Python* выполняются в среде *Python Interpreter*, которая обеспечивает кроссплатформенную совместимость и возможность работы на большинстве операционных систем. Данный язык также является основой для многих популярных фреймворков, таких как *Django* для веб-разработки и *scikit-learn* для машинного обучения.

На сегодняшний день *Python* является одним из самых востребованных языков программирования в мире, благодаря своей гибкости, мощности и широкому применению в различных областях.

Язык поддерживает множество расширенных функций, таких как генераторы, декораторы и контекстные менеджеры, которые делают код более компактным и удобным для поддержки. Он продолжает активно развиваться, и с каждым новым релизом предлагает всё больше возможностей для разработчиков.

Java – язык программирования общего назначения. Относится к объектно-ориентированным языкам программирования, к языкам с сильной типизацией. Синтаксис языка *Java* похож на синтаксис других си-подобных языков. Программы на *Java* могут быть транслированы в байт-код, выполняемый на виртуальной *Java*-машине (*JVM*) – программе, обрабатывающей байт-код и передающей инструкции оборудованию, как интерпретатор, но с тем отличием, что байт-код, в отличие от текста, обрабатывается значительно быстрее. Реализация проекта была начата на языке *C++*, но вскоре возник ряд проблем, наилучшим средством борьбы с которыми было изменение самого инструмента – языка программирования. Стало очевидным, что необходим платформо-независимый язык программирования, позволяющий создавать программы,

которые не приходилось бы компилировать отдельно для каждой архитектуры и можно было бы использовать на различных процессорах под различными операционными системами.

Три ключевых элемента объединились в технологии языка *Java*, которые будут описаны далее.

Во-первых, *Java* предоставляет для широкого использования свои апплеты (*applets*) – небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы *Web*. Апплеты *Java* могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы *HTML*;

Во-вторых, *Java* показывает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты;

В-третьих, *Java* предоставляет программисту богатый набор классов и объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов.

C# – объектно-ориентированный, компилируемый, со статической типизацией, автоматическим управлением мусора, ориентированный на компоненты язык программирования. *C#* активно используется для реконструкции изображений из двухмерного пространства в трёхмерное и работы с нейронными сетями. Этот язык программирования обладает мощными объектно-ориентированными возможностями, которые идеально подходят для создания сложных систем. *C#* предоставляет разработчикам набор инструментов для эффективной работы с графикой и алгоритмами машинного обучения, что делает его отличным выбором для проектов, связанных с компьютерным зрением и искусственным интеллектом.

Когда говорят *C#*, нередко имеют в виду технологии платформы *.NET* (*Windows Forms, WPF, ASP.NET, Xamarin*). И, наоборот, когда говорят *.NET*, нередко имеют в виду *C#*. Однако, хотя эти понятия связаны, отождествлять их неверно. Язык *C#* был создан специально для работы с фреймворком *.NET*, однако само понятие *.NET* несколько шире.

Программы *C#* выполняются в *.NET*, виртуальной системе выполнения, вызывающей общеязыковую среду выполнения (*CLR*) и набор библиотек классов. Среда *CLR* – это реализация общеязыковой инфраструктуры языка (*CLI*), являющейся международным стандартом, от корпорации Майкрософт.

CLI является основой для создания сред выполнения и разработки, в которых языки и библиотеки прозрачно работают друг с другом.

На сегодняшний момент язык программирования *C#* один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нём пишутся самые различные приложения: от небольших десктопных программок до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И *C#* продолжает активно развиваться, и с каждой новой версией появляется всё больше интересных функциональностей.

Исходя из вышеупомянутых преимуществ, язык программирования *C#* является наиболее предпочтительным для реконструкции изображений из двухмерного пространства в трёхмерное. Он содержит большое количество системных функций и библиотек, которые облегчают работу в этой области. Именно поэтому он был выбран для разработки приложения, реализующего алгоритм *Dual contouring*.

2 ПРОЕКТИРОВАНИЕ ПРОГРАММЫ ДЛЯ 3D-РЕКОНСТРУКЦИИ

2.1 Анализ и разбор задачи 3D-реконструкции

Для успешного решения поставленной задачи необходимо разработать целостный программный комплекс. Этот комплекс должен быть создан на языке программирования *C#*, который известен своей эффективностью и надёжностью. Данный комплекс будет использовать снимки, полученные при помощи компьютерной томографии, позвоночника человека, чтобы создать его трёхмерную реконструкцию. Но перед этим, снимки должны быть предварительно очищены при помощи порогового фильтра, чтобы убедиться, что все излишние или ненужные элементы будут удалены перед началом процесса реконструкции. В результате, программный комплекс позволит создать точные и детализированные трёхмерные модели позвоночника на основе КТ-снимков.

Основные требования для разрабатываемого программного продукта:

– в качестве входных данных принимать снимки, полученные при помощи компьютерной томографии в формате файла *DICOM*;

– фильтровать послойно КТ снимки с помощью порогового фильтра;

– реконструировать 3*D*-модель поясничного отдела позвоночника человека при помощи метода *Dual contouring*.

Решаемая задача основывается на использовании КТ снимков. Эти снимки представлены в формате файла *DICOM*, который широко используется в медицинской сфере для хранения, обмена и просмотра диагностических изображений. Пример того, как выглядят эти данные, можно увидеть на рисунке 2.1.

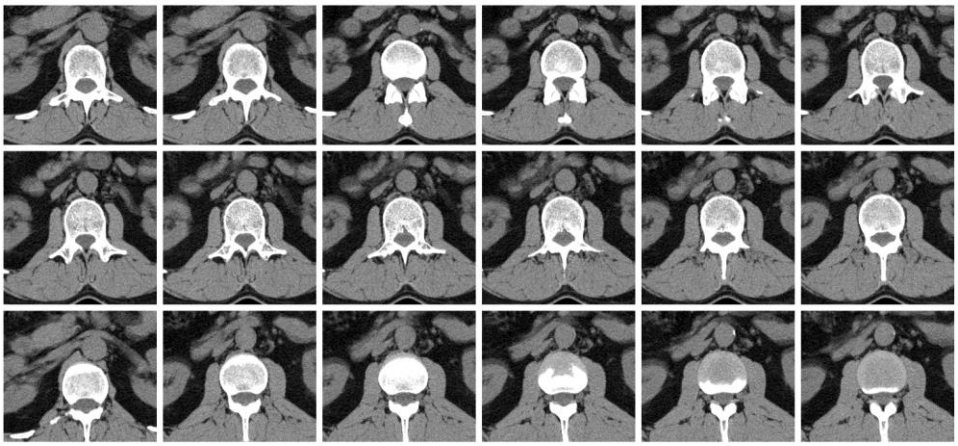

Рисунок 2.1 – Пример входных данных

Именно с подобными данными будет производиться дальнейшая работа, включая анализ и преобразование их для дальнейшего использования в задаче.

Также стоит определить общую структурную схему поставленной задачи, которая включает в себя следующие этапы:

– модуль для получения входных данных;

– модуль для обработки входных данных;

– модуль для реконструкции 3*D* модели.

2.2 Анализ и определение входных данных

Исходные данные представлены в формате файлов *DICOM*, которые были получены с помощью компьютерной томографии.

Формат *DICOM*, полное название которого – *Digital Imaging and Communications in Medicine*, был впервые представлен в 1980-х годах. Это было время, когда возникла острая необходимость в стандарте для обмена медицинскими изображениями, чтобы обеспечить совместимость между различными медицинскими устройствами. Создание этого формата стало результатом совместных усилий Американского колледжа радиологии и Национальной ассоциации производителей электротехники, которые признали наличие проблемы и необходимость стандартизации в области медицинской изображения.

С момента своего введения формат *DICOM* потерпел множество изменений и улучшений. Это позволило адаптировать его к новым технологиям изображений и улучшить его возможности обмена данными. Сегодня *DICOM* является глобальным стандартом для обмена и хранения медицинских изображений. Он используется в более чем 180 странах и поддерживается тысячами производителей медицинского оборудования по всему миру.

Этот формат охватывает огромный спектр медицинской диагностики, включая, но не ограничиваясь такими видами как рентген, компьютерная томография, магнитно-резонансная томография, ультразвук и множество других видов диагностического оборудования. Благодаря его универсальности и гибкости, стандарт *DICOM* все больше внедряется и становится неотъемлемой частью медицинской отрасли. Однако, формат *DICOM* не просто служит в качестве хранилища изображений. Он также сохраняет критически важную информацию о пациенте и процедуре сканирования, что делает его неотъемлемой составляющей электронных медицинских записей. Эффективное использование формата *DICOM* способствует обмену информацией между различными системами и устройствами, что является ключевым элементом для обеспечения качественного и своевременного медицинского обслуживания.

Файлы *DICOM* имеют строго определённую структуру и состоят из двух основных частей: заголовка и тела файла. Заголовок содержит информацию о пациенте и диагностической процедуре, включая, но не ограничиваясь, такими данными как имя пациента, дату рождения, пол, а также дату и время проведения

сканирования, тип сканирования и другие подробности. Тело файла, в свою очередь, содержит изображения, которые могут быть представлены в виде двухмерных снимков или трехмерных реконструкций, в зависимости от конкретного исследования и используемого оборудования. Это отражает гибкость формата *DICOM*, способного адаптироваться к различным потребностям медицинской диагностики.

2.3 Описание алгоритма обработки входных данных

Перед тем как использовать входные данные для создания детализированной 3D-модели, их необходимо тщательно обработать. В данном случае, обработка включает в себя фильтрацию данных с помощью порогового фильтра. В качестве входных данных используются КТ снимки. Эти снимки могут содержать множество лишней информации, включая воздух, мягкие ткани, полости и прочее. Эта информация является излишней, поскольку нашей конечной целью является создание 3*D*-модели позвоночного отдела человека. Если при процессе реконструкции эта информация останется на изображениях, то это приведет к искажению конечной 3*D*-модели. В результате, модель будет включать в себя ненужные элементы, вместо того чтобы воспроизводить только структуру костей. В таком случае, по полученной модели будет сложно сделать точные и правильные выводы, что существенно уменьшит её практическую ценность.

Пиксели, собранные при помощи самых современных и точных технологий компьютерной томографии, тщательно переводятся в шкалу Хаунсфилда [5]. Это процесс, который требует большой точности и внимания к деталям. Шкала Хаунсфилда известна своей надёжностью и точностью, и она широко используется в медицинской практике для визуальной и количественной оценки плотности различных структур, которые могут быть визуализированы с помощью метода компьютерной томографии. Важно подчеркнуть, что визуальное отражение этой плотности на дисплее устройства представляет собой чёрно-белый спектр изображения. Это позволяет врачам более четко и ясно видеть все детали, что невероятно важно для точной диагностики.

В медицинской практике широко используется понятие диапазона единиц шкалы, которые также известны как «денситометрические показатели» или, если использовать английский термин, *Hounsfield units*. Эти единицы представляют собой числовые значения, соответствующие степени ослабления рентгеновского излучения различными анатомическими структурами организма. Диапазон этих значений варьируется от -1024 до +1024, что позволяет достаточно точно оценивать состояние определенных тканей и органов. Однако следует отметить, что в практическом применении эти значения могут несколько отличаться на разных устройствах. Это связано с их индивидуальными характеристиками,

спецификацией и природой работы, поэтому при использовании данных единиц всегда необходимо учитывать это обстоятельство.

Средним показателем в шкале Хаунсфилда является 0 *HU*, что соответствует плотности воды. Отрицательные величины шкалы указывают на присутствие воздуха и жировой ткани, в то время как положительные значения указывают на наличие мягких тканей, костной ткани и более плотного вещества, такого как металл. Это ключевая информация, которая обеспечивает нам понимание того, как рентгеновский луч проходит через различные материалы и структуры в теле. Фактически, это помогает расшифровывать различные области изображения рентгеновских снимков, и позволяет медицинским профессионалам устанавливать диагноз и проводить эффективное лечение.

Средние денситометрические показатели шкалы Хаунсфилда представлены на рисунке 2.2, что обеспечивает более наглядное представление об этой сложной, но важной системе.

Вещество	HU
Воздух	−1000
Жир	−120
Вода	0
Мягкие ткани	+40
Кости	+400 и выше

Рисунок 2.2 Средние денситометрические показатели шкалы Хаунсфилда

Данные показатели, безусловно, являются фундаментом, на котором строится работа порогового фильтра. Этот фильтр играет критически важную роль в процессе обработки изображений, обеспечивая качественный и точный анализ. Суть работы порогового фильтра заключается в детальном рассмотрении каждого пикселя на изображении. Это означает, что он тщательно просматривает значение каждого пикселя, определяя его важность и роль в общем контексте изображения [4].

В этом процессе, главной стадией является присваивание нового значения пикселю, основываясь на его текущем значении. Для более четкого понимания, представим, что пиксель имеет значение 500 по шкале Хаунсфилда. В данном случае, система автоматически меняет значение этого пикселя на максимально возможное, что может равняться, например, 1024. Это действие позволяет увеличить контрастность изображения. В то же время, если система обнаруживает, что пиксель имеет значение, например, 300, она тогда меняет его значение на минимально возможное, что может быть, например, -1024. Такой подход обеспечивает более высокую детализацию изображения, что в свою очередь повышает точность диагностики.

Это преобразование, основанное на применении порогового фильтра, играет критическую роль в отсеивании всей избыточной информации, которая может служить препятствием для правильной интерпретации изображения. Оно позволяет делать упор только на ключевых, наиболее важных и ценных элементах, минимизируя шум и нерелевантные данные.

В результате такой обработки, на снимке остаётся только информация о костях. Эта информация оказывается является очень важной в контексте создания более точной 3*D*-реконструкции пациента. Это значительно упрощает процесс диагностики, позволяя медицинским работникам быстрее и точнее выявить возможные патологии на ранней стадии развития. Это, в свою очередь, обеспечивает возможность назначить соответствующее, наиболее эффективное лечение, способствуя улучшению качества жизни пациентов.

На рисунке 2.3, а, представлено исходное КТ изображение, а на рисунке 2.3, б, предполагаемый результат работы порогового фильтра.

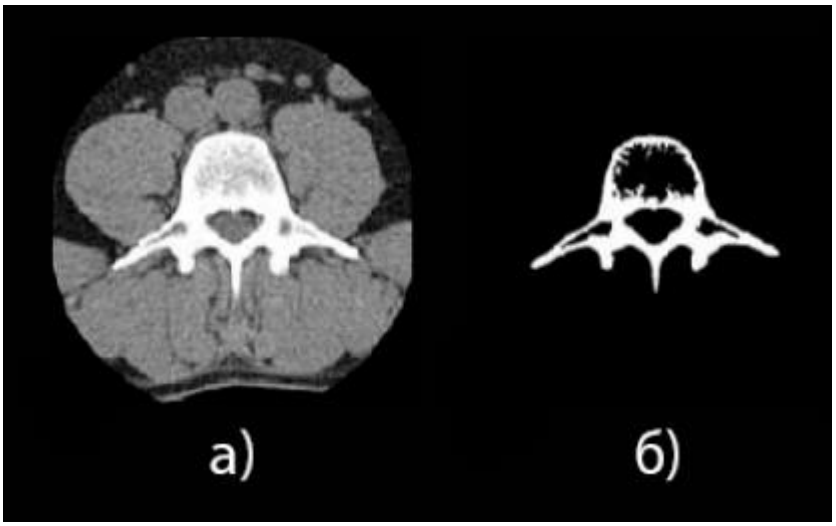

Рисунок 2.3 – КТ изображение: а –исходное; б – предполагаемый результат работы порогового фильтра

Как можно заметить, на предполагаемом результате работы порогового фильтра отсутствует любая лишняя или ненужная информация, что очень важно для точности и чёткости результата. Вместо этого, результат сосредоточен исключительно на представлении кости. Это гарантирует, что вся информация, которая в нём содержится, является релевантной и ценной. Это ключевой аспект, который позволит более точно и детально сделать 3*D*-реконструкцию, что в свою очередь повышает качество и точность конечного продукта.

2.4 Разбор алгоритма Dual contouring для реконструкции 3D модели

Так как изображения, полученные с помощью компьютерной томографии, отображают слои позвоночника, то их можно объединить в трёхмерную модель,

что является следующим шагом работы программы, после обработки входных данных.

В процессе 3*D*-реконструкции используется метод двойного контурирования, который представляет собой точную технологию для создания детальных трехмерных изображений объекта, включая все его особенности и детали. Этот метод полезен для врачей, так как он помогает им лучше понимать структуру позвоночника и выявлять возможные проблемы, что способствует более точной диагностике.

Алгоритм двойного контурирования – это уникальный и сложный метод создания трёхмерных изображений. Он включает в себя сбор и обработку большого количества данных о границах объекта и его сложной внутренней структуре. Эта информация получается с использованием точек и нормалей к поверхности, которые играют важную роль в получении наиболее точного представления о форме объекта. Блок-схема алгоритма данного метода приведена в приложении Б.

В начале своей работы алгоритм *Dual contouring* разделяет пространство на ячейки. Это делается для упрощения обработки и анализа данных, так как каждая ячейка может быть рассмотрена отдельно. Затем алгоритм находит ячейки, где функция меняет знак вдоль любого ребра, то есть находит границы объекта.

Следующий шаг – определение точки внутри ячейки и соединение точек из соседних ячеек в полигоны. Стоит отметить, что в алгоритме *Dual contouring* невозможно обрабатывать ячейки по отдельности, так как для формирования полигона рассматриваются несколько соседних ячеек. Это является ключевым моментом в работе этого алгоритма и позволяет достичь высокой точности в воссоздании формы объекта.

Далее, для определения координат вершины внутри ячейки, *Dual contouring* применяет несколько сложных методов. Сначала алгоритм находит исходную функцию поверхности в каждой вершине ячейки. Затем между вершинами, у которых значения функции разных знаков, алгоритм вычисляет координаты точек пересечения ребер ячейки с поверхностью. Это делается с использованием линейной аппроксимации значений функции в вершинах ячейки.

После этого, для каждой точки пересечения ребра ячейки с поверхностью рассчитывается градиент функции поверхности. Этот шаг критически важен, так как градиент функции поверхности позволяет получить информацию о направлении наибольшего изменения функции, что помогает в определении формы объекта.

В итоге, искомая вершина размещается в точке, которая наилучшим образом соответствует найденным градиентам. Это позволяет алгоритму *Dual contouring* создавать детальные и точные трёхмерные изображения, обеспечивая высокую степень точности в воспроизведении формы и деталей объекта.

Метод *Dual contouring* обладает важным преимуществом, которое заключается в его уникальной способности точно воспроизводить острые и

плоские границы объекта. Эта ключевая особенность является решением проблемы, которую представлял его предшественник, известный как алгоритм *Marching cubes*. Точность воспроизведения деталей и высокое качество 3*D*-моделей, которые он обеспечивает, делают этот алгоритм особенно ценным в широком спектре применений, включая различные задачи, связанные с 3*D*-моделированием.

Вместе с тем, несмотря на всю его эффективность и превосходную точность, важно отметить, что алгоритм требует высококачественных входных данных. Необходимо провести тщательную предварительную подготовку данных для достижения наилучших результатов. Этот процесс включает в себя сбор, обработку и анализ данных, что может потребовать значительных усилий и времени.

Рисунок 2.4 наглядно демонстрирует сравнение контуров, полученных с использованием алгоритмов *Dual contouring* и *Marching cubes*, подтверждая преимущества первого над последним.

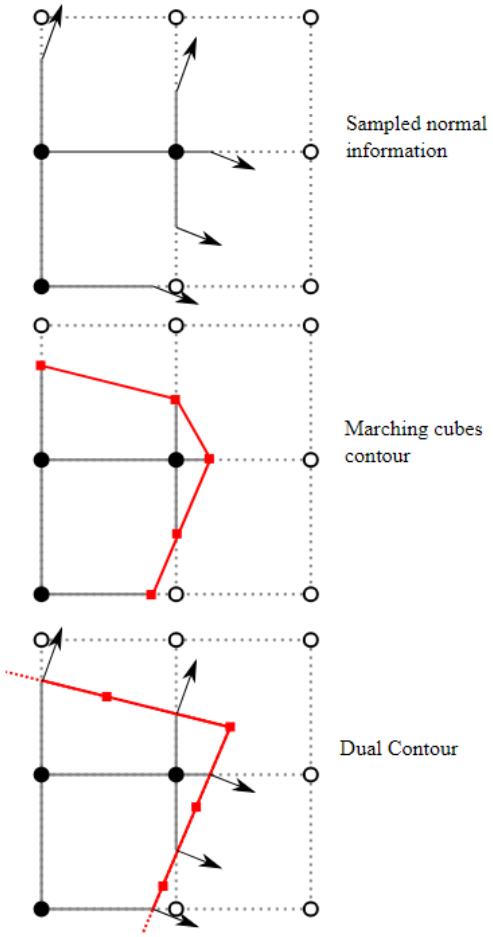

Рисунок 2.4 – Сравнение результирующего контура алгоритмов *Dual contouring* и *Marching cubes*

Как видно на рисунке 2.4, результаты, полученные с использованием *Dual contouring*, обеспечивают гораздо более четкое и точное представление о структуре объекта, что подтверждает преимущества этого алгоритма перед *Marching cubes*. Это является критически важным для создания детализированных и точных 3*D*-моделей, особенно в контексте медицинских исследований и диагностики.

2.5 Проектирование архитектуры приложения

Прежде всего, выбор архитектуры приложения – это важное решение, которое определяет его эффективность, гибкость и масштабируемость. Правильный выбор архитектуры позволяет не только обеспечить корректную работу приложения, но и сделать его легко модифицируемым и адаптируемым к изменяющимся требованиям и условиям. Он способствует более эффективному и качественному взаимодействию между элементами системы, облегчает процесс тестирования и отладки, и упрощает процесс разработки. Вообще, выбор архитектуры должен быть продуманным и основываться на требованиях и целях проекта.

Поэтому, основываясь на требованиях поставленной задачи, была разработана архитектура, представляющая собой два ключевых слоя: *UI* (*User Interface*) слой и слой бизнес-логики.

UI слой, или слой пользовательского интерфейса, отвечает за взаимодействие пользователя с приложением. Он обеспечивает чёткую и понятную обратную связь, позволяя пользователю легко и интуитивно управлять процессами приложения.

Слой бизнес-логики, с другой стороны, охватывает все ключевые аспекты функциональности приложения. Он отвечает за выполнение основного функционала, такого как обработка данных, выполнение алгоритмов и обеспечение результатов, которые затем отображаются на *UI* слое.

Выделение этих двух слоев позволяет обеспечить четкую структуру и логику работы приложения, упрощая процесс разработки и поддержки. Это также способствует гибкости приложения, позволяя легко модифицировать или расширять функционал в любой из слоев без значительного влияния на другой.

Таким образом, эффективное разграничение на слой пользовательского интерфейса и слой бизнес-логики обеспечивает гибкость и модульность приложения. Это позволяет легко вносить изменения в один слой, не затрагивая работу другого, что существенно упрощает процесс разработки и последующей поддержки.

Но также важное значение имеет структура приложения и взаимодействие между различными модулями и слоями. Важно обеспечить эффективное и надежное взаимодействие между модулями, чтобы обеспечить бесперебойную работу всего приложения. Это также поможет упростить процесс отладки и найти, и устранить возможные ошибки.

2.6 Разработка структуры приложения

Приложение будет структурировано в виде трёх основных модулей, каждый из которых будет иметь уникальную и незаменимую роль, внося свой ценный вклад в общую цель приложения.

Первый модуль, который можно считать основой всего приложения, будет отвечать за логику работы порогового фильтра. Этот модуль будет применяться для тщательной обработки входных данных. Он будет осуществлять фильтрацию и отбор значимой информации из полученных изображений, что является критическим и необходимым шагом перед переходом к следующему, не менее важному этапу – 3*D*-реконструкции.

Второй модуль, будет реализовывать логику работы алгоритма реконструкции *Dual contouring*. Этот модуль будет отвечать за создание детализированных трёхмерных моделей на основе обработанных данных, полученных от первого модуля. Благодаря высокой точности и детализации этого алгоритма, второй модуль будет способствовать созданию наиболее точных и детальных 3*D*-моделей.

Третий модуль будет представлять собой пользовательский интерфейс, который будет служить связующим звеном между первыми двумя модулями. Этот модуль будет обеспечивать удобное и интуитивное взаимодействие пользователя с приложением. Он обеспечит доступ к функционалу первых двух модулей и предоставит возможность просмотра и анализа входных и обработанных изображений, а также сохранения полученной 3*D*-модели в файл формата *stl*.

Для удобства использования, интерфейс будет разделён на несколько основных секций, включая область просмотра изображений, панель инструментов для управления процессом обработки и анализа. Также будет предусмотрена возможность настройки параметров обработки, таких как пороговые значения для фильтрации. Это позволит каждому пользователю адаптировать работу приложения под свои специфические задачи и потребности.

3 ЭКСПЛУАТИЦИЯ И ТЕСТИРОВАНИЕ ПРОЛОЖЕНИЯ

3.1 Загрузка входных данных

Первый этап выполнения разработанной программы – это процесс загрузки данных, который был тщательно продуман и разработан для оптимальной эффективности. В приложении разработан интуитивно понятный интерфейс, который был создан с учётом потребностей пользователя, чтобы обеспечить удобное и понятное взаимодействие с программой.

При запуске программы пользователю необходимо выбрать несколько опций. Во-первых, он должен выбрать папку, в которой находятся исходные данные. Это должны быть срезы, полученные при помощи компьютерной томографии, и имеющие специфическое расширение файла *DICOM*. Во-вторых, он должен указать папку, куда будут загружены обработанные при помощи порогового фильтра данные. В-третьих, пользователь должен установить пороговое значение, которое будет использоваться в процессе обработки данных.

На рисунке 3.1 представлена визуализация начального окна программы.

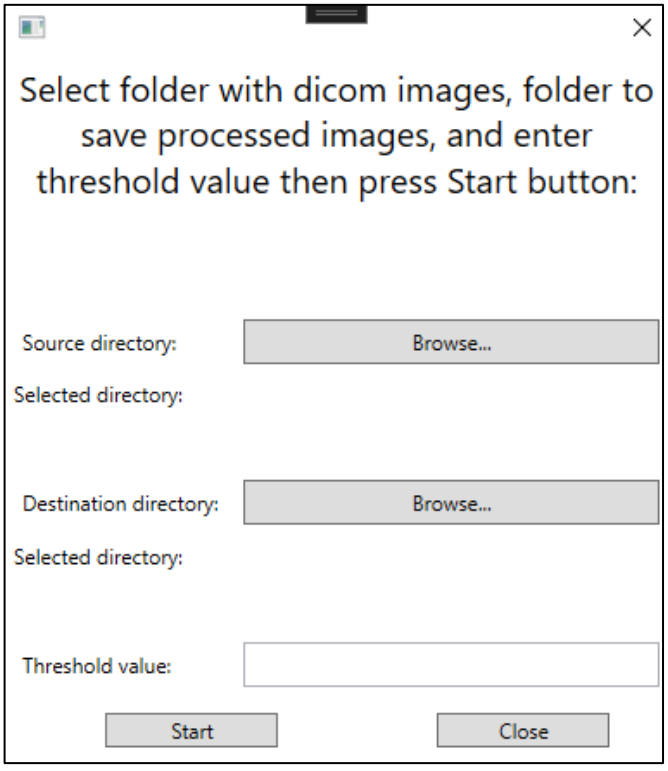

Рисунок 3.1 – Визуализация начального окна программы

Можно заметить, что в рамках обеспечения удобства для пользователя, был реализован функционал автоматического выбора папок. Таким образом, пользователю не требуется самостоятельно вводить пути к папкам. Это

достигнуто благодаря созданию специальных кнопок, которые открывают диалоговое окно. В этом окне предлагается выбрать нужную папку для дальнейшей работы. Более того, разработанная система предусматривает наличие поля для ввода порогового значения для порогового фильтра. Это позволяет каждому пользователю ввести своё уникальное значение, которое наиболее подходит для его специфического набора изображений, полученных при помощи компьютерной томографии. Это, в свою очередь, обеспечивает большую гибкость и настройку под индивидуальные требования каждого пользователя.

Важно отметить одну существенную деталь, связанную с выбором папок для хранения данных. Путь к указанной папке обязательно не должен содержать в себе русские символы. В противном случае, в процессе выполнения программы возникнет ошибка, что может привести к прерыванию работы и потере данных. Поэтому, настоятельно рекомендуется убедиться в отсутствии русских символов в пути к папке до начала работы программы.

В результате выполнения этого этапа исходные срезы позвоночного отдела человека будут успешно загружены в оперативную память системы для дальнейших этапов работы программы.

3.2 Обработка и визуализация данных

Следующим этапом в этом процессе, который следует после успешной загрузки ценных данных в оперативную память компьютера, является их детальная обработка с использованием специального инструментас – порогового фильтра. Это достаточно важный процесс, который обеспечивает высококачественную, детальную и тщательную подготовку данных для их дальнейшего анализа и интерпретации. Все данные, которые были загружены, включая каждый отдельный файл, который был найден и выбран в целевой директории, обязательно пропускаются через этот мощный фильтр. Этот процесс фильтрации позволяет значительно улучшить качество данных, избавившись от ненужной, избыточной или несоответствующей информации и тем самым сохраняя только те данные, которые релевантны и пригодны для дальнейшего анализа.

После того, как данные проходят через фильтр, они сохраняются в другую папку, которую выбирает пользователь. Это дает пользователю возможность управлять и организовывать свои данные эффективнее, так как обработанные данные теперь могут быть легко доступны в любое время, в любом месте.

Затем, сразу после обработки, исходные и полученные данные визуализируются на экране в новом открывшемся окне. Это позволяет пользователю иметь визуальное представление о том, как работает программа, наглядно увидеть разницу между исходными и обработанными данными, а также убедиться в корректности выполненной обработки.

В этом же окне, пользователь также может увидеть, где именно в общем объеме данных находится определенный срез данных. Это помогает

пользователю лучше понять и визуализировать структуру изучаемого объекта, а также дает ему возможность лучше интерпретировать полученные результаты.

На рисунке 3.2 представлено открывшееся окно программы, которое визуализирует исходные и полученные срезы.

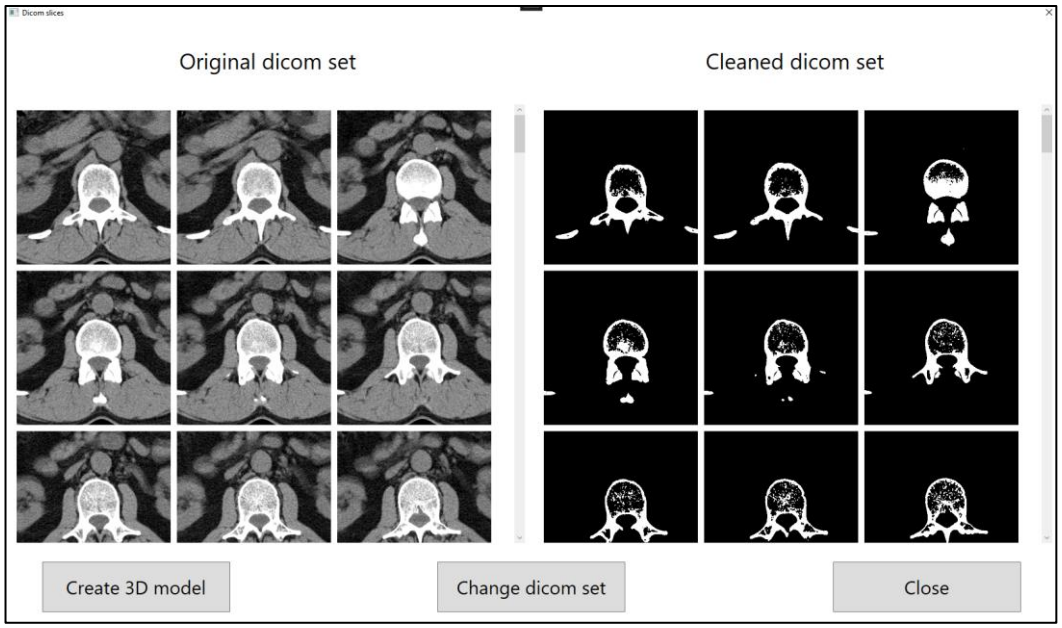

Рисунок 3.2 – Окно, визуализирующее исходные и полученные срезы

На данном этапе работы приложения, все обработанные данные уже будут загружены в указанную пользователем папку, где они будут полностью доступны для независимого просмотра и анализа вне контекста данной программы.

Вместе с этим, пользователю предоставляется выбор одного из трёх возможных действий, которые могут быть выполнены в этой точке:

– сгенерировать детализированную 3*D*-модель на основе обработанных срезов данных, что позволит пользователю лучше оценить и визуализировать полученные результаты;

– поменять набор срезов на другой;

– прекратить работу приложения, если пользователь решит, что все необходимые действия уже выполнены.

Данные действия пользователь может осуществить путём нажатия на соответствующие кнопки в графическом интерфейсе окна программы.

3.3 Реконструкция 3D модели

После того как данные прошли через процесс обработки с использованием порогового фильтра, пользователь получает возможность запустить процесс 3*D*-реконструкции модели, нажав на соответствующую кнопку. Это позволяет преобразовать обработанные срезы в трехмерную форму.

Этот сложный процесс превращения двухмерных изображений в трехмерную модель необходим для более точного представления исследуемого объекта. Полученная 3*D*-модель затем сохраняется в указанном пользователем месте в формате *stl*. Этот формат выбран для того, чтобы обеспечить возможность просмотра модели с использованием различных программных обеспечений, что дает пользователю большую гибкость при работе с полученной моделью.

На рисунке 3.3 будет представлен пример полученной, при помощи разработанной программы, 3*D*-модели в программной системе для обработки трёхмерных сеток *Meshlab*.

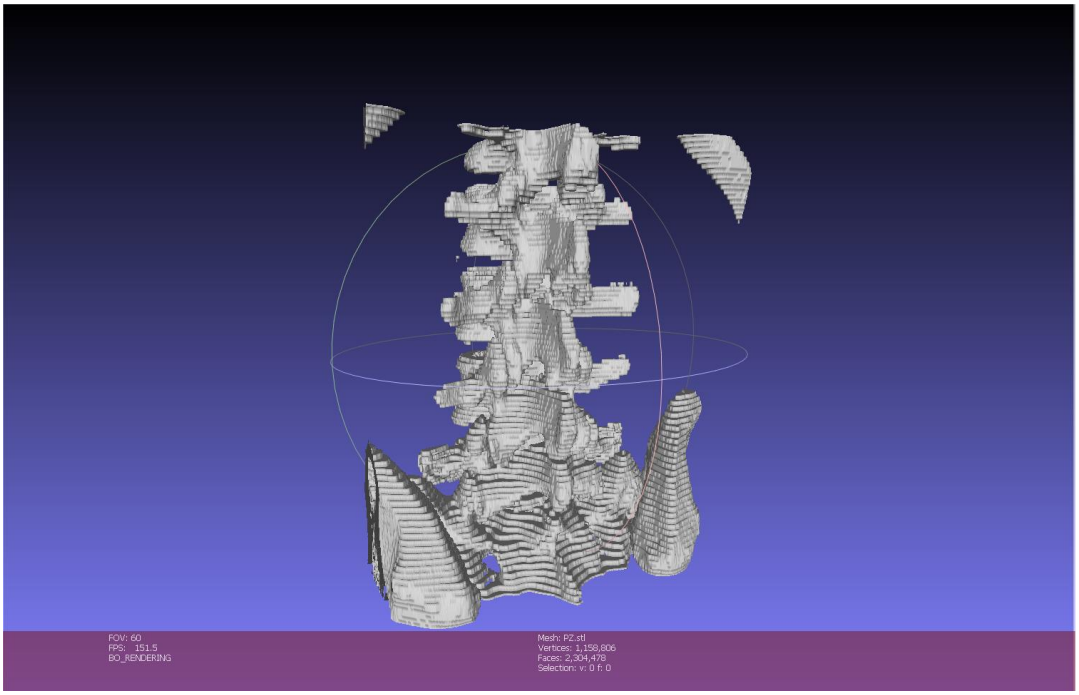

Рисунок 3.3 – Визуализация полученной в ходе работы разработанной программы 3*D* модели

После завершения процесса 3*D*-реконструкции, пользователь может провести анализ и проверку полученных результатов. Это включает в себя оценку качества и точности 3*D*-модели, определение степени её соответствия исходным изображениям, а также идентификацию возможных ошибок и неточностей.

3.4 Тестирование приложения

Для уверенности в корректности работы программного обеспечения, важно провести процесс верификации. Этот процесс позволяет подтвердить, что программа функционирует в соответствии с заявленными параметрами и не содержит ошибок.

В данном контексте, можно провести верификацию качества визуализации КТ изображений. Для этого можно использовать специализированные программные решения, разработанные для просмотра срезов, полученных с помощью компьютерной томографии. Эти программы обеспечивают детальный анализ КТ изображений, что позволяет оценить качество работы программы по визуализации этих изображений.

MicroDicom – это бесплатное программное обеспечение, разработанное специально для просмотра медицинских изображений в формате *DICOM*. Это прогрессивное приложение обеспечивает пользователям все необходимые функции и инструменты для работы с *DICOM* файлами. Это включает в себя открытие, просмотр, анализ и сохранение медицинских изображений в удобной и эффективной манере.

Кроме того, *MicroDicom* поддерживает различные форматы изображений, что делает его ещё более универсальным и гибким в использовании. Он предлагает широкий спектр инструментов для редактирования изображений, включая изменение яркости и контрастности, масштабирование, измерение и многое другое.

Эти функции позволяют улучшить качество просмотра изображений, что в свою очередь повышает эффективность работы с медицинскими данными. В общем, *MicroDicom* – это мощный инструмент, который может значительно облегчить работу медицинских специалистов и исследователей в области здравоохранения.

Теперь нужно определить набор срезов, при помощи которого будет производиться квалификация. После чего необходимо визуализировать выбранный набор в *MicroDicom*.

Пример визуализации набора КТ изображений представлен на рисунке 3.4.

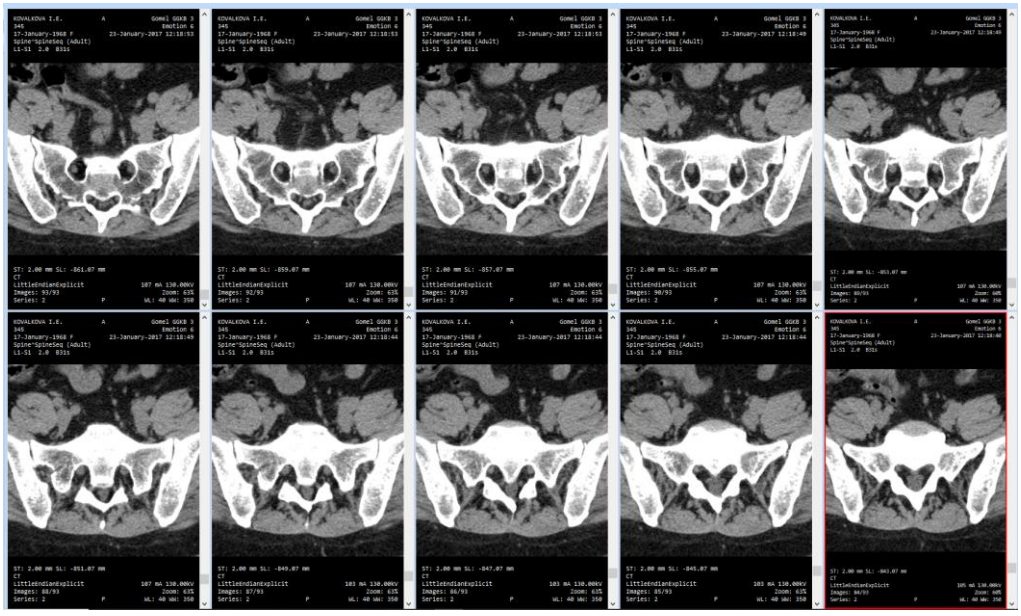

Рисунок 3.4 – Визуализация набора КТ изображений в программе *MicroDicom*

Можно заметить, что в результате было получено высококачественное отображение срезов.

Кроме того, важно отметить, что каждое изображение содержит разнообразные надписи. Эти надписи представляют собой информацию, которая была извлечена прямо из *DICOM* файлов. В числе представленной информации можно найти данные о пациенте, такие как его возраст и пол, а также сведения о медицинском учреждении, где было проведено исследование, и многие другие существенные детали.

Если рассмотреть какой-либо срез отдельно, используя данные из *DICOM* файла, возможно более удобно визуализировать данную информацию о пациенте, нежели при просмотре всего набора вместе. Это даёт возможность более глубоко и всесторонне анализировать исходные данные, что, в свою очередь, позволяет получить более полное и детализированное представление о состоянии пациента. Это подчеркивает ценность и важность использования *DICOM* файлов в медицине.

Визуализация отдельного КТ изображения в программе *MicroDicom* показана на рисунке 3.5.

Рисунок 3.5 – Визуализация отдельного КТ изображения

Важно отметить, что для обеспечения удобства пользовательского взаимодействия и просмотра вся необходимая информация тщательно и продуманно расположена в различных углах экрана. Это обеспечивает быстрый доступ к ключевым элементам и помогает пользователям легко находить то, что им нужно, не тратя лишнее время на поиск.

Теперь, для верификации разработанной программы выведем в ней данный набор срезов и сравним качество с выводом в *MicroDicom*.

На рисунке 3.6 представлена визуализация выбранного набора КТ изображений при помощи разработанной программы.

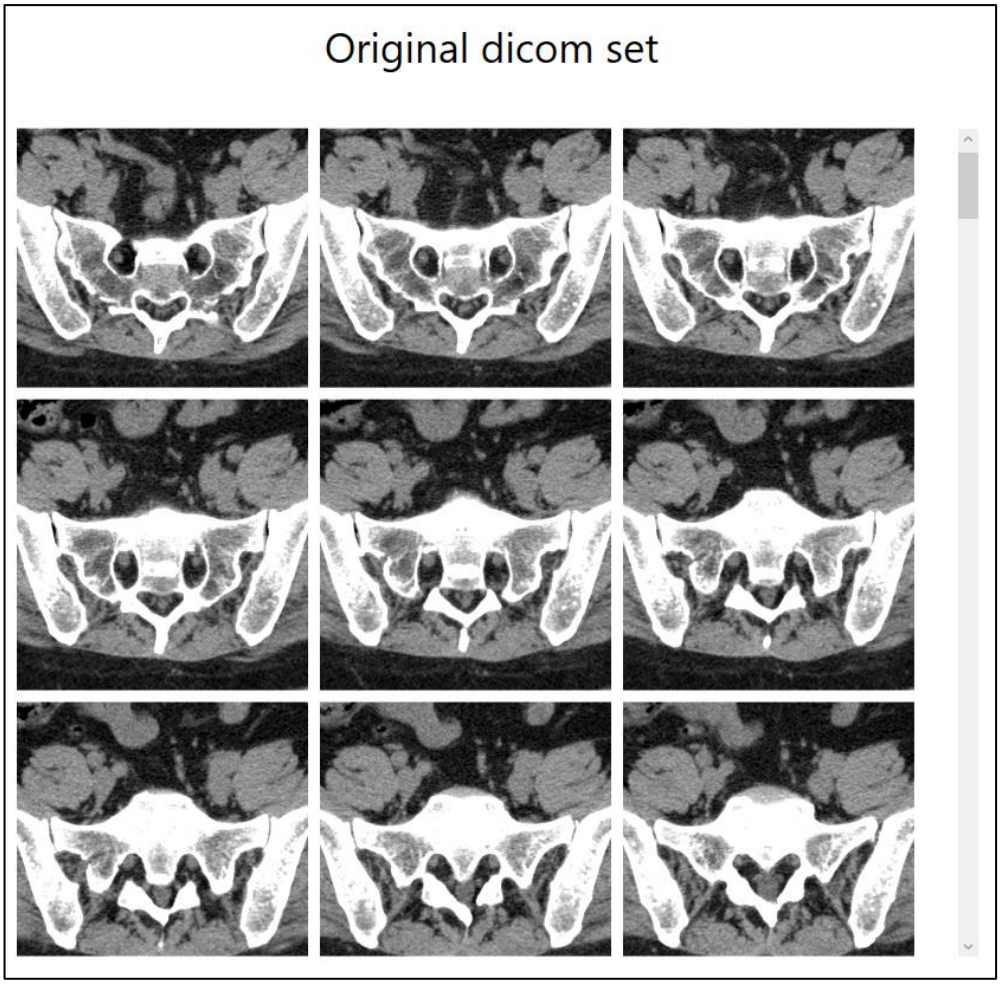

Рисунок 3.6 – Визуализация набора КТ изображений в разработанной программе

При детальном анализе работы разработанной программы, можно обнаружить, что она способна выводить томографические изображения с высокой степенью идентичности тем, которые выводит специализированная программа *MicroDicom*. Это подтверждение высокого качества визуализации, которое было достигнуто в результате тщательной разработки программного обеспечения. Благодаря столь высокому качеству визуализации, пользователи могут делать точные и надёжные выводы при интерпретации полученных изображений. Это крайне важно в таких областях, как медицина и другие, где точность интерпретации может иметь критическое значение.

Таким образом, разработанная программа прошла верификацию и в итоге можно сказать, что она способна обеспечить высокую степень точности и надежности, что делает её весьма полезной в ряде сфер применения.

3.5 Верификация приложения

Для проверки корректной работоспособности разработанного программного обеспечения можно провести измерение времени работы алгоритма построения 3*D* модели, а также его ресурсоёмкость.

Верификация проводится на устройстве со следующими характеристиками:

– видеокарта *NVIDIA GeForce RTX* 3050 *Laptop GPU*;

– процессор 12*th Gen Intel* (*R*) *Core* (*TM*) *i*7-12700*H*;

– частота процессора 2.70 *GHz*;

– количество ядер 14;

– оперативная память 16 гигабайт.

Перед выполнением тестов стоит определить потребление ресурсов приложением в бездейственном состоянии:

– использование процессора 0%;

– использование оперативной памяти 730 мегабайт;

– использование графического процессора 0%.

Тестирование будет проводиться на исходных данных разного объема, представленных в виде *DICOM* файлов. Эти файлы представляют собой срезы позвоночного отдела человека и варьируются по количеству – 66, 94, 100, 108 штук. Это позволит оценить, как изменение объема входных данных влияет на производительность алгоритма.

Первый тест был проведён с набором, состоящим из 66 срезов. Итоги теста показали следующие результаты:

– время реконструирования 11 секунды;

– использование процессора 7.8%;

– использование оперативной памяти 835 мегабайт;

– использование графического процессора 0%.

Второй тест был проведён с набором, состоящим из 94 срезов. Итоги теста показали следующие результаты:

– время реконструирования 8.3 секунды;

– использование процессора 8.1%;

– использование оперативной памяти 879 мегабайт;

– использование графического процессора 0%.

Третий тест был проведён с набором, состоящим из 100 срезов. Итоги теста показали следующие результаты:

– время реконструирования 8.8 секунды;

– использование процессора 8.3%;

– использование оперативной памяти 890 мегабайт;

– использование графического процессора 0%.

Четвёртый тест был проведён с набором, состоящим из 108 срезов. Итоги теста показали следующие результаты:

– время реконструирования 12.6 секунды;

– использование процессора 8.5%;

– использование оперативной памяти 939 мегабайт;

– использование графического процессора 0%.

По результату проведённых тестов становится очевидным, что объём входных данных оказывает непосредственное влияние на скорость работы и потребление ресурсов программой при выполнении алгоритма 3*D*-реконструкции. Однако это далеко не единственный фактор, который играет роль в эффективности работы программы.

Производительность программного обеспечения также существенно зависит от специфических параметров представленных *DICOM* файлов. К таким параметрам относятся размер изображений, их разрешение, количество цветов, уровень контрастности и другие менее очевидные характеристики, которые могут влиять на сложность обработки данных. Эти аспекты могут стать причиной различных изменений в скорости и ресурсоемкости работы программы.

Несмотря на всё вышеуказанные факторы и потенциальные сложности, разработанное программное обеспечение показывает достаточно стабильную работу. Это подтверждает его эффективность и способность справляться с поставленной задачей.

ЗАКЛЮЧЕНИЕ

В процессе выполнения данного курсового проекта было успешно разработано десктопное приложение. Это приложение предоставляет возможности для визуализации и последующей обработки набора снимков, полученных с помощью компьютерной томографии, что позволяет реконструировать их в трехмерную модель.

В рамках проекта также была выполнена верификация программы. В результате, было подтверждено, что разработанная программа функционирует корректно и эффективно, что подтверждается высоким качеством её работы.

Разработанное приложение имеет ряд значительных преимуществ. Во-первых, оно обеспечивает высококачественную визуализацию КТ изображений, что позволяет пользователям лучше понять и интерпретировать полученные данные. Во-вторых, оно предлагает мощный набор инструментов для обработки данных, включая пороговый фильтр и возможность 3*D*-реконструкции при помощи метода *Dual contouring*. Это позволяет пользователям преобразовывать собранные данные в полезную, практически применимую информацию. В-третьих, приложение имеет интуитивно понятный интерфейс, что облегчает его использование и сокращает время на обучение. В-четвертых, оно позволяет пользователям сохранять обработанные данные и 3*D* модели для дальнейшего анализа и использования.

Разработанное программное обеспечение обладает значительной практической значимостью и потенциалом. Это инновационное решение может быть активно использовано медицинскими специалистами различных направлений для анализа КТ-снимков и реконструкции трёхмерных моделей. Это даёт возможность более точно и детально диагностировать различные болезни, предлагать наиболее эффективные методы лечения и многое другое.

Кроме своего применения в медицине, приложение может быть полезно в научной и исследовательской области. Оно предоставляет учёным инструмент для визуализации и анализа сложных структур, упрощая их работу и расширяя возможности для проведения новых исследований.

В конечном итоге программа открывает новые горизонты для улучшения диагностики, лечения и исследовательской работы. Это доказательство силы и возможностей цифровых технологий, которые становятся всё более важными в мире.

СПИСОК ИПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Dual Contouring Tutorial. – Электрон. данные. – Режим доступа: https://www.boristhebrave.com/2018/04/15/dual-contouring-tutorial/. – Дата доступа: 16.03.2024.

2. Курочка, К. С. Адаптированные алгоритмы Dual Contouring и Marching Cubes для 3D-реконструкции поясничного отдела позвоночника человека / К. С. Курочка, Т. С. Семенченя // Доклады БГУИР. – 2023. – Т. 21, № 6. – С. 99–105.

3. Семенченя, Т. С. Реконструкция 3D-модели поясничного отдела позвоночника человека методом Dual Contouring на основе анализа цифровых КТ-изображений / Т. С. Семенченя, К. С. Курочка // Информационные технологии и системы – 2022 (ИТС – 2022): матер. междунар. науч. конф., Минск, 23 ноября 2022 г. / Минск: Белорус. гос. ун-т информ. и радиоэлектр. – 2022. – С. 163–165.

4. Semenchenya, T. S. Construction of an Individual Geometric 3D Model of the Lumbar Spine of a Person Based on the Analysis of Medical Images / T. S. Semenchenya, K. S. Kurochka // Открытые семантические технологии проектирования интеллектуальных систем (OSTIS-2020): сб. науч. тр. Минск: Белор. гос. ун-т информ. и радиоэлектр., 2020. Вып. 4. С. 291–297

5. Шкала Хаунсфилда. – Электрон. данные. – Режим доступа: https://studfile.net/preview/5165071/page:5/. – Дата доступа: 26.04.2024.

6. Li, Z. Adaptive dual contouring for high-quality surface reconstruction from point clouds / Li, Z., Zhang, Y., Zhou, C., & Gao, Y. // Материалы междунар., конф., IEEE по автоматизированному проектированию, Сан-Диего, 4-8 ноября 2018 г. / С. 59-70

7. High-Resolution and Efficient Neural Dual Contouring for Surface Reconstruction from Point Clouds. – Электрон. данные. – Режим доступа: https://www.mdpi.com/2072-4292/15/9/2267. – Дата доступа: 25.03.2024.

ПРИЛОЖЕНИЕ А
(обязательное)

Листинг программы

DicomThresholdFilter.cs

```
using Dicom;
using Dicom.Imaging;
using Dicom.IO.Buffer;
using System;
using System.IO;

namespace ThresholdFilter
{
   public class DicomThresholdFilter
   {
      /// <summary>
      /// Очистка dicom срезов при помощи порогового фильтра
      /// </summary>
      /// <param name="pathToSourceDirectory">Путь к папке с исходными dicom срезами</param>
      /// <param name="pathToDestinationDirectory">Путь к папке для сохранения dicom срезов</param>
      /// <param name="thresholdValue">Пороговое значение</param>
      /// <exception cref="DirectoryNotFoundException">Если папки с dicom срезами или для сохранения срезов не существует</exception>
      public void CleanTheDicomKit(string pathToSourceDirectory, string pathToDestinationDirectory, short thresholdValue)
      {
         if (!Directory.Exists(pathToSourceDirectory))
         {
            throw new DirectoryNotFoundException($"Directory '{pathToSourceDirectory}' not found.");
         }

         if (!Directory.Exists(pathToDestinationDirectory))
         {
            throw new DirectoryNotFoundException($"Directory '{pathToDestinationDirectory}' not found.");
         }

         foreach (string pathToDicomFile in Directory.EnumerateFiles(pathToSourceDirectory))
         {
            var dicomFile = DicomFile.Open(pathToDicomFile);

            var pixelData = DicomPixelData.Create(dicomFile.Dataset);

            var originalPixelBytes = pixelData.GetFrame(0).Data;

            var originalPixelShorts = new short[originalPixelBytes.Length / sizeof(short)];

            Buffer.BlockCopy(originalPixelBytes, 0, originalPixelShorts, 0, originalPixelBytes.Length);

            var modifiedPixelBytes = CleanPixels(originalPixelShorts, thresholdValue);

            var modifiedPixelBytesBuffer = new MemoryByteBuffer(modifiedPixelBytes);

            dicomFile.Dataset.AddOrUpdatePixelData(DicomVR.OB, modifiedPixelBytesBuffer);

            string fileName = Path.GetFileName(pathToDicomFile);

            dicomFile.Save($"{pathToDestinationDirectory}\\{fileName}");
         }
      }
```

```
        /// <summary>
        /// Очистка исходных пикселей
        /// </summary>
        /// <param name="originalPixelShorts">Исходные пиксели</param>
        /// <param name="thresholdValue">Пороговое значение</param>
        /// <returns>Очищенные пиксели</returns>
        private byte[] CleanPixels(short[] originalPixelShorts, short thresholdValue)
        {
            var modifiedPixelBytes = new byte[originalPixelShorts.Length * sizeof(short)];

            for (int i = 0; i < originalPixelShorts.Length; i++)
            {
                var newPixelValueShort = originalPixelShorts[i] >= thresholdValue ? (short)2000 : (short)0;
                var newPixelValueBytes = BitConverter.GetBytes(newPixelValueShort);
                Buffer.BlockCopy(newPixelValueBytes, 0, modifiedPixelBytes, i * sizeof(short), sizeof(short));
            }

            return modifiedPixelBytes;
        }
    }
}
```

DualContouringFilter.cs

```
using Kitware.VTK;
using System.IO;

namespace DualContouring
{
    public class DualContouringFilter
    {
        /// <summary>
        /// Реконструкция dicom срезов в 3D модель
        /// </summary>
        /// <param name="pathToDicomSlicesDirectory">Путь к папке с dicom срезами</param>
        /// <param name="stlSavePath">Путь для сохранения stl файла</param>
        /// <exception cref="DirectoryNotFoundException">Если папки с dicom срезами не существует</exception>
        public void ReconstructionDicomSlices(string pathToDicomSlicesDirectory, string stlSavePath)
        {
            if (!Directory.Exists(pathToDicomSlicesDirectory))
            {
                throw new DirectoryNotFoundException($"Directory '{pathToDicomSlicesDirectory}' not found.");
            }

            var reader = new vtkDICOMImageReader();

            reader.SetDirectoryName(pathToDicomSlicesDirectory);

            reader.Update();

            var imageData = reader.GetOutput();

            var contourFilter = new vtkContourFilter();

            contourFilter.SetInput(imageData);

            contourFilter.SetValue(0, 250);

            contourFilter.Update();

            var contourFilterOutput = contourFilter.GetOutputPort();

            WriteToStl(contourFilterOutput, stlSavePath);
```

```
        }
        /// <summary>
        /// Запись 3D модели в stl файл
        /// </summary>
        /// <param name="contourFilterOutput">Результат работы Dual Contouring алгоритма</param>
        /// <param name="stlSavePath">>Путь для сохранения stl файла</param>
        private void WriteToStl(vtkAlgorithmOutput contourFilterOutput, string stlSavePath)
        {
            if (File.Exists(stlSavePath))
            {
                File.Delete(stlSavePath);
            }

            var stlWriter = new vtkSTLWriter();

            stlWriter.SetInputConnection(contourFilterOutput);

            stlWriter.SetFileName(stlSavePath);

            stlWriter.Write();
        }
    }
}
```

MainWindow.xaml

```
<Window x:Class="UserUi.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:UserUi"
        mc:Ignorable="d"
        Height="450" Width="400"
        ResizeMode="NoResize">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>

        <TextBlock x:Name="ShortInfoTextBlock" TextWrapping="Wrap" Grid.Row="0" Grid.Column="0"
Grid.ColumnSpan="4" Margin="5" TextAlignment="Center" FontSize="20"
                   Text="Select folder with dicom images and folder to save processed images, then press Start button:"/>

        <Label Content="Source directory:" Grid.Row="2" Grid.Column="0" Margin="5"/>
        <Button x:Name="BrowseSourceButton" Content="Browse..." Grid.Row="2" Grid.Column="1"
Grid.ColumnSpan="3" Margin="5" Click="BrowseSourceButton_Click"/>
        <TextBlock x:Name="SourceDirectoryTextBlock" TextWrapping="Wrap" Grid.Row="3" Grid.Column="0"
Grid.ColumnSpan="4" Margin="5" Height="70" Text="Selected directory: "/>
```

```
        <Label Content="Destination directory:" Grid.Row="4" Grid.Column="0" Margin="5"/>
        <Button x:Name="BrowseDestinationButton" Content="Browse..." Grid.Row="4" Grid.Column="1" Grid.ColumnSpan="3" Margin="5" Click="BrowseDestinationButton_Click"/>
        <TextBlock x:Name="DestinationDirectoryTextBlock" TextWrapping="Wrap" Grid.Row="5" Grid.Column="0" Grid.ColumnSpan="4" Margin="5" Height="70" Text="Selected directory: "/>

        <Label Content="Threshold value:" Grid.Row="6" Grid.Column="0" Margin="5"/>
        <TextBox x:Name="ThresholdValueTextBox" TextWrapping="Wrap" Grid.Row="6" Grid.Column="1" Grid.ColumnSpan="3" Margin="5"/>

        <Button x:Name="StartButton" Content="Start" Grid.Row="7" Grid.Column="0" Grid.ColumnSpan="2" Margin="10" Width="100" Click="StartButton_Click"/>
        <Button x:Name="CloseButton" Content="Close" Grid.Row="7" Grid.Column="2" Grid.ColumnSpan="2" Margin="10" Width="100" Click="CloseButton_Click"/>
    </Grid>
</Window>
```

MainWindow.xaml.cs

```
using System;
using System.IO;
using System.Windows;
using System.Windows.Forms;
using MessageBox = System.Windows.MessageBox;

namespace UserUi
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
        private void BrowseSourceButton_Click(object sender, RoutedEventArgs e)
        {
            var dialog = new FolderBrowserDialog();
            if (dialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                SourceDirectoryTextBlock.Text = "Selected directory: " + dialog.SelectedPath;
            }
        }
        private void BrowseDestinationButton_Click(object sender, RoutedEventArgs e)
        {
            var dialog = new FolderBrowserDialog();
            if (dialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                DestinationDirectoryTextBlock.Text = "Selected directory: " + dialog.SelectedPath;
            }
        }
        private void StartButton_Click(object sender, RoutedEventArgs e)
        {
            var sourceDirectory = SourceDirectoryTextBlock.Text.Substring(SourceDirectoryTextBlock.Text.IndexOf(":") + 1).Trim();
            var destinationDirectory = DestinationDirectoryTextBlock.Text.Substring(SourceDirectoryTextBlock.Text.IndexOf(":") + 1).Trim();

            if (sourceDirectory == destinationDirectory)
            {
                MessageBox.Show("The paths to the directories cannot be the same.", "Invalid paths", MessageBoxButton.OK, MessageBoxImage.Error);
```

```
            return;
        }

        if (!Directory.Exists(sourceDirectory))
        {
            MessageBox.Show("Source directory does not exist.", "Invalid directory", MessageBoxButton.OK,
MessageBoxImage.Error);
            return;
        }

        if (!Directory.Exists(destinationDirectory))
        {
            MessageBox.Show("Destination directory does not exist.", "Invalid directory", MessageBoxButton.OK,
MessageBoxImage.Error);
            return;
        }

        if (Int16.TryParse(ThresholdValueTextBox.Text, out short thresholdValue))
        {
            var processingWindow = new ProcessingWindow();
            processingWindow.Show();
            processingWindow.StartProcessing(sourceDirectory, destinationDirectory, thresholdValue);
            Close();
        }
        else
        {
            MessageBox.Show("Enter valid threshold value.", "Invalid threshold value", MessageBoxButton.OK,
MessageBoxImage.Error);
            return;
        }

    }
    private void CloseButton_Click(object sender, RoutedEventArgs e) => Close();
  }
}
```

ProcessingWindow.xaml

```
<Window x:Class="UserUi.ProcessingWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:UserUi"
    mc:Ignorable="d"
    Title="Dicom slices" Height="1000" Width="1700"
    ResizeMode="NoResize">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="*"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*"/>
      <ColumnDefinition Width="*"/>
      <ColumnDefinition Width="*"/>
      <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>

    <TextBlock Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="2" FontSize="35"
HorizontalAlignment="Center" VerticalAlignment="Center" Text="Original dicom set" />
```

```
    <TextBlock Grid.Row="0" Grid.Column="2" Grid.ColumnSpan="2" FontSize="35" HorizontalAlignment="Center" VerticalAlignment="Center" Text="Cleaned dicom set" />

    <ScrollViewer Grid.Column="0" Grid.Row="1" Grid.ColumnSpan="2" Margin="10" Height="700" VerticalScrollBarVisibility="Auto" CanContentScroll="True" HorizontalScrollBarVisibility="Disabled">
        <WrapPanel x:Name="OriginalImagesPanel" Orientation="Horizontal" ItemWidth="256" ItemHeight="256" Margin="5" ScrollViewer.CanContentScroll="True"/>
    </ScrollViewer>

    <ScrollViewer Grid.Column="2" Grid.Row="1" Grid.ColumnSpan="2" Margin="10" Height="700" VerticalScrollBarVisibility="Auto" CanContentScroll="True" HorizontalScrollBarVisibility="Disabled">
        <WrapPanel x:Name="ProcessedImagesPanel" Orientation="Horizontal" ItemWidth="256" ItemHeight="256" Margin="5" ScrollViewer.CanContentScroll="True"/>
    </ScrollViewer>

    <Button x:Name="Create3DModelButton" Grid.Row="2" Grid.Column="0" Content="Create 3D model" Width="300" Height="80" FontSize="30" Margin="10" Click="Create3DModelButton_Click"/>
    <Button x:Name="ChangeDicomSetButton" Grid.Row="2" Grid.Column="1" Grid.ColumnSpan="2" Content="Change dicom set" Width="300" Height="80" FontSize="30" Margin="10" Click="ChangeDicomSetButton_Click"/>
    <Button x:Name="CloseButton" Grid.Row="2" Grid.Column="3" Content="Close" Width="300" Height="80" FontSize="30" Margin="10" Click="CloseButton_Clock"/>
  </Grid>
</Window>
```

ProcessingWindow.xaml.cs

```
using Dicom;
using Dicom.Imaging;
using DualContouring;
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Forms;
using System.Windows.Media.Imaging;
using ThresholdFilter;
using Image = System.Windows.Controls.Image;
using MessageBox = System.Windows.MessageBox;

namespace UserUi
{
  /// <summary>
  /// Логика взаимодействия для ProcessingWindow.xaml
  /// </summary>
  public partial class ProcessingWindow : Window
  {
    private string _sourceDirectory;
    private string _destinationDirectory;

    public ProcessingWindow()
    {
      InitializeComponent();
    }
    /// <summary>
    /// Обработка dicom срезов
    /// </summary>
    /// <param name="sourceDirectory">Путь к папке с исходными dicom срезами</param>
    /// <param name="destinationDirectory">Путь к папке для сохранения dicom срезов</param>
    public void StartProcessing(string sourceDirectory, string destinationDirectory, short thresholdValue)
    {
```

```
    _sourceDirectory = sourceDirectory;
    _destinationDirectory = destinationDirectory;

    var thresholdFilter = new DicomThresholdFilter();

    try
    {
        thresholdFilter.CleanTheDicomKit(_sourceDirectory, _destinationDirectory, thresholdValue);
    }
    catch (DirectoryNotFoundException ex)
    {
        MessageBox.Show(ex.Message, "Invalid directory", MessageBoxButton.OK, MessageBoxImage.Error);
        ShowMainWindow();
        Close();
        return;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Error);
        ShowMainWindow();
        Close();
        return;
    }

    DrawDicomSet(_sourceDirectory, OriginalImagesPanel);
    DrawDicomSet(_destinationDirectory, ProcessedImagesPanel);
}
/// <summary>
/// Отображение dicom срезов
/// </summary>
/// <param name="pathToDicomSetDirectory">Путь к папке с dicom срезами</param>
/// <param name="panel">Панель в которой будут содержаться изображения</param>
private void DrawDicomSet(string pathToDicomSetDirectory, WrapPanel panel)
{
    foreach (string pathToDicomFile in Directory.EnumerateFiles(pathToDicomSetDirectory))
    {
        var dicomFile = DicomFile.Open(pathToDicomFile);

        var dicomImage = new DicomImage(dicomFile.Dataset);

        using (Bitmap bitmap = dicomImage.RenderImage().AsSharedBitmap())
        {
            var bitmapImage = ConvertBitmapToBitmapImage(bitmap);

            var image = new Image
            {
                Margin = new Thickness(5),
                Source = bitmapImage
            };

            panel.Children.Add(image);
        }
    }
}
/// <summary>
/// Преобразование Bitmap в BitmapImage
/// </summary>
/// <param name="bitmap">Bitmap сущность</param>
/// <returns></returns>
private BitmapImage ConvertBitmapToBitmapImage(Bitmap bitmap)
{
    var bitmapImage = new BitmapImage();
```

```
            using (var memoryStream = new MemoryStream())
            {
                bitmap.Save(memoryStream, ImageFormat.Bmp);
                memoryStream.Position = 0;
                bitmapImage.BeginInit();
                bitmapImage.CacheOption = BitmapCacheOption.OnLoad;
                bitmapImage.StreamSource = memoryStream;
                bitmapImage.EndInit();
            }

            return bitmapImage;
        }
        private void Create3DModelButton_Click(object sender, RoutedEventArgs e)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "STL files (*.stl)|*.stl";
            saveFileDialog.DefaultExt = ".stl";
            saveFileDialog.AddExtension = true;

            if (saveFileDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                string filePath = saveFileDialog.FileName;

                var dualContouringFilter = new DualContouringFilter();

                dualContouringFilter.ReconstructionDicomSlices(_destinationDirectory, filePath);

                MessageBox.Show("Model has been saved successfully.", "Reconstruction successful",
MessageBoxButton.OK, MessageBoxImage.Information);
            }
        }
        private void ChangeDicomSetButton_Click(object sender, RoutedEventArgs e)
        {
            ShowMainWindow();
            Close();
        }
        private void CloseButton_Clock(object sender, RoutedEventArgs e) => Close();
        /// <summary>
        /// Показать главное окно
        /// </summary>
        private void ShowMainWindow()
        {
            var mainWindow = new MainWindow();
            mainWindow.Show();
        }
    }
}
```

ПРИЛОЖЕНИЕ Б
(обязательное)

Функциональная схема алгоритма Dual contouring

ПРИЛОЖЕНИЕ В
(обязательное)

Руководство системного программиста

Разработанное программное обеспечение предназначено для 3*D*-реконструкции поясничного отдела позвоночника человека. Перед, непосредственно, самой реконструкцией пользователю необходимо будет задать папку, где находятся файлы, имеющие расширение *DICOM* и представляющие собой набор срезов позвоночника человека, папку в которую будут загружены обработанные при помощи порогового фильтра исходные данные, а также пороговое значение, для фильтра. Без задания корректных этих параметров пользователю будет запрещён доступ к полному функционалу приложения, а именно к просмотру исходных и обработанных срезов и также 3*D*-реконструкции модели из обработанных данных.

Для корректной работы программного обеспечения необходимо соблюдение следующих требований:

– поддерживаемые операционные системы *Windows* 7 и выше;

– наличие стандартной клавиатуры;

– наличие компьютерной мышки;

– наличие следующих устройств вывода: экран, подключаемый по *HDMI*.

Перед запуском программы нет необходимости выполнения каких-либо действий.

ПРИЛОЖЕНИЕ Г
(обязательное)

Руководство программиста

Разработанное программное обеспечение предназначено для 3*D*-реконструкции поясничного отдела позвоночника человека. Перед, непосредственно, самой реконструкцией пользователю необходимо будет задать папку, где находятся файлы, имеющие расширение *DICOM* и представляющие собой набор срезов позвоночника человека, папку в которую будут загружены обработанные при помощи порогового фильтра исходные данные, а также пороговое значение, для фильтра. Без задания корректных этих параметров пользователю будет запрещён доступ к полному функционалу приложения, а именно к просмотру исходных и обработанных срезов и также 3*D*-реконструкции модели из обработанных данных.

Для корректной работы программного обеспечения необходимо соблюдение следующих требований:

– поддерживаемые операционные системы *Windows* 7 и выше;

– наличие стандартной клавиатуры;

– наличие компьютерной мышки;

– наличие следующих устройств вывода: экран, подключаемый по *HDMI*.

Перед запуском программы нет необходимости выполнения каких-либо действий.

Приложение запускается путём открытия файла *UserUi.exe*, находящимся в каталоге *bin*/*Release*.

В данной программе в качестве исходных данных используется язык программирования *C#* и платформа *WPF*.

ПРИЛОЖЕНИЕ Д
(обязательное)

Руководство пользователя

Разработанное программное обеспечение предназначено для 3*D*-реконструкции поясничного отдела позвоночника человека. Перед, непосредственно, самой реконструкцией вам необходимо будет задать папку, где находятся файлы, имеющие расширение *DICOM* и представляющие собой набор срезов позвоночника человека, папку в которую будут загружены обработанные при помощи порогового фильтра исходные данные, а также пороговое значение, для фильтра. Без задания корректных этих параметров пользователю будет запрещён доступ к полному функционалу приложения, а именно к просмотру исходных и обработанных срезов и также 3*D*-реконструкции модели из обработанных данных. Необходимо учесть, что путь к папке в которой будут храниться обработанные срезы не должен содержать русских символов.

После указания необходимых данных будут доступен полностью весь функционал, а именно будет доступна функция просмотра начального и обработанного набора срезов, 3*D*-реконструкции модели из обработанного набора срезов, а также можно будет заменить исходный набор срезов на другой и поменять пороговое значение.

Для корректной работы программного обеспечения необходимо соблюдение следующих требований:

– поддерживаемые операционные системы *Windows* 7 и выше;

– наличие стандартной клавиатуры;

– наличие компьютерной мышки;

– наличие следующих устройств вывода: экран, подключаемый по *HDMI*.

Для установки приложения необходимо загрузить на компьютер каталог *Application*.

В случае непредвиденного «зависания» программы рекомендуется завершить процесс в диспетчере задач и запустить снова. Или можно подождать некоторое время, так как существует возможность зависания программы за счет длительного выполнения 3*D*-реконструкции.

Приложение имеет простую реализацию, а все операции понятны на интуитивном уровне.