САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

Улитина Ирина Александровна

Выпускная квалификационная работа бакалавра

Построение изображений и 3D-моделей по проекциям с использованием анатомического атласа

Направление 010400.62

Прикладная математика, фундаментальная информатика и программирование

Научный руководитель,

старший преподаватель

Стученков А.Б.

Санкт-Петербург

2017

Содержание

Введение

В современной медицине огромную роль играют модели тела человека, человеческих органов, их расположение, взаимодействие. Подобные модели широко распространены при обучении студентов-медиков, поскольку являются компактным пособием, которое доступно даже с экрана смартфона. Помимо учебных целей, подобные модели помогают в проведении операций, моделировании процесса лечения заболеваний. Построение трёхмерных моделей не только человека, но и его отдельных органов (например, позвонков) открывает широкие возможности для развития травматологии и ортопедии.

В настоящее время всё более и более актуальной становится проблема нехватки высококачественных и дорогих приборов в клиниках. Одним из важнейших приборов для исследования и профилактики заболеваний является томограф (в первую очередь, рентгеновский томограф и магнитно-резонансный томограф). С его помощью можно получить информацию о состоянии внутренних органов человека, на основании которой делают выводы о тех процессах, которые происходят в организме.

Ещё одним из способов исследования тела пациента является рентгенография. Рентгенологическое исследование органов позволяет уточнить форму данных органов, их положение, тонус, перистальтику, состояние рельефа слизистой оболочки. В отличие от томографа, рентгеновский аппарат – относительно недорогой прибор и имеется практически в каждом медицинском учреждении.

Однако стоит принять во внимание тот факт, что в число недостатков рентгенографии входит следующее:

- Обычные рентгеновские изображения отражают проекционное наслоение сложных анатомических структур, то есть их суммационную рентгеновскую тень, в отличие от послойных серий изображений, получаемых современными томографическими методам.
- Без применения контрастирующих веществ рентгенография недостаточно информативна для анализа изменений в мягких тканях, мало отличающихся по плотности (например, при изучении органов брюшной полости).

Именно поэтому разработка метода, который бы объединял сильные и слабые стороны томографии и рентгенографии, является одной из приоритетных задач современной медицины. Открытия в этой области смогли бы повысить эффективность лечения, стать инструментом ранней диагностики и оценки состояния многих заболеваний, в том числе и опухолей благодаря наглядности, которую может предоставить объёмная модель тела пациента.

В данной работе рассматривается один из возможных способов реализации подобной идеи: моделирование на основе данных, которые могут быть получены непосредственно на месте (рентгеновские снимки), и при помощи стандартного анатомического атласа, переведённого в цифровой формат.

Постановка задачи

Цель работы состоит в решении двух задач: задачи реализации алгоритма реконструкции среза тела пациента по двум проекциям с использованием анатомического атласа и задачи создания объемной модели исследуемой области тела пациента по набору срезов, полученных в результате решения первой задачи.

Для решения первой задачи необходимо было реализовать рабочую версию алгоритма, описанного в статьях [1] и [2]. Упомянутый алгоритм предназначен для построения среза тела пациента в нужной области с использованием двух рентгеновских снимков и анатомического атласа.

Помимо этого, одной из важных подзадач исследования было изучение технологий обработки изображений, а именно рентгенограмм. Данная проблема требовала отдельного рассмотрения в рамках исследования и реализации алгоритма, поскольку на вход программе подаются необработанные предварительно снимки, полученные в условиях обычной лаборатории при больнице, и, соответственно, ни о какой специальной цифровой обработке до прогонки через алгоритм и речи идти не может. Поэтому предполагалось ознакомление с рядом методик, описывающих специальную предварительную обработку изображений до того, как начать поиск контуров на загруженных в программу снимках и срезах.

Кроме этого, отдельно требовалось обработать изображения из атласа для корректного построения объёмной модели.

Второй задачей, решаемой в данной работе, была реализация одного из возможных вариантов построения объёмной модели, используя данные, которые были получены после реализации алгоритма построения картины среза тела пациента и прогона через него подготовленных данных для получения нескольких последовательных срезов.

Обзор литературы

При написании работы была использована следующая литература:

1. Sergeev S. L., Stuchenkov A. B. An algorithm of deformation of a flat image.

   В данной статье описан общий способ нахождения деформации изображения среза тела человека при вписывании в многоугольник. При реализации алгоритма было использовано описание шагов, приведённых в статье, для случая, когда источников излучения только два.

2. Sergeev S. L., Stuchenkov A. B. Modeling of deformation of an elastic body slice.

   В статье представлен частный случай, когда имеется только два источника излучения. За основу была принята именно эта идея, которая, впоследствии, и была реализована.

3. Доля П.Г. Методы обработки изображений.

   В данном пособии описаны наиболее частые способы обработки монохромных изображений в цифровом формате. При разработке способа обработки рентгенограмм использовались разделы, посвящённые преобразованию яркости и пространственной фильтрации.

4. Гонсалес Р., Вудс Р., Эддинс С. Цифровая обработка изображений в среде MATLAB.

   Данная книга посвящена описанию не только практической стороны работы с изображениями, но и математическому обоснованию производимых над ними операций. При работе над обработкой рентгенограмм использовались разделы, посвященные морфологической реконструкции и преобразованию яркости.

5. Методы компьютерной обработки изображений (под ред. В.А.Сойфера).

   Данная книга затрагивает проблемы, с которыми можно столкнуться при работе с изображениями. Особое внимание при работе с данным источником уделялось разделу поиска контуров.

Глава 1. Обзор существующих аналогов

Задача построения объёмной модели тела человека является крайне актуальной в настоящее время. Построение модели для каждого конкретного пациента может сыграть очень важную роль при проведении хирургических операций. Например, при проведении глубокой стимуляции головного мозга до самой операции пациенту необходимо пройти тщательное обследование, после которого уже возможно проведение самой операции. В ходе самой операции пациенту имплантируется мозговой электростимулятор, известный как нейростимулятор. Мозговой электростимулятор подаёт постоянные электрические импульсы в конкретные области мозга. Однако неудачное предварительное обследование способно привести к таким последствиям, как: проблемы с концентрацией, просачивание цереброспинальной жидкости, головокружение и потеря равновесия, затруднение в движениях, зрительное и речевое нарушение или неспособность, кома, отёк мозга, инсульт, судороги[3].

Решить задачу построения объёмной модели можно несколькими способами (см. рис. 1). Первый способ предполагает наличие объёма данных в виде заранее составленных математических моделей и данных томографии. Второй способ предназначен для случая, если входными данными являются отдельные изображения и их количество ограничено (как в случае рентгеновских снимков). Для второго способа существует классификация по наличию образцовой модели: по её наличию и её отсутствию (классификация взята из статьи [4]). Более того, в статье [4] приведён большой список алгоритмов, которые представляют собой решение как для первого способа построения, так и для второго.

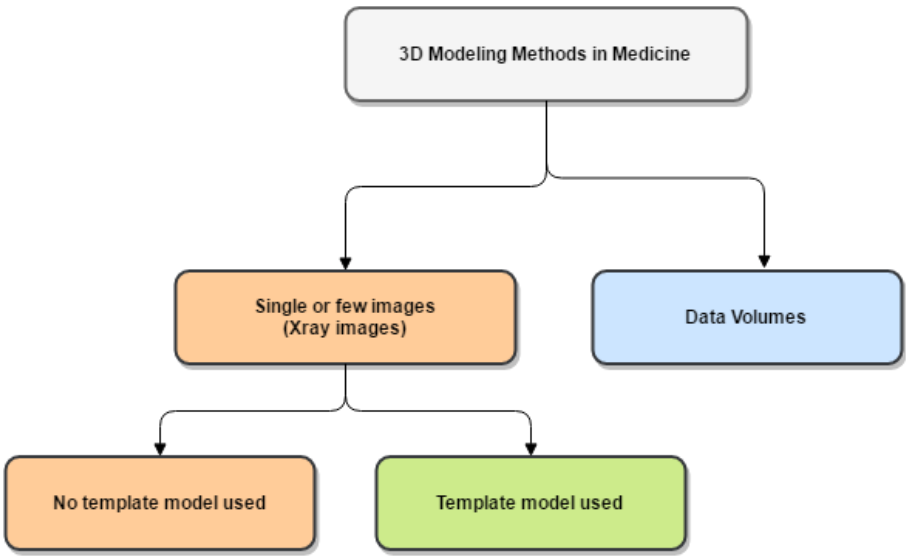


Рис. 1. Классификация методов построения объёмной модели.

Существует огромное количество проектов, которые посвящены данной задаче. В данной работе будут упомянуты только два из них.

Первым является проект BioDigital [5].

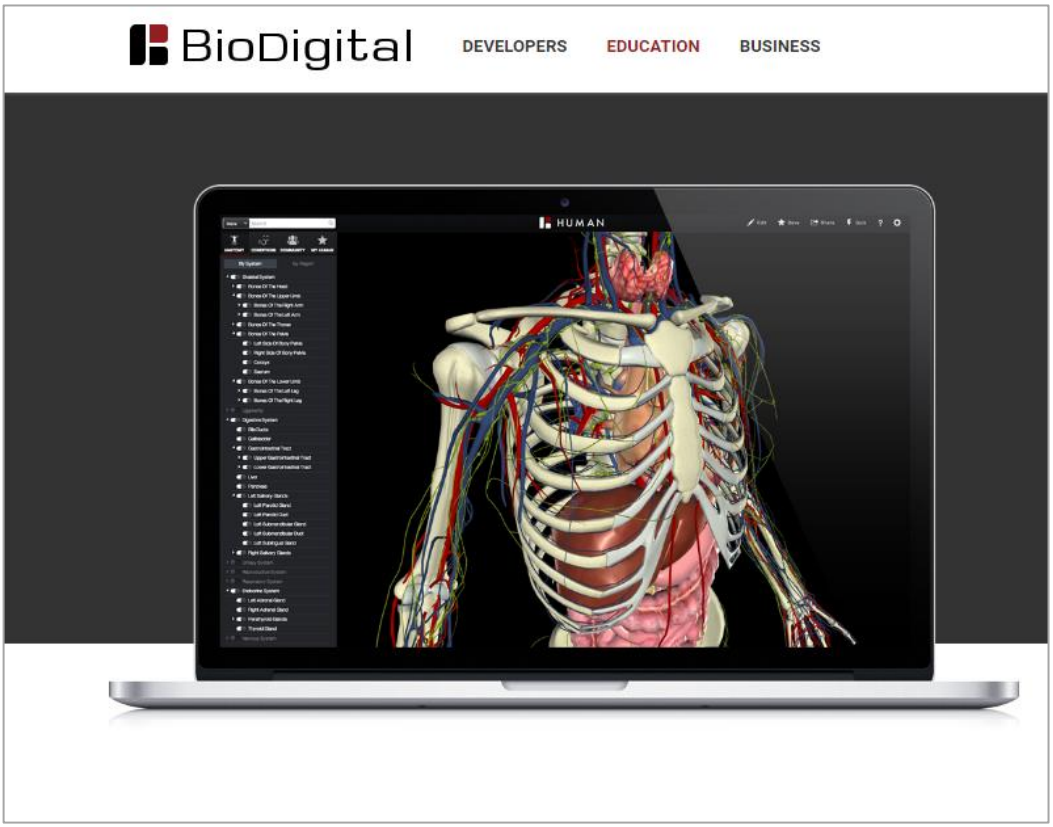


Рис. 2. Скриншот сайта проекта BioDigital.

Подобный сервис представляет большой интерес, поскольку на нём не только реализованы объёмные модели тела человека и внутренних органов (рис. 2), но также реализована анимация, представляющая динамические процессы, происходящие внутри.

Ещё одним проектом, заслуживающим внимания, является 3D Slicer [6], который представляет собой платформу для анализа (включая регистрацию и интерактивную сегментацию) и визуализации (в том числе объемную визуализацию) медицинских изображений и для исследований в области терапии, управляемой изображением (Image-Guided Therapy) (см. рис. 3).

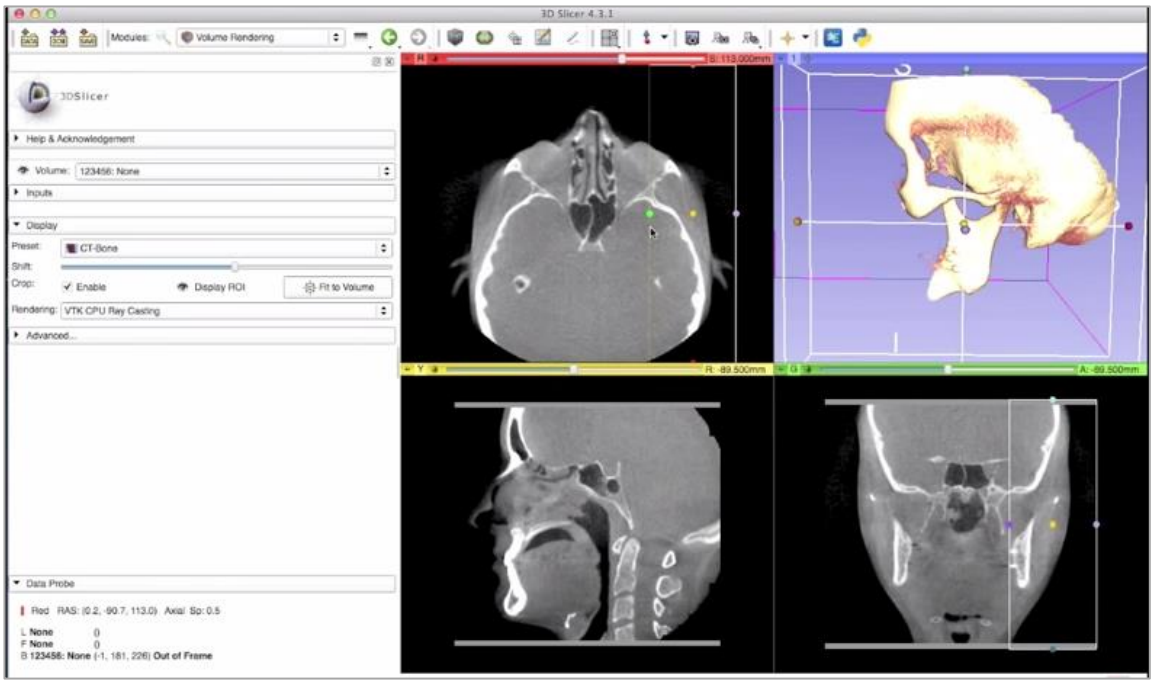


Рис.3. Скриншот работы программы 3D Slicer.

Помимо описанных выше сервисов, особый интерес представляет построение объёмных моделей, основываясь на рентгенограммах, в частности моделей костей (см. рис. 4).

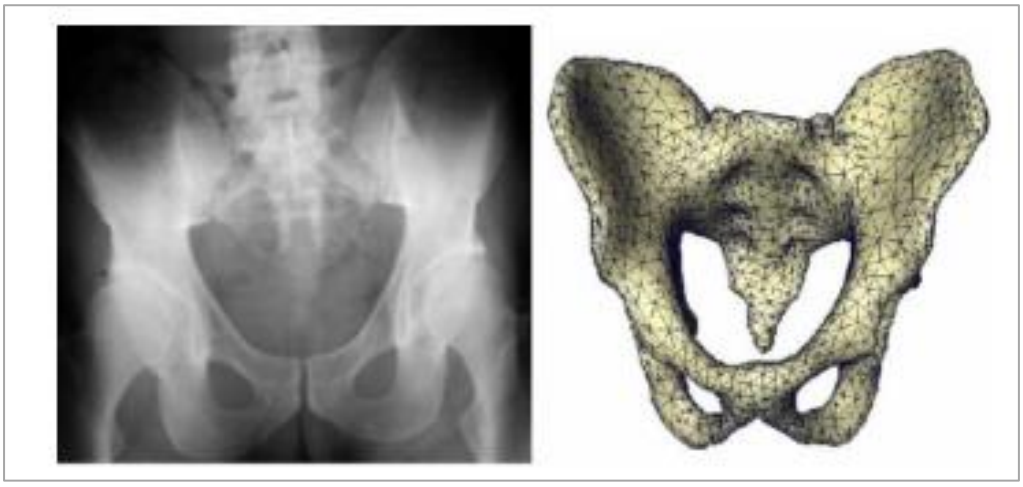


Рис. 4. Пример полученной объёмной модели костей на основе рентгенограмм. Рисунок взят из статьи [7].

Целесообразность построения объёмных моделей в этом случае заключается в том, что подобные реконструированные модели могут помочь при предоперационном планировании и при проведении операций с компьютерным управлением. Построение таких моделей также стараются делать без применения МРТ, а с использованием рентгеновских снимков, где полученный контур костей затем корректируют в соответствии с заранее заданной математической моделью. Подобная модель может быть получена, например, в результате статического анализа коллекции форм костей. Найденная при помощи рентгенограмм модель может быть подкорректирована при помощи нейронных сетей. Например, в статье [8] описано использование самоорганизующихся карт Кохонена (см. рис. 5).

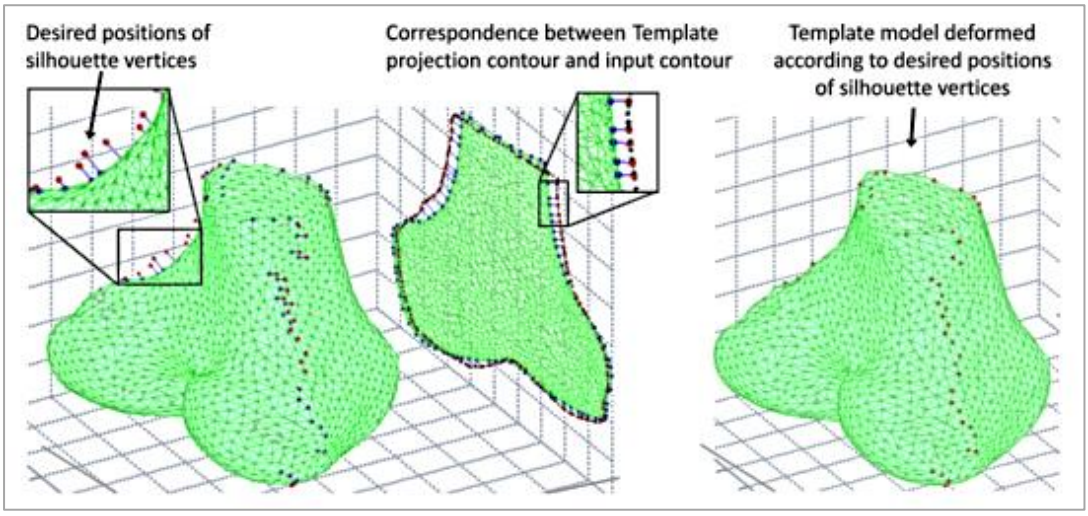


Рис.5. Применение самоорганизующихся карт Кохонена для корректирования формы костей. Рисунок взят из статьи [8].

Помимо этого, существует метод построения среза тела пациента на основе метода упругой плёнки, учитывающий деформацию внутренних контуров органов, который описан в [9].

Глава 2. Реализация алгоритма получения данных для построения модели

В данной главе описывается реализация алгоритма построения рисунка среза тела пациента, описанного в статьях [1] и [2].

2.1. Используемые средства

Программная реализация поставленных задач велась с использованием технологий, представленных на диаграмме ниже (см. рис. 6):

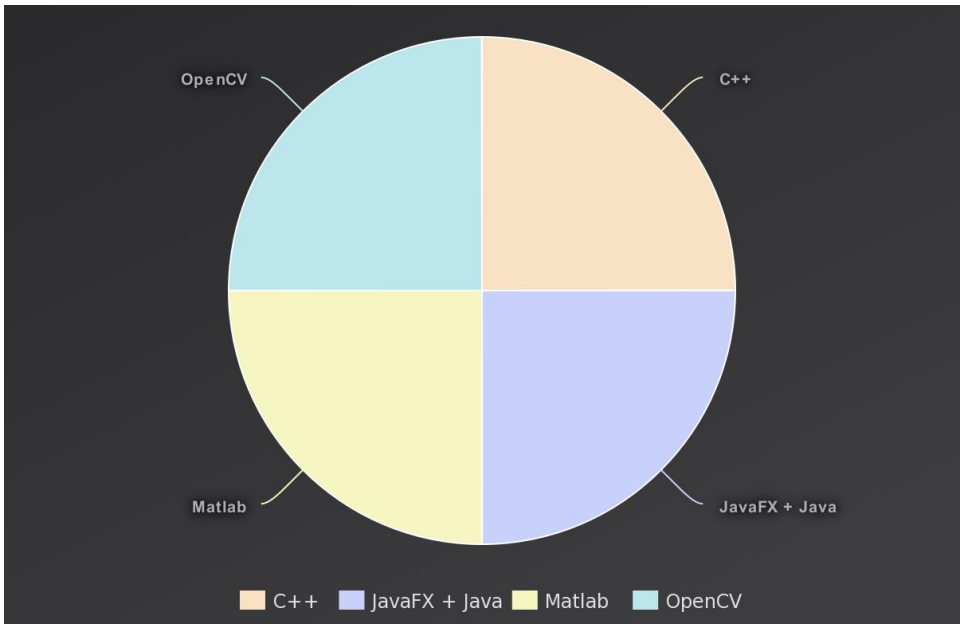


Рис. 6. Диаграмма технологий, используемых при написании работы.

Разработка алгоритма нахождения пространственных координат и обработки срезов велась на языке C++ с использованием библиотеки OpenCV.

Для разработки интерфейса для данного шага применялась JavaFX — платформа для создания RIA, которая позволяет строить приложения с насыщенным графическим интерфейсом пользователя для непосредственного запуска из-под операционных систем. Написанное приложение было конвертировано в Java-архив – jar, что позволило создать исполняемый файл exe, используя Launch4j [10]. Данное

приложение a является кросс-платформенным инструментом для обертывания Java-приложений. Исполняемый файл может быть сконфигурирован с использованием конкретной версии JRE. Помимо этого, Launch4j также позволяет добавить иконку приложения, а также установить параметры исполнения приложения.

Для обработки рентгенограмм и построения объёмной модели использовался MATLAB и Image Processing Toolbox.

Общая схема работы программы представлена на следующей диаграмме (см. рис. 7):

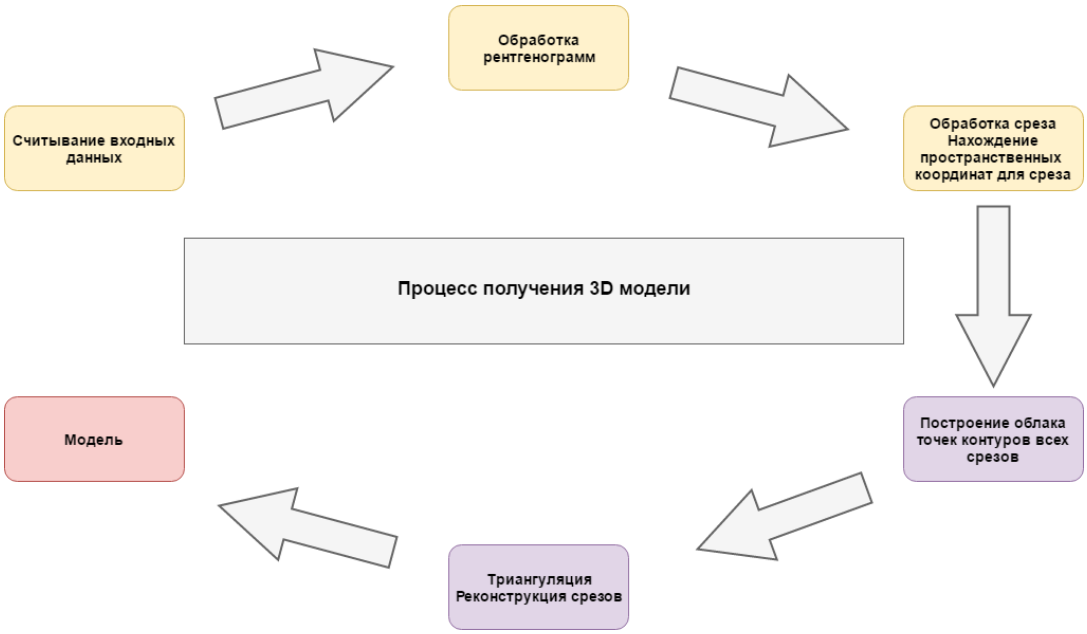


Рис. 7. Этапы работы программы. Жёлтым цветом выделены этапы вписывания среза, фиолетовым – этапы построения объёмной модели, красным – финальный результат.

2.2. Описание работы с интерфейсом

Главное меню представлено на рис. 8. Для того, чтобы приступить к выполнению операции вписывания, необходимо перейти в раздел «Start New». Краткие инструкции по установке окружения доступны в разделе «How it works».

Рис. 8. Работа программы: Начальное меню.

Стоит учесть, что перед первым запуском программы необходимо зайти в раздел «Settings» и указать следующие пути (см. рис. 9):

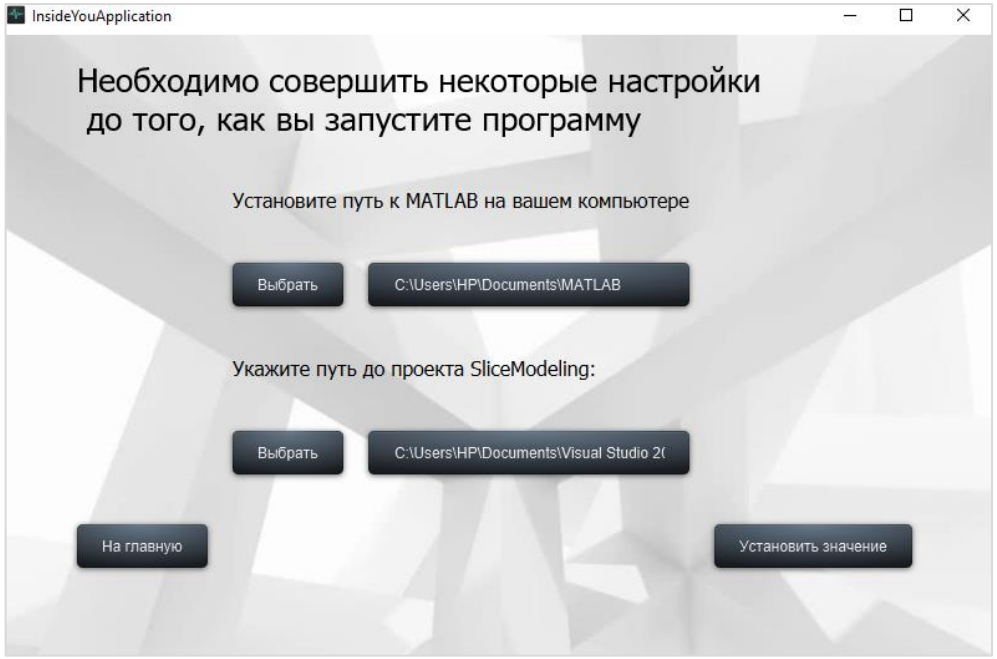


Рис. 9. Работа программы: Меню настроек.

При переходе в раздел «Start New» пользователю предлагается выбрать сам аппарат, при помощи которого были сделаны рентгенограммы, загрузить эти снимки, а также указать путь до анатомического среза из атласа (см. рис. 10).

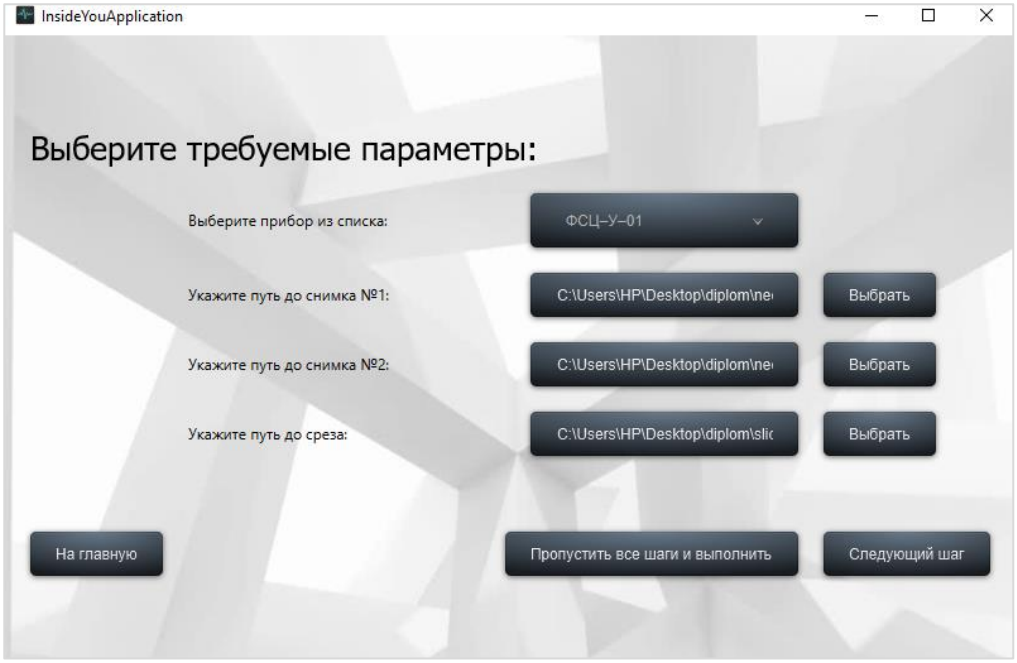


Рис. 10. Работа программы: выбор основных параметров

После этого пользователю предлагается указать уровень, на котором расположен выбранный ранее срез. Эта возможность реализована двумя способами: при помощи бегунка и с помощью текстового поля под изображением (см. рис. 11).

Рис. 11. Работа программы: выбор уровня для построения среза

После нажатия «Приступить к выполнению» происходит запись параметров в файл, который располагается в папке MATLAB.

2.3. Реализация обработки рентгенограмм

Обработка рентгенограмм представляет собой нетривиальную задачу, которую можно решить, используя библиотеки для работы с изображениями.

В качестве библиотеки для работы с изображениями была выбрана библиотека OpenCV . Выбор был сделан в её пользу, поскольку OpenCV обладает большой функциональностью, позволяющей работать с контурами, найденными на изображениях. Однако в ходе работы над исследованием для предварительной обработки рентгеновских снимков использовалась библиотека Image Processing Toolbox. Данное решение было принято потому, что реализовать обработку рентгенограмм исключительно средствами OpenCV оказалось невозможным ввиду отсутствия необходимых инструментов обработки, которые были найдены в библиотеке Image Processing Toolbox (функции imadjust, bwboundaries).

Рентгенограммы подвергаются специальной обработке, которая описана в статье [11] и включает в себя следующие операции:

1. Изменение яркости изображений.

Это один наиболее успешных способов обработки рентгенограмм, математическая постановка для которого заключается в следующем: необходимо подобрать некий оператор *T*, который выполнит преобразование изображения *μ(x,y)*

$$g(x,y) = T\big(\mu(x,y)\big),$$

где $g(x,y)$ – обработанное выходное изображение. Преобразование изображений может осуществляться, например, при помощи логарифмических, кусочно-линейных функций. Подбирать

преобразующую яркость кривую бывает крайне сложно, поэтому было принято решение воспользоваться готовым инструментом библиотеки Image Processing Toolbox – функцией imadjust [12; 13].

Таким образом, осуществляется установка значений интенсивностей на изображении с помощью функции imadjust, где указан диапазон интенсивностей результирующего изображения. В ходе эксперимента были получены следующие параметры для данной функции: значения яркости в диапазоне [0, 75/255] преобразуются в значения яркости в диапазоне [0, 1] , параметр gamma равен 1, что соответствует линейной характеристике передачи уровней и отсутствию гамма-коррекции. Параметр gamma служит для задания формы кривой, отображающей яркость. Линейное отображение как раз характеризуется значением, равным единице [13].

Данные параметры были найдены эвристическим путём: в результате тестирования на небольшой контрольной группе рентгенограмм было получено оптимальное значение, подходящее для всех изображений в контрольном множестве. Затем была проведена обработка тестового множества изображений, которая показала работоспособность найденных значений параметров.

2. Заполнение отверстий (holes) на бинарном изображении, полученном после изменения интенсивности.

Заполнить отверстия на бинарном изображении возможно с помощью алгоритма на основе морфологической реконструкции. Отверстием называется множество фоновых пикселов, которых нельзя достигнуть путём заполнения фона, начиная от края изображения. Реконструкция является морфологическим преобразованием, в котором участвуют маркер, маска и структурообразующий элемент. Применяя функцию imfill, получаем на выходе бинарное изображение, где пиксели

отверстий инвертированы, т.е. отверстия на двоичном изображении заполнены.

3. Поиск контура на изображении (bwboundaries) и сохранение найденного контура на данном изображении.

Контуром изображения будем называть пространственно протяжённый разрыв (перепад) значений яркости. Основные проблемы, которые связаны с обнаружением контуров, связаны, прежде всего, с недостаточно быстрой сменой яркости, размытости, шумов. Обычно для решения проблем подобного характера применяют специальные процедуры обработки бинарного изображения [14]. В данном алгоритме обработки рентгенограмм уже проведены операции изменения интенсивности, перевод в бинарный формат и заполнение отверстий, что позволило получить вполне чётко выделенный объект на рентгеновском снимке. Это позволит найти требуемый контур правильно.

Поиск контура на изображении осуществляется путём применения модифицированного алгоритма отслеживания границ на основе окрестности Мура (Moore neighborhood [15]) для каждого пикселя (the Moore-Neighbor tracing algorithm) с использованием критерия остановки Якоба (Jacob's stopping criteria) [16].

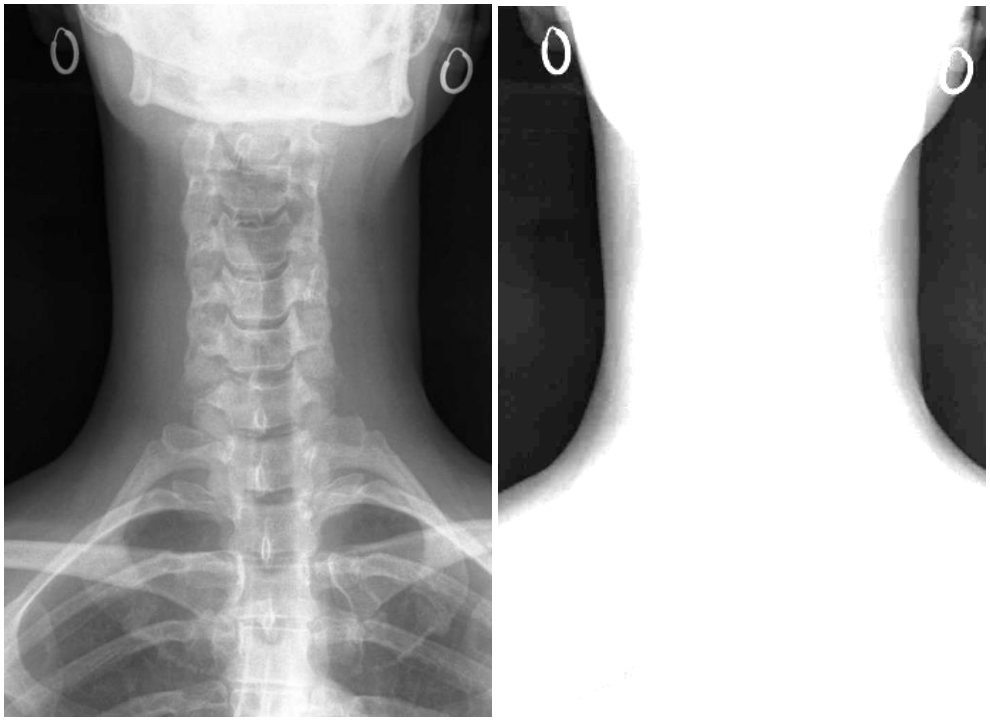
Код для описанных выше операций выглядит следующим образом:

```
I = imread(fileName);
J = imadjust(I, [0 75]/255, [ ], 1);
bw = im2bw(J);
BW_filled = imfill(bw,'holes');

boundaries = bwboundaries(BW_filled);
b = [1;1];
```

```
for p=1:size(boundaries)
   bb = boundaries{p};
   if(size(b,1)<size(bb,1))
       b = bb;
   end
end
```

После исполнения данного фрагмента программы получаем следующий результат (см. рис. 12):



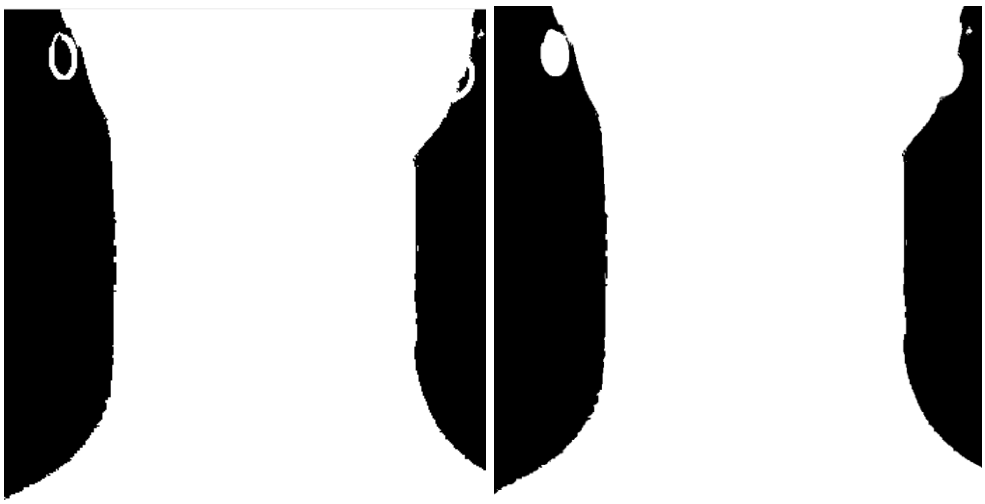


Рис.12. Последовательная обработка рентгенограммы, взятой с [17]. На верхнем левом изображении представлен исходный снимок. На верхнем правом изображении – после применения функции imadjust. На нижнем левом изображении – после применения функции im2bw. На нижнем правом изображении – после применения функции imfill.

Зачастую найденная граница расположена на границах изображения и плохо распознается средствами библиотеки OpenCV. С целью решения данной проблемы было решено добавить к матрице изображения дополнительные пиксели с каждой из сторон. Это позволило обеспечить безошибочное опознание найденного контура в OpenCV.

```
RGB = padarray(RGB,[3 3],'both');
```

После проделанных выше операций необходимо отобразить конечное изображение рентгенограммы и наложить на него найденную границу (см. рис. 13).

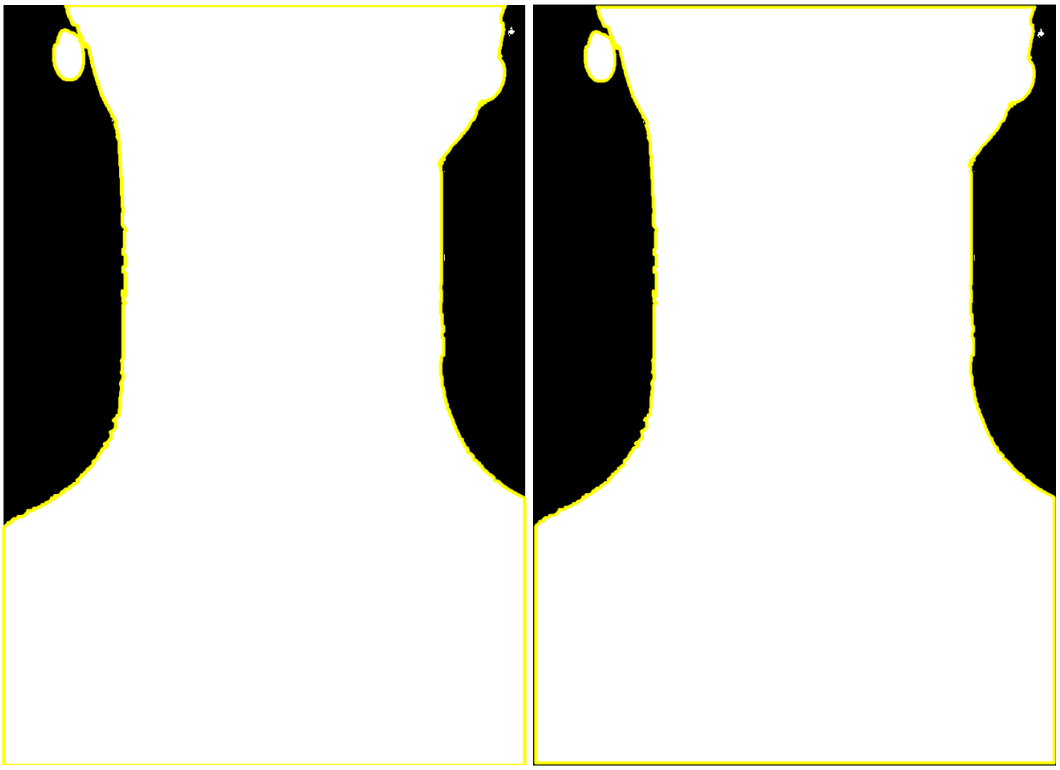


Рис. 13. Наложение границы на итоговый вариант обработки рентгенограммы. Найденная граница выделена жёлтым цветом. Слева представлен вариант без добавления дополнительных пикселей к каждой из сторон, справа – результат при добавлении.

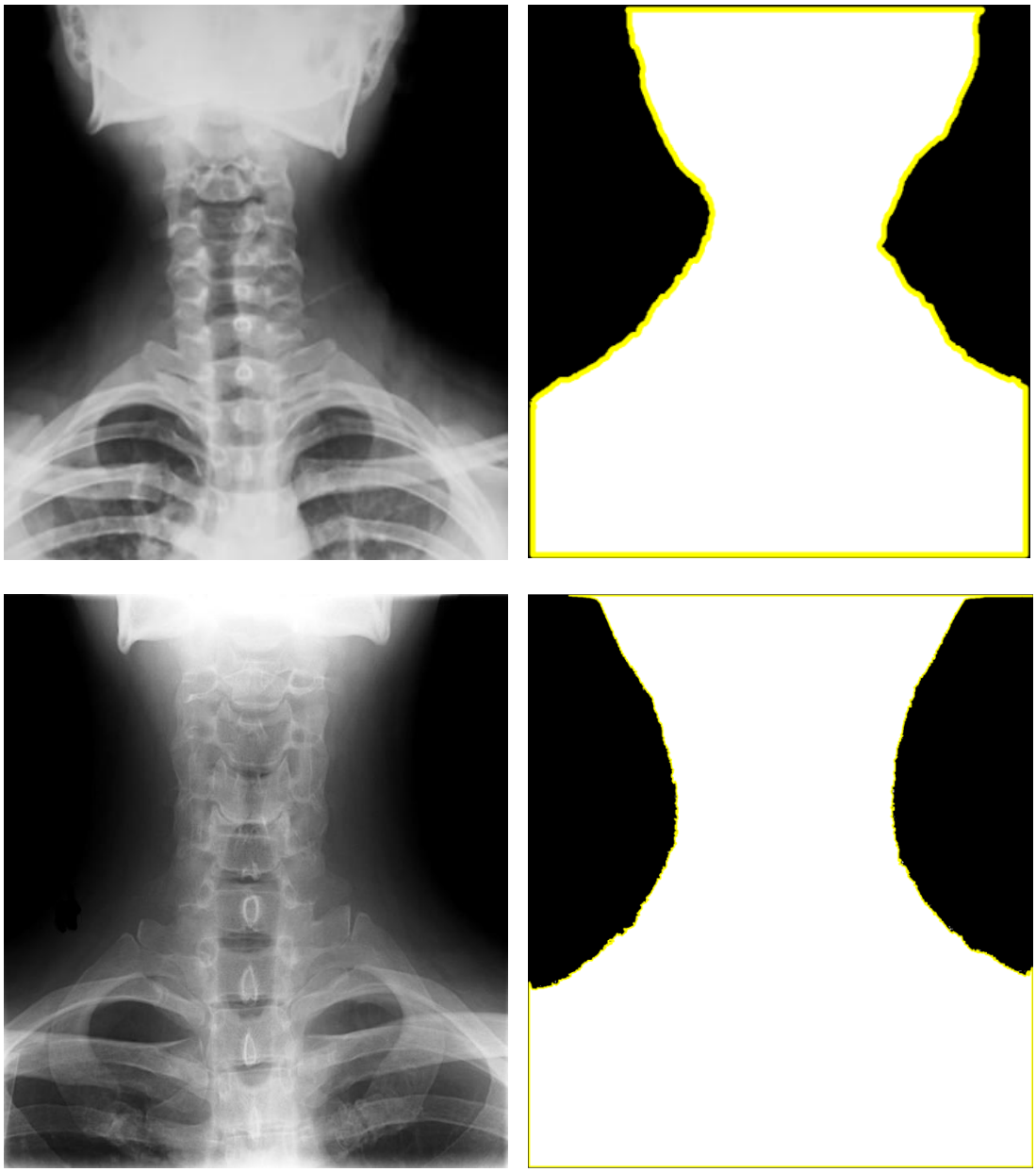
Данный этап можно проделать следующим образом:

```
figure (1)
imshow(RGB,'Border','tight');
hold on
plot(b(:,2)+3,b(:,1)+3,'y','LineWidth',2);
r = 150; % pixels per inch
set(gcf, 'PaperUnits', 'inches', 'PaperPosition',
[0 0 size(RGB,2) size(RGB,1)]/r);
print(gcf,'-dpng',sprintf('-r%d',r), newName);
```

В данном фрагменте кода также учтен тот факт, что сохранение figure в формате png должно происходить без сохранения дополнительных

полей, которые являются стандартными при отображении изображений или графиков. С этой целью изменяем свойства PaperUnits и PaperPosition так, как показано выше.

Далее показаны другие примеры обработки рентгенограмм, которые были взяты из интернета (см. рис. 14).





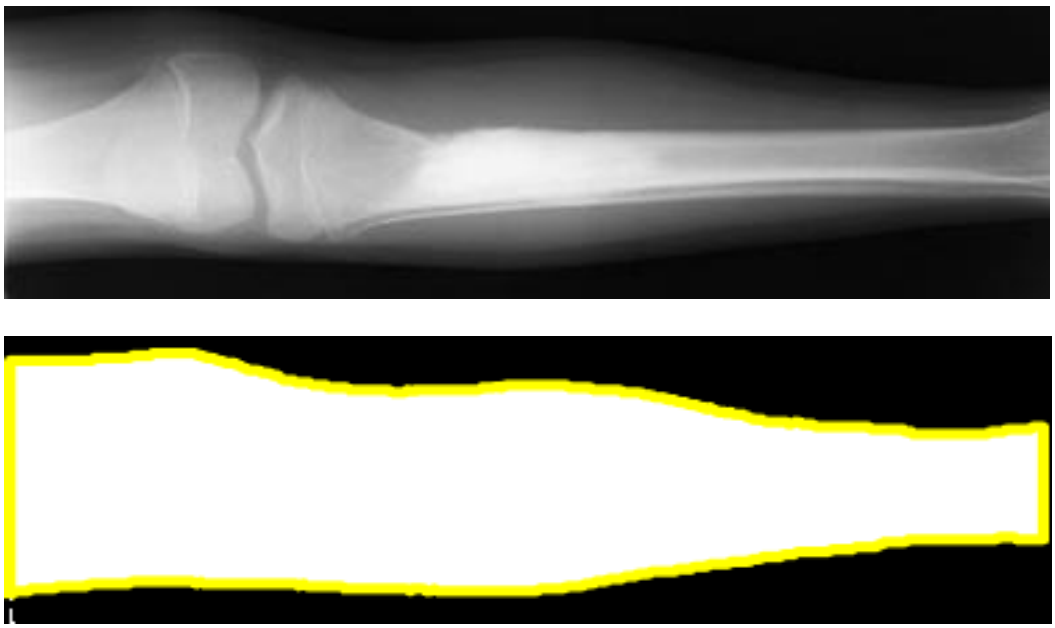



Рис.14. Примеры обработки рентгенограмм разного разрешения. Справа показано исходное изображение, слева – результат работы алгоритма.

Как видно из результатов работы описанного выше алгоритма в случае, если разрешение изображения большое, то найденный контур на таких изображениях почти не заметен. И наоборот: если разрешение небольшое, то контур дольно отчётливо отображён на итоговой картинке. Кроме того, как видно на полученных изображениях, не всегда находится нужный контур (иногда границы тела на рентгенограммах расплывчатые и очень нечёткие), из-за чего происходит выделение уже внутренних слоёв тела, которые обладают более высокой яркостью на рентгеновских снимках, а внешние остаются неопознанными программой.

2.4. Реализация алгоритма

Краткое описание шагов алгоритма приведено на рис. 15:

Рис.15. Схема работы алгоритма получения данных для каждого среза.

Срезы для тестирования работы алгоритма были взяты с сервиса [18] и были предварительно обработаны при помощи программы Photoshop (см. рис. 16).

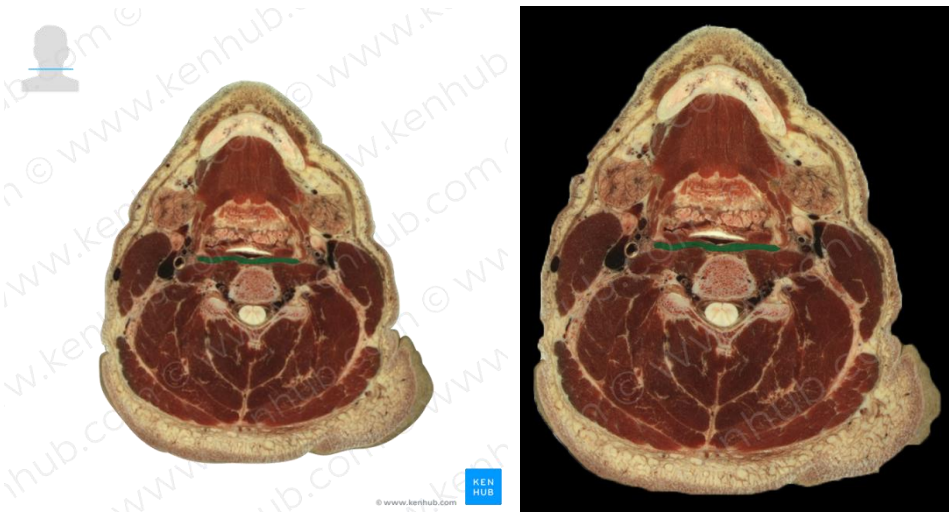


Рис. 16. Пример ручной обработки среза, взятого с сайта [18].

Данный сервис был выбран в виду следующих причин:

- Низкое качество снимков на других сервисах.

  В ходе поисков анатомического атласа в цифровой форме были найдены другие сервисы, такие как Anatomy Atlases [19], Visible Human Server [20] и т.д. Главной проблемой таких сервисов является невозможность получения качественного изображения с хорошим разрешением.

- Наличие посторонней информации на срезах.

  Часто в анатомических атласах добавлены специальные указатели и заметки. Они также являются нежелательными для использования срезов в качестве объектов для построения объёмной модели.

Именно поэтому все тестовые срезы брались с сервиса Kenhub.

Задача вписывания изображения среза в четырёхугольник, полученный путём пересечения прямых, соединяющих два источника и границы контуров, состояла из трёх этапов.

На первом этапе считываются все входные данные, строится четырехугольник ABCD (см. рис. 17 и 18), основываясь на параметрах отечественных и некоторых импортных моделей цифровых матричных флюорографов [21] (см. табл.1).

Наименование	Размер рабочего поля, мм	Фокусое расстояние, см
ФМЦ «Диарс-МР»	380 × 380	80
ФЦ-01 «Электрон»	390 × 390	100
ФСЦ–У–01	385 × 385	80
VERTIX UM-D	390 × 390	115 - 180
IMIX Thorax	390 × 390	115 - 180
ФЦМБ «Ренекс–флюоро»	390 × 390	120

Таблица 1. Характеристики рентгенофлюорографических комплексов для исследования грудной клетки.

В найденный четырёхугольник будет производиться вписывание среза.

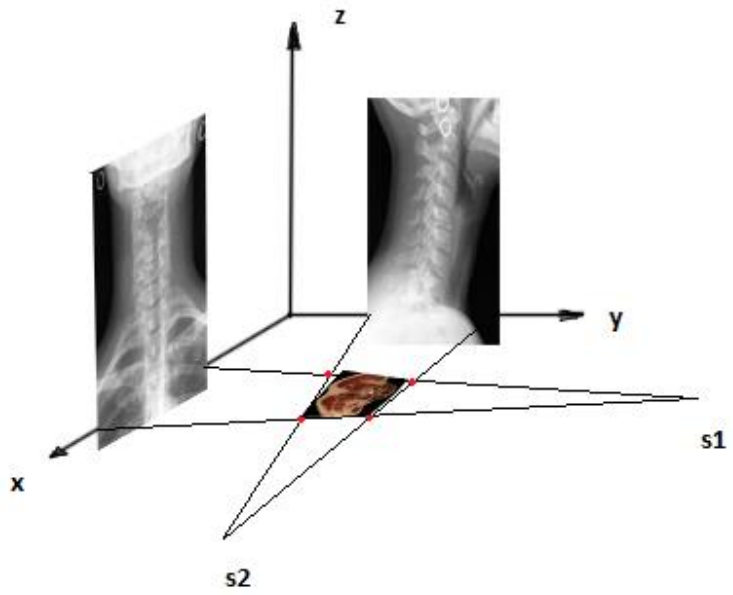


Рис. 17. Конечный результат работы алгоритма после прохождения всех стадий.

Поскольку размер рабочего поля цифрового флюорографа (x) и фокусное расстояние (fd) даны в миллиметрах и сантиметрах соответственно, то требуется перевод миллиметров и сантиметров в пиксели, чтобы сопоставить размеры рентгенограмм и источников в системе координат. Делается это по следующим формулам:

$$S_{1_x} = 37.94 * df$$

$$S_{1_y} = \frac{3.794 * x}{2}$$

$$S_{2_x} = \frac{3.794 * x}{2}$$

$$S_{2_y} = 37.94 * df$$

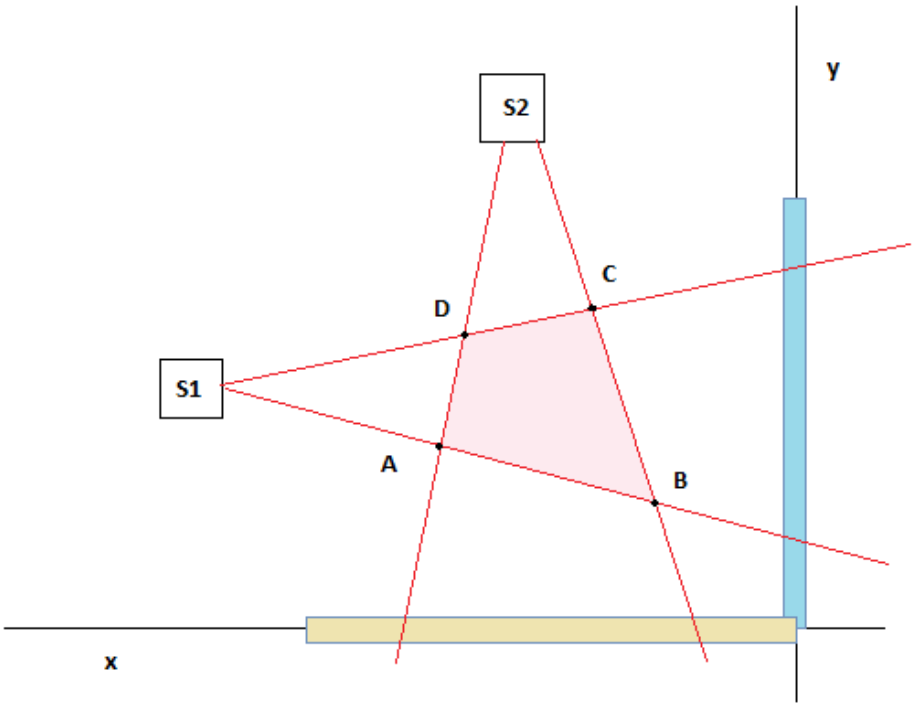


Рис. 18. Пример нахождения четырёхугольника ABCD, где s1 и s2 – источники излучения, синий и бежевый прямоугольники – рентгенограммы.

Четырёхугольник находится через обыкновенную систему линейных уравнений вида:

$$\begin{cases} k_j * x_i + b_j = y_i \\ k_l * x_i + b_l = y_i' \end{cases}$$

где (x_i, y_i) – координаты i-вершины четырёхугольника *ABCD*, а k_j, b_j, k_l и b_l– коэффициенты прямых, проходящих через вершину i. Подобная процедура проводится для всех вершин четырёхугольника.

Коэффициенты прямых, проходящих через вершины прямоугольника – это коэффициенты прямых, проходящих через источники излучения и границы объекта на рентгенограмме. Параметры для этих прямых могут быть найдены через следующую систему линейных уравнений:

$$\begin{cases} k_j * x_{Slice_i^m} + b_j = y_{Slice_i^m} \\ k_j * x_{S_i} + b_j = y_{S_i} \end{cases},$$

где (x_{S_i}, y_{S_i}) – координаты i-ого источника, а $x_{Slice_i^m}$и $y_{Slice_i^m}$ – это координаты m-ой крайней точки i-ой рентгенограммы. Для каждого снимка будет найдено 2 крайних точки. Составляем четыре системы для каждой прямой и находим коэффициенты прямых.

На втором этапе мы ищем линейное преобразование, используя полученный на предыдущем этапе четырёхугольник $ABCD$ и преобразованное изображение среза. Прежде всего, описываем прямоугольники вокруг найденной границы среза и вокруг построенного четырёхугольника $ABCD$. Находим отношение сторон (функция calculateNewSize) этих двух прямоугольников, ищем во сколько раз необходимо изменить высоту и ширину среза, строим изображение в соответствии с найденными пропорциями. Для этого вызываем функцию resizeImageToRectangle класса SliceMain. На вход этой функции подаётся изображение, которое было построено, оно является заменой предыдущего изображения среза. Помимо этого, для удобства проведения вычислений переносим четырёхугольник $ABCD$ в начало координат, поскольку изображение среза располагается в начале координат (см. рис. 19).

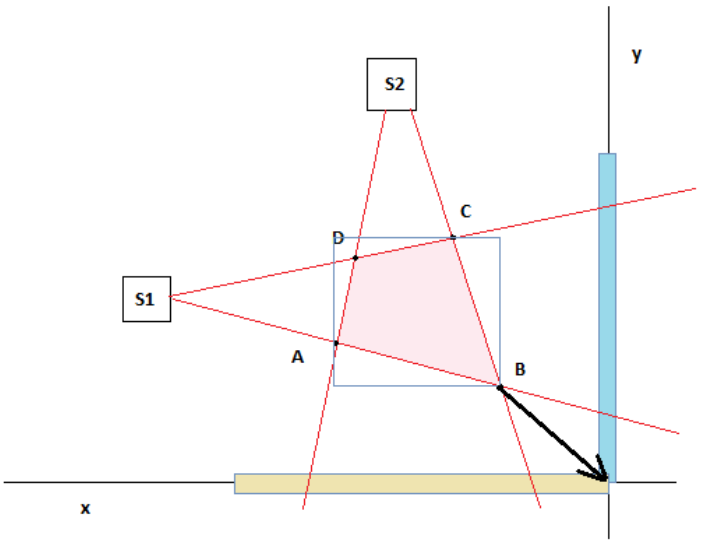


Рис.19. Перенос четырёхугольника ABCD в начало координат. Чёрной стрелкой показан вектор переноса.

По сути, происходит сопоставление прямоугольников, описанных вокруг четырёхугольника $ABCD$ и среза. Сохраняем величину проведённого сдвига. Находим контур среза на обновлённом изображении среза.

Затем осуществляем поиск данных для четырёхугольника, который описан вокруг среза и касается его крайних точек - четырёхугольника $A_1B_1C_1D_1$ (см рис. 20). Строя прямые, параллельные прямым четырёхугольника $ABCD$, которые касаются крайних точек среза, получаем четырёхугольник $A_1B_1C_1D_1$. Нахождение сторон, параллельных сторонам четырёхугольника $ABCD$, производилось по следующему правилу (класс SliceMain, функция findBkoeffForParallel): для каждой из четырёх сторон обходился контур среза, в каждой точке контура искалось значение коэффициента b, в массивы сохранялись координаты точки и значение b. Затем массив значений b сортировался по возрастанию, вместе с ним менялись массивы координат соответственно. Затем, для каждой из сторон брались либо первые значения из всех массивов, либо последние, в зависимости от того, с какой стороной работал алгоритм:

- Для AB – нам требовалось найти параллельную прямую с наименьшим возможным значением y.

- Для BC – нам требовалось найти параллельную прямую с наименьшим возможным значением x.
- Для CD – нам требовалось найти параллельную прямую с наибольшим возможным значением y.
- Для DA – нам требовалось найти параллельную прямую с наибольшим возможным значением x.

Рис. 20. Срез, для которого был найден четырёхугольник $A_1B_1C_1D_1$.

Линейное преобразование позволит найти новые координаты для каждой точки среза.

Поскольку система

$$-x_0 sin\varphi_i + y_0 cos\varphi_i + k\bar{b_i} cos\varphi_i - h_i = b_i cos\varphi_i \,, i = 1,2,3,4$$

содержит четыре уравнения и четыре неизвестных, а именно $x_0, y_0, k,\ h$, где h - это расстояние между двумя контурами, а x_0, y_0, k - это коэффициенты линейного преобразования:

$$\bar{x} = kx + x_0, \quad \bar{y} = ky + y_0$$

то был применён МНК для поиска всех неизвестных.

Все требуемые значения отсылаются в рабочее пространство MATLAB, где первоначально проводится преобразование радиан в градусы:

```
fi = [rad2deg(atan(kKoeff(1)))
rad2deg(atan(kKoeff(2)))
rad2deg(atan(kKoeff(3)))
rad2deg(atan(kKoeff(4)))];
```

а затем составляются матрицы A и b, которые выглядят следующим образом:

$$A = \begin{pmatrix} -\sin\varphi_1 & cos\varphi_1 & \bar{b_1}\cos\varphi_1 & 1 \\ -\sin\varphi_2 & cos\varphi_2 & \bar{b_1}\cos\varphi_1 & sign(A1 * A2) \\ -\sin\varphi_3 & cos\varphi_3 & \bar{b_1}\cos\varphi_1 & sign(A2 * A3) \\ -\sin\varphi_4 & cos\varphi_4 & \bar{b_1}\cos\varphi_1 & sign(A3 * A4) \end{pmatrix}$$

$$b = \begin{pmatrix} b_1 cos\varphi_1 \\ b_2 cos\varphi_2 \\ b_3 cos\varphi_3 \\ b_4 cos\varphi_4 \end{pmatrix}$$

Третий этап предполагает окончательную подгонку контура изображения среза под четырёхугольник $ABCD$ (рис. 21).

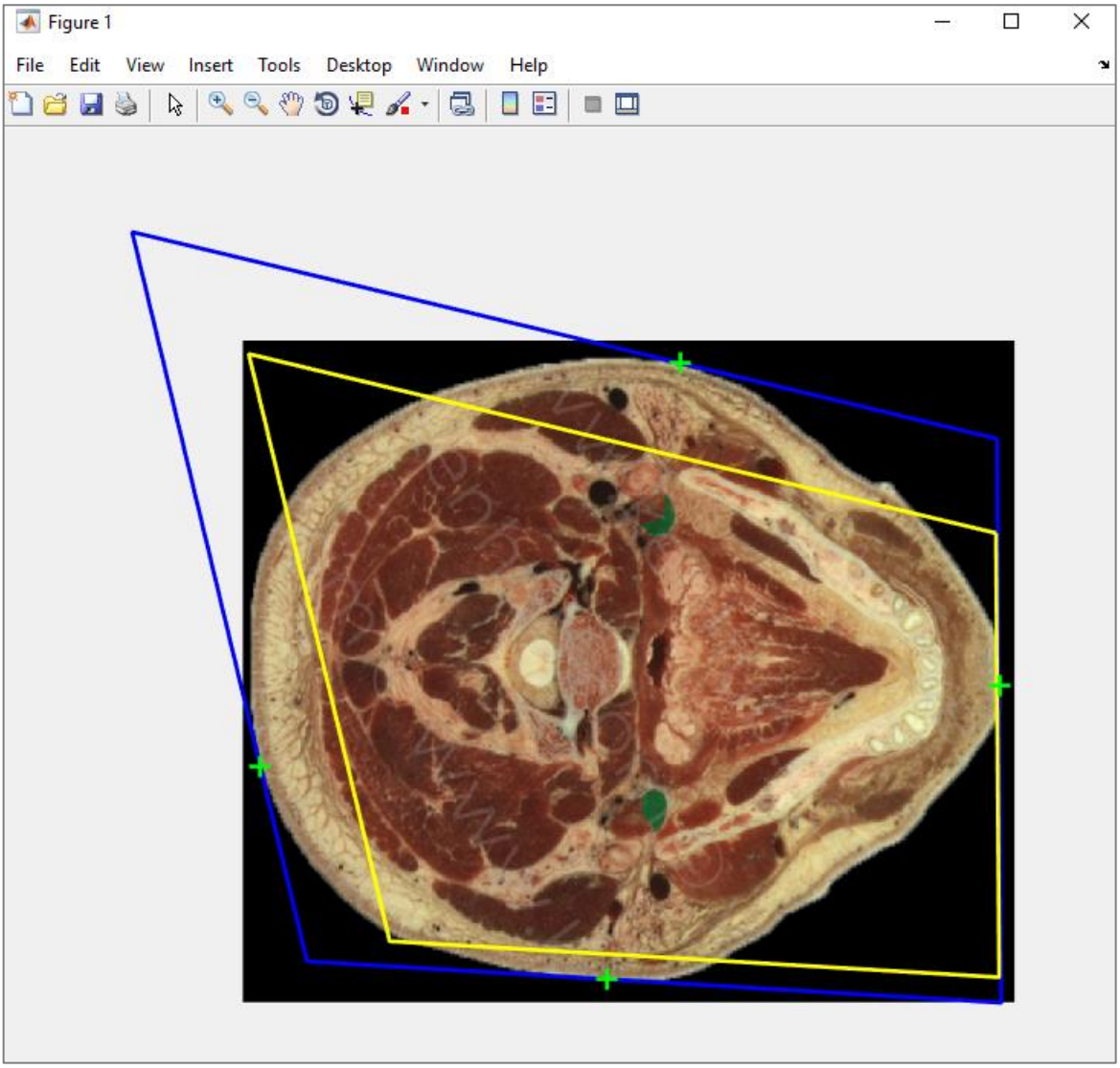


Рис.21. Данные, которые были отправлены в рабочее пространство MATLAB, отображены для наглядности проведения конечных преобразований. Четырёхугольник *ABCD* представлен на изображении жёлтым цветом, четырёхугольник $A_1B_1C_1D_1$ – синим цветом. Зелёным отмечены точки касания среза и четырёхугольника $A_1B_1C_1D_1$.

Помимо данных о четырёхугольниках, в рабочее пространство также переданы данные о сдвиге изображения, которое произошло после проведения операций изменения длины и ширины изображения среза, а также переноса четырёхугольника *ABCD* в начало координат.

Первым этапом является получение аффинного преобразования(affine transformation, tform1) по имеющейся матрице,

составленной с использованием параметра k, полученного после решения системы на предыдущем шаге

$$\text{Atransf} = \begin{pmatrix} k & 0 & 0 \\ 0 & k & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Затем мы трансформируем изображение среза в соответствии с геометрической трансформацией, определённой tform1, используя функцию imwarp:

```
tform1  = affine2d([k 0 0; 0 k 0; 0 0 1]);
J = imwarp(I,tform1);
```

После проделанных преобразований получим следующий результат, представленный на рис. 22:

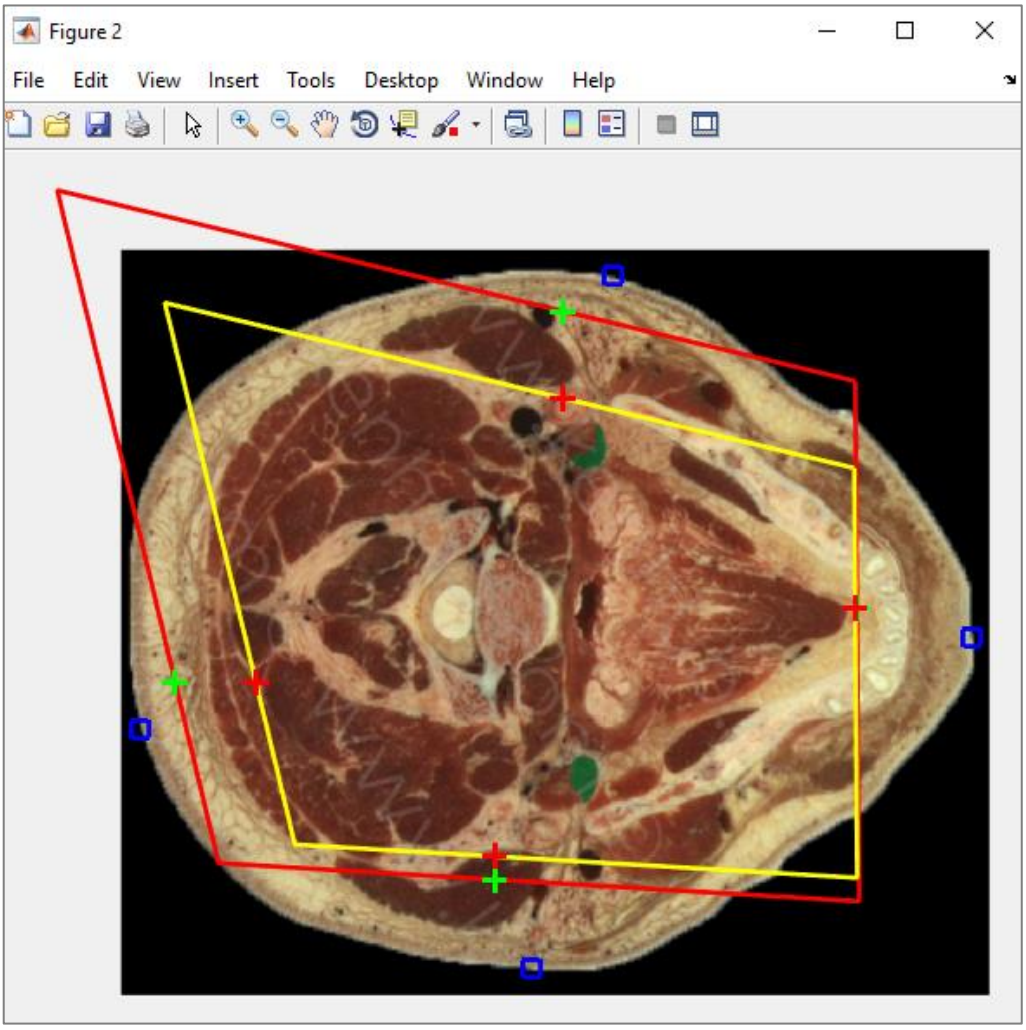


Рис.22. После применения аффинного преобразования к изображению среза.

Однако найденный коэффициент для аффинного преобразования не даёт нужного нам результата. Поэтому необходима дополнительная трансформация изображения, которая осуществляется путём сопоставления расстояния от крайних точек среза до точек предполагаемого расположения граничных точек среза на границах четырёхугольника *ABCD* (на рис. 22 крайние точки среза отмечены синим цветом, точки соприкосновения среза до трансформации отмечены зелёным цветом, точки предполагаемого расположения граничных точек среза на границах четырёхугольника *ABCD* отмечены красным цветом). После того, как нашли расстояние для каждой из четырёх сторон, вычисляется величина, на которую изображение среза необходимо расширить или сжать по каждой из осей. Параллельно с этим вычисляется вектор сдвига, который получается при изменении размера среза.

Конечный результат работы можно увидеть на рис. 23:

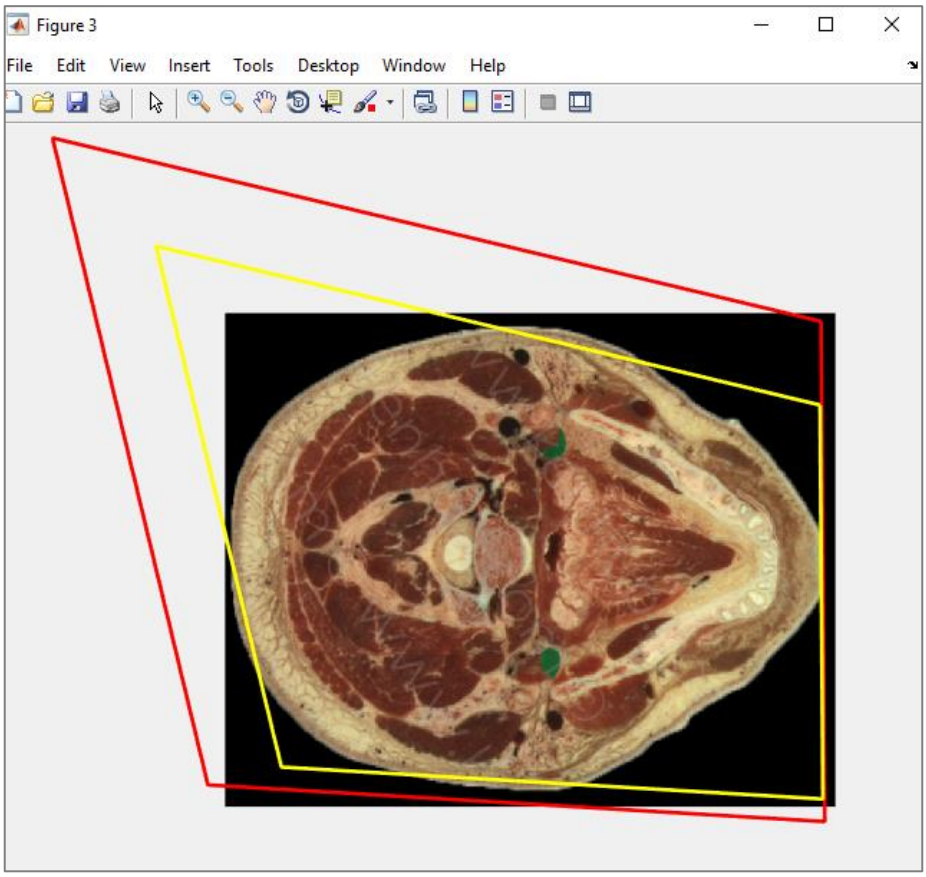


Рис.23. Конечный результат работы алгоритма.

После отработки алгоритма получаем изображение среза, а также записанные в файл данные об уровне и векторе сдвига данного среза.

Безусловно, иногда срез вписывается в четырёхугольник с небольшой погрешностью (как это можно видеть на рис. 23).

Глава 3. Построение трёхмерной модели

В данной главе последовательно описываются этапы построения объёмной модели. Приведено описание работы с разработанным интерфейсом.

3.1. Подготовка данных для отображения в объёмной модели

После того, как был проведён запуск программы для нескольких срезов, которые являются частью общей модели, в специальной папке формируется набор данных, состоящий из обработанного изображения среза в формате png и файла с идентичным названием в формате txt, в котором указаны такие параметры, как уровень расположения среза(level) и вектор сдвига в плоскости z = level.

Первым этапом станет обработка набора срезов, получение информации об альфа-канале и контуре каждого среза. Для этого считываем обработанное изображение среза и создаём три единичные матрицы, размеры которых совпадают по размерности с матрицей изображения среза, а затем действуем по следующему алгоритму:

- Поскольку базовый фон изображений чёрный, то необходимо найти пиксели на изображении среза, где значение каждого из трёх цветовых каналов равно нулю. В том случае обнаружения подходящего пикселя, заменяется значение соответствующего пикселя в созданных единичных матрицах:

```
A1 = ones(size(img));
A2 = ones(size(img));
A3 = ones(size(img));
A1(img(:,:,1)==0)=0;
A2(img(:,:,2)==0)=0;
A3(img(:,:,3)==0)=0;
```

- Формируем итоговую матрицу прозрачности:

```
A = A1+A2+A3;
A= A(:,:,1);
```

Затем временно сохраняем срез с наложенной на него найденной маской прозрачности (см. рис. 24).

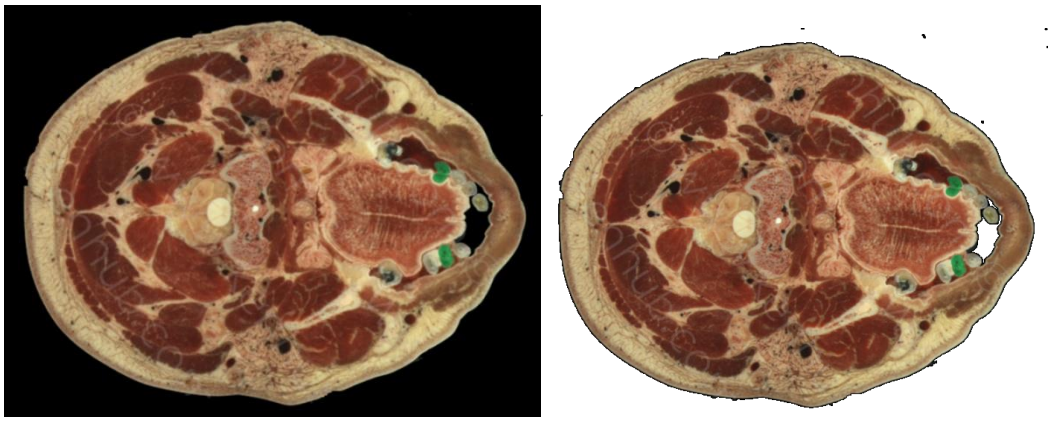


Рис. 24. Пример получения изображения среза с найденной маской прозрачности.

Считываем изображение снова, но уже для того, чтобы сохранить срез в массиве ячеек, куда будут помещаться все срезы, обработанные ранее описанным алгоритмом. Также в отдельный массив ячеек помещается альфа-канал изображения среза.

Для формирования облака точек (3D Point Cloud), по которому потом будет проводиться построение границ объёмной модели, необходимо найти точки границы для каждого среза. Для этой цели прекрасно подошёл алгоритм обработки рентгенограмм: изменяем интенсивность, переводим изображение в бинарное, заполняем пустоты на изображении (см. рис. 25):

```
J = imadjust(img, [0 75]/255, [ ], 1);
bw = im2bw(J);
bw = imfill(bw,'holes');
```

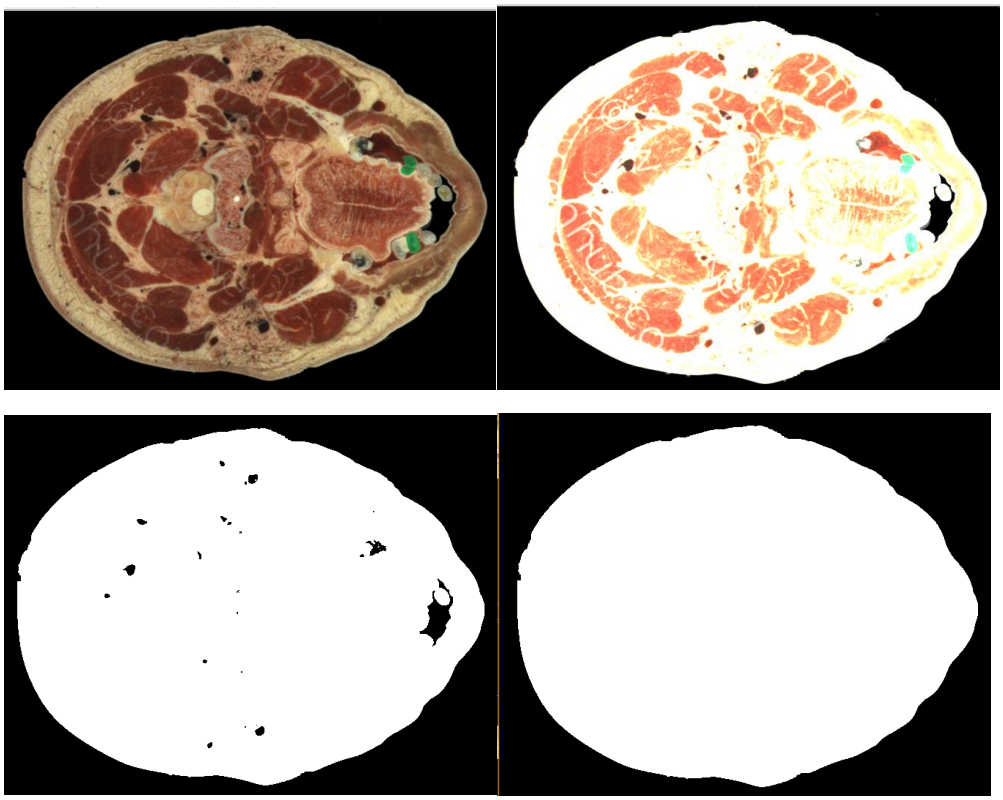


Рис.25. Последовательные этапы обработки изображения среза.

Затем находим границу для данного среза, используя модифицированный алгоритм отслеживания границ (the Moore-Neighbor tracing algorithm) с использованием критерия остановки Якоба (Jacob's stopping criteria), и сохраняем полученные координаты точек в специальный массив ячеек.

Таким образом, получаем своего рода облако точек, по которому будет происходить построение контура объёмной модели.

3.2. Этапы создания объёмной модели

Теперь переходим к следующему шагу: построение самой модели.

Первым этапом будет сортировка всех срезов по уровням, начиная от самого нижнего и заканчивая срезом, уровень расположения которого наибольший. Это необходимо для корректной отработки следующего шага.

Далее проводим «натягивание» контура в пространство между срезами. В качестве варианта построения контура был взят алгоритм триангуляции Делоне, который последовательно применялся для каждых двух соседних срезов. Данный метод построения модели поверхности обычно применяется при моделировании земной поверхности, карт местности, конкретных объектов (см. рис. 26, 27).

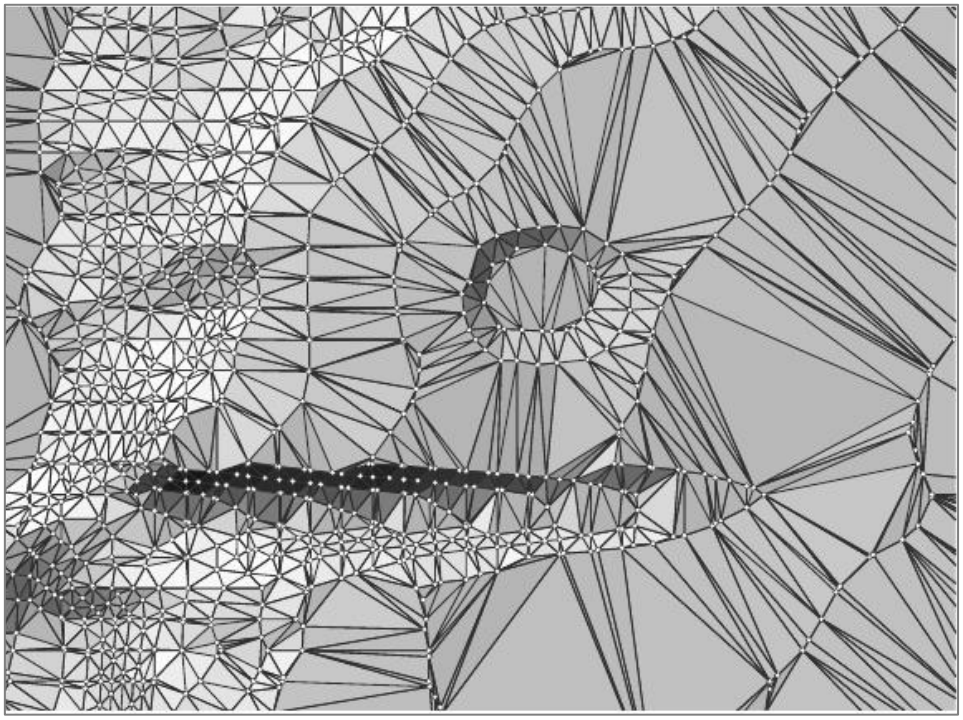
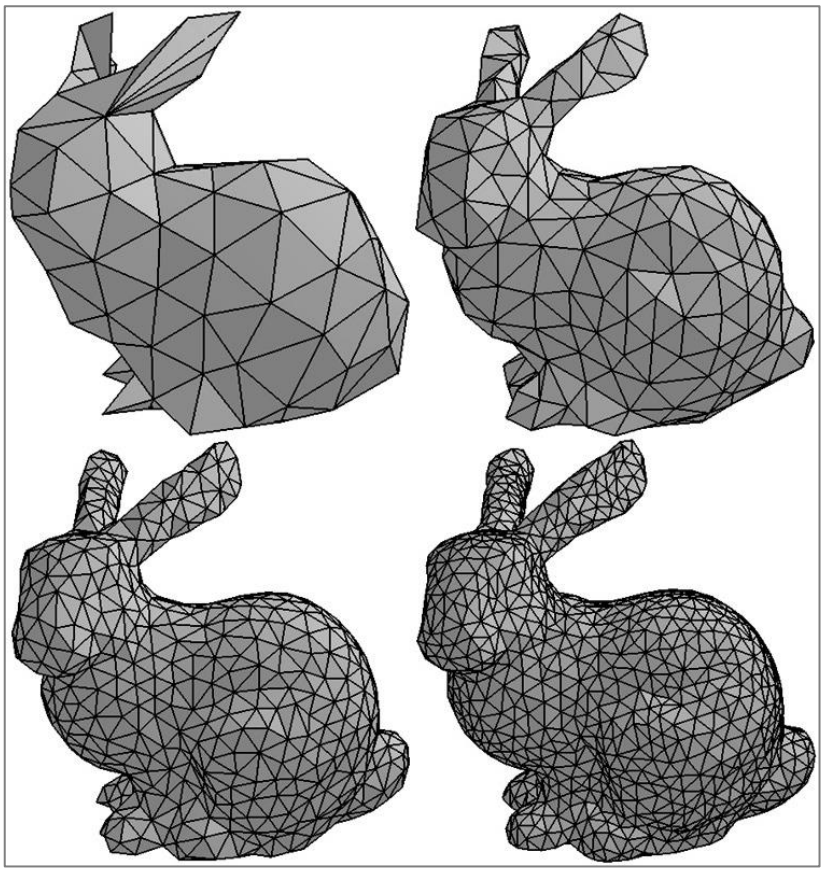
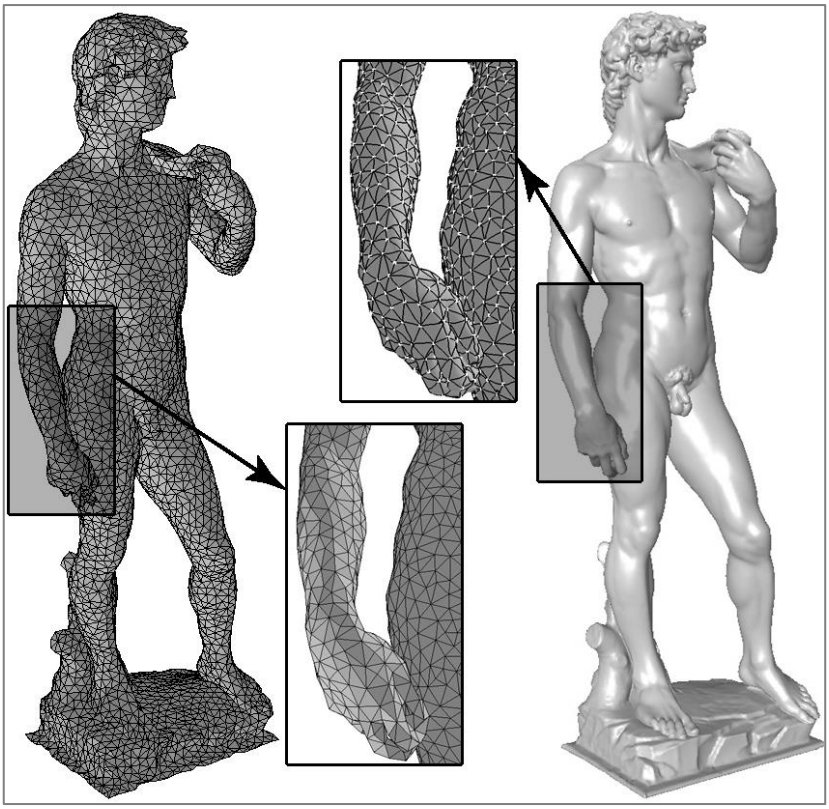


Рис. 26. Триангуляционная модель поверхности Земли. Изображение взято из книги [22].





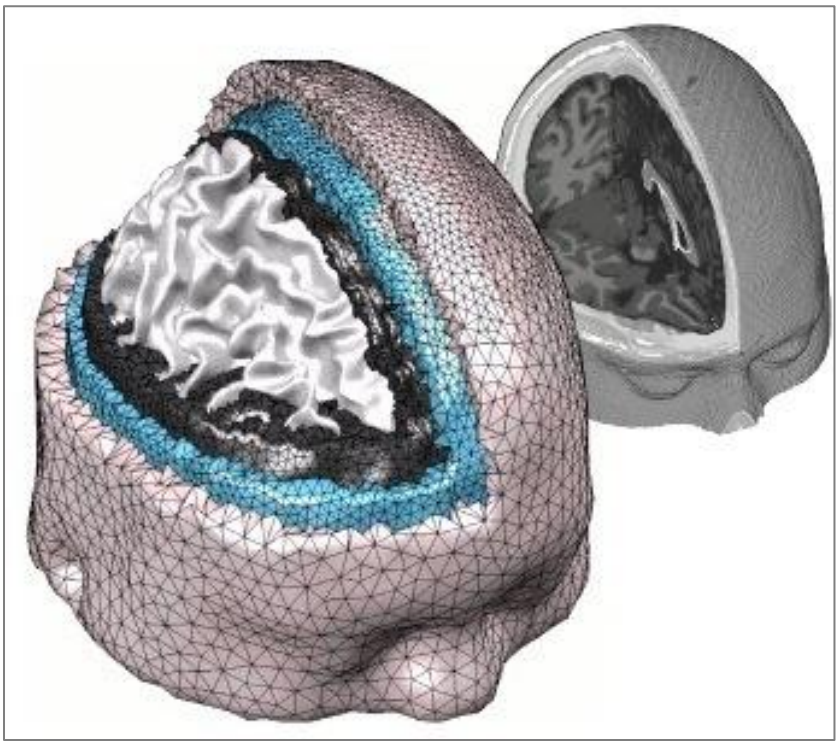


Рис. 27. Представлены примеры получения моделей с использованием геодезических вычислений в 3D-сетках. Изображения взяты из источников [23] и [24].

В данном случае триангуляция проводится для каждых двух соседних срезов, поскольку при построении на всей совокупности точек происходит крайне грубая аппроксимация границ, особенно если модель включает в себя участки, которые имеют вогнутую форму. Таким образом, перед нами стоит задача построения триангуляционной модели поверхности тела человека, где в качестве исходных данных берутся трёхмерные точки в пространстве [22].

Для получения триангуляции между двумя срезами составляется облако точек, состоящее из координат точек контуров срезов в пространстве. Координаты по каждой из осей записываются в отдельные массивы, которые затем передаются в функцию DelaunayTri:

```
DT{k} = DelaunayTri(Xx',Yy',Zz');
```

На выходе получаем сетку, на которой затем находим выпуклые грани:

```
hullFacets = convexHull(DT{k});
```

Затем строим трёхмерную выпуклую оболочку, устанавливая среднюю прозрачность для неё (см. рис. 28):

```
hh=trisurf(hullFacets,DT{k}.X(:,1),DT{k}.X(:,2),
DT{k}.X(:,3),'FaceColor',[1,.75,.65],
'FaceAlpha',.5);
```

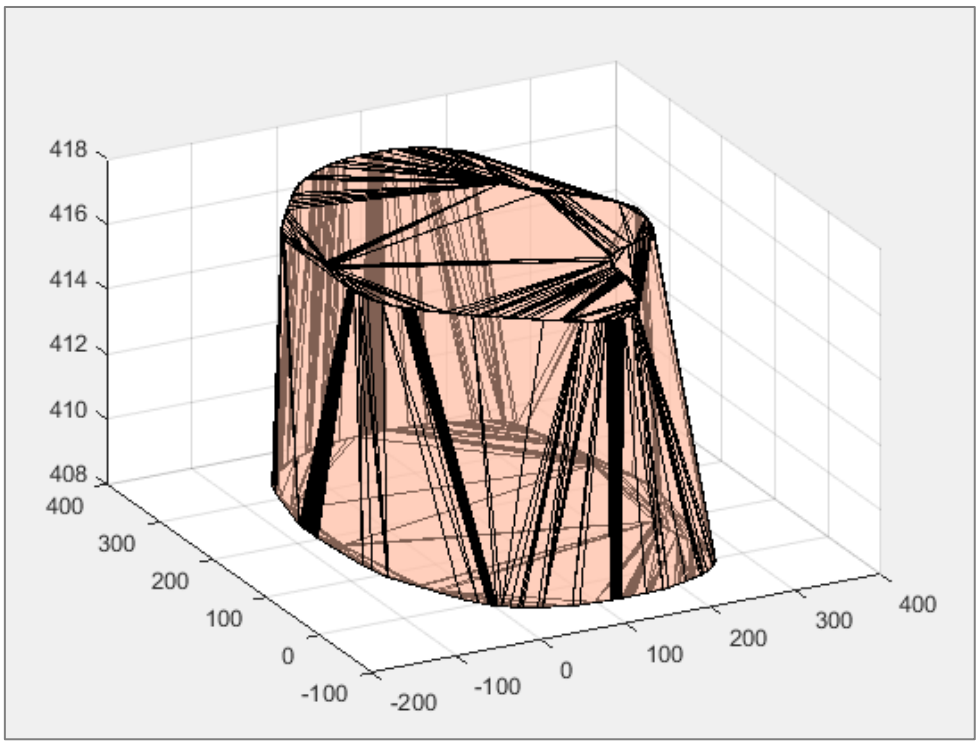


Рис. 28. Результат построения оболочки для двух срезов.

Затем убираются границы треугольников (ярко выраженные на рис. 28) и итоговая триангуляция выглядит следующим образом (рис. 29):

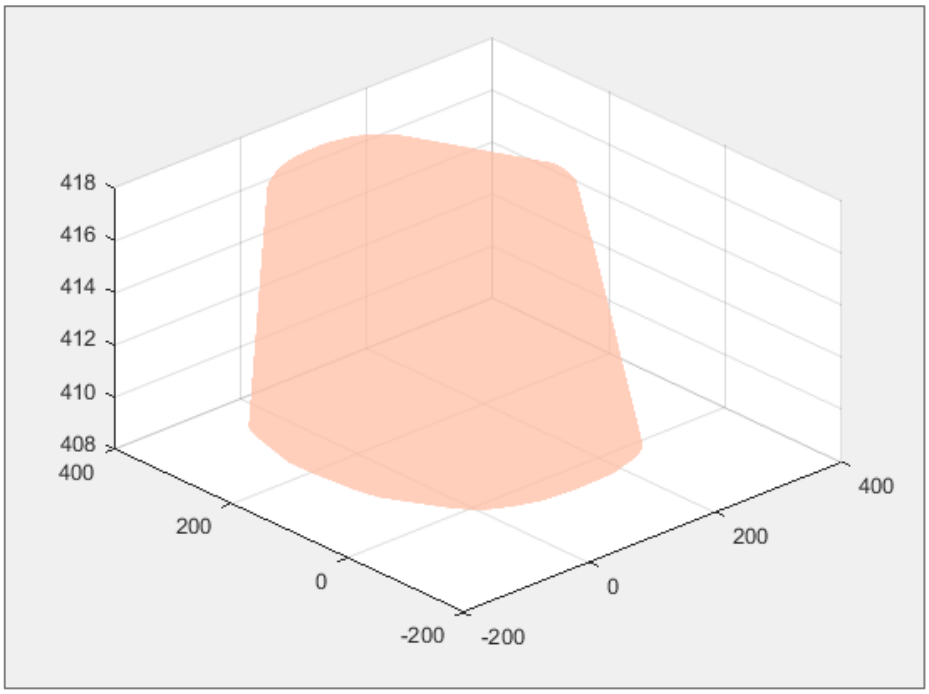


Рис. 29. Итоговая триангуляция для двух срезов.

После выполнения описанных выше действий для построения триангуляции между всеми соседними срезами получаем результат, представленный на рис. 30:

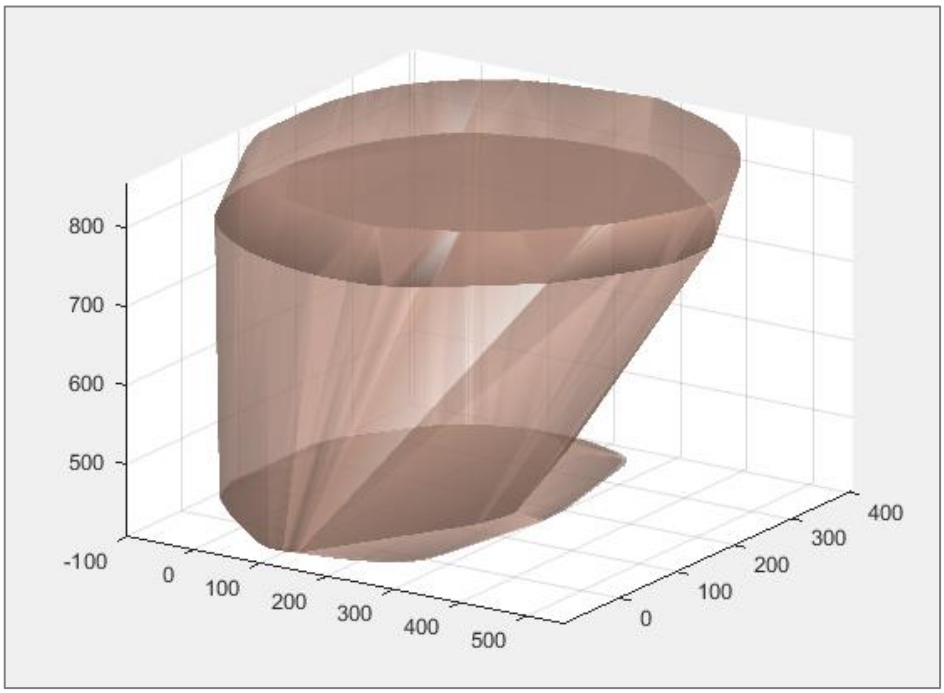


Рис. 30. Результат выполнения операции построения поверхности для всех соседних пар срезов.

Затем накладываем известные изображения срезов на плоскости, соответствующие заданному уровню расположения среза. В качестве смещения относительно точки начала координат в плоскости OXY берётся вектор сдвига, который был считан из файла. Помимо самого изображения также используем сохранённый альфа-канал для каждого среза, чтобы сделать пространство вокруг самих срезов на объёмной модели прозрачным (см. рис. 31). Это можно сделать при помощи следующего фрагмента кода, используя функцию surf:

```
hobj = surf([-vecY(l) -vecY(l)+size(planeimg,2)],...
    [-vecX(l) -vecX(l)+size(planeimg,1)],...
    repmat(imgzposition, [2 2]),planeimg,...
    'FaceColor','texturemap', 'EdgeColor','none',
    'FaceAlpha','texture','AlphaData',
    alphaChannel{l});
```

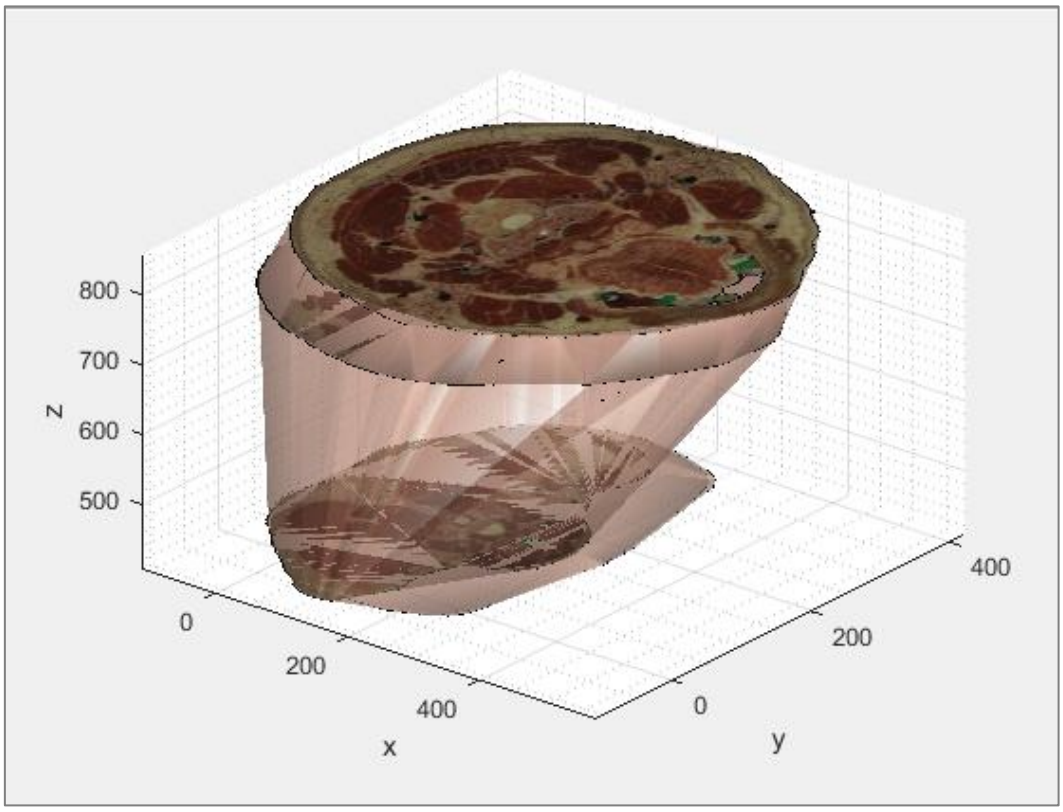


Рис. 31. Итоговая модель.

Стоит отметить, что для самого нижнего и самого верхнего срезов модели необходимо произвести смещение по оси OZ для отображения на поверхности построенной поверхности. Это необходимо, поскольку происходит совмещение результата проведённой триангуляции и изображений срезов, из-за чего корректность отображения указанных срезов нарушена.

3.3. Описание работы с интерфейсом в среде MATLAB

Помимо основного интерфейса для работы с алгоритмом обработки срезов, в рамках данной работы было разработано приложение с графическим интерфейсом пользователя в среде MATLAB. Внешний вид окна представлен на рис. 32 ниже:

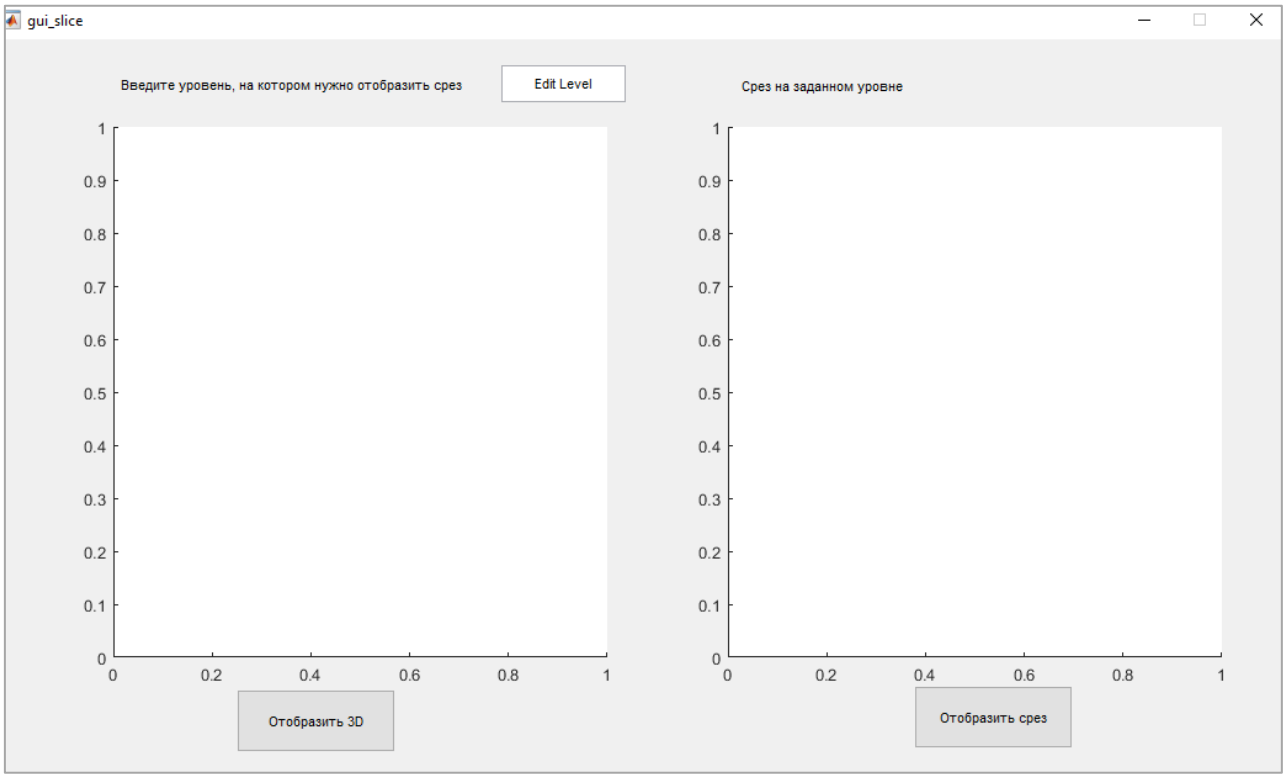


Рис. 32. Основное окно gui_slice приложения.

Кнопка «Отобразить 3D» предназначена для запуска скрипта, который осуществляет построение объёмной модели, которая будет отображена в первой паре осей (modelob.m).

Текстовое поле «**Введите уровень, на котором нужно отобразить срез**» предназначено для ввода значения уровня, на котором расположен срез, который пользователь хочет увидеть на соседней паре осей. После того, как значение введено с клавиатуры, необходимо нажать кнопку «**Отобразить срез**». Произойдёт запуск скрипта find_slice_between.m. После чего произойдёт отображение готового среза, если такой был предоставлен в наборе данных, подготовленных для построения модели (см. рис. 33).

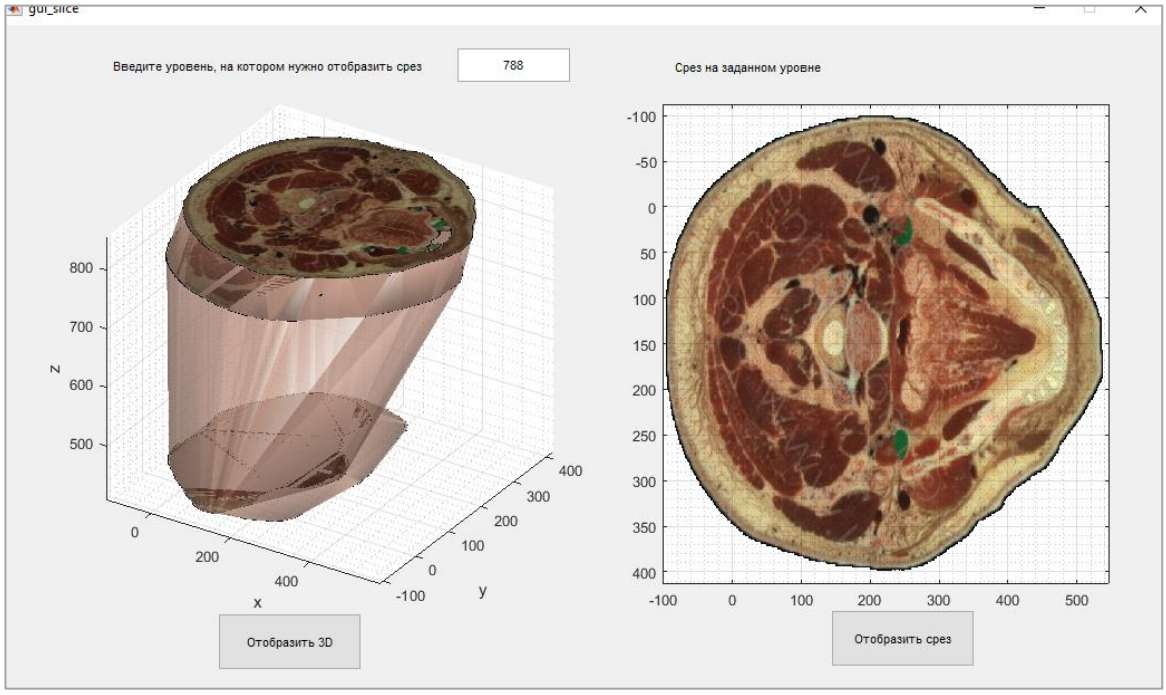


Рис.33. Работа программы в случае наличия готового изображения среза.

В ином случае, если расстояние между срезами небольшое, то произойдёт запуск скрипта, отвечающего за проведение реконструкции изображений двух срезов, между которыми расположен предполагаемый срез. Если же расстояние большое (больше заранее заданной величины), то проведение реконструкции является нецелесообразным, поскольку речь идёт о построении модели тела человека, где внутренняя структура является сложной.

Для обоих вариантов имеет смысл провести операцию построения среза построенной модели на выбранном уровне. Для этого строится плоскость z = level, находится интерполяционная функция

```
F = TriScatteredInterp(DT{k}.X(:,1),DT{k}.X(:,2),
DT{k}.X(:,3));
```

Соответствующее значение найденной функции на выбранной плоскости может быть найдено следующим образом:

```
qz = F(x, y);
```

После этого ищем контуры для каждого уровня. Для этого рисуем линии уровня между каждыми двумя срезами, которые имеют готовые обработанные изображения.

```
[C,ha] = contour(x, y, qz, vecLevel(k+1)-
vecLevel(k)-1,'visible','off');
```

Чтобы ускорить работу выполнения программы рекомендуется распараллелить вычисления для каждого из уровней между двумя срезами. MATLAB предлагает гибкую модель для организации распределенных вычислений и вычислений. В данной реализации был использован оператор parfor, для которого были выполнены необходимые требования: независимость от задач и независимость от порядка выполнения.

После проделанных операций, контуры отображаются на второй паре осей (см. рис. 34).

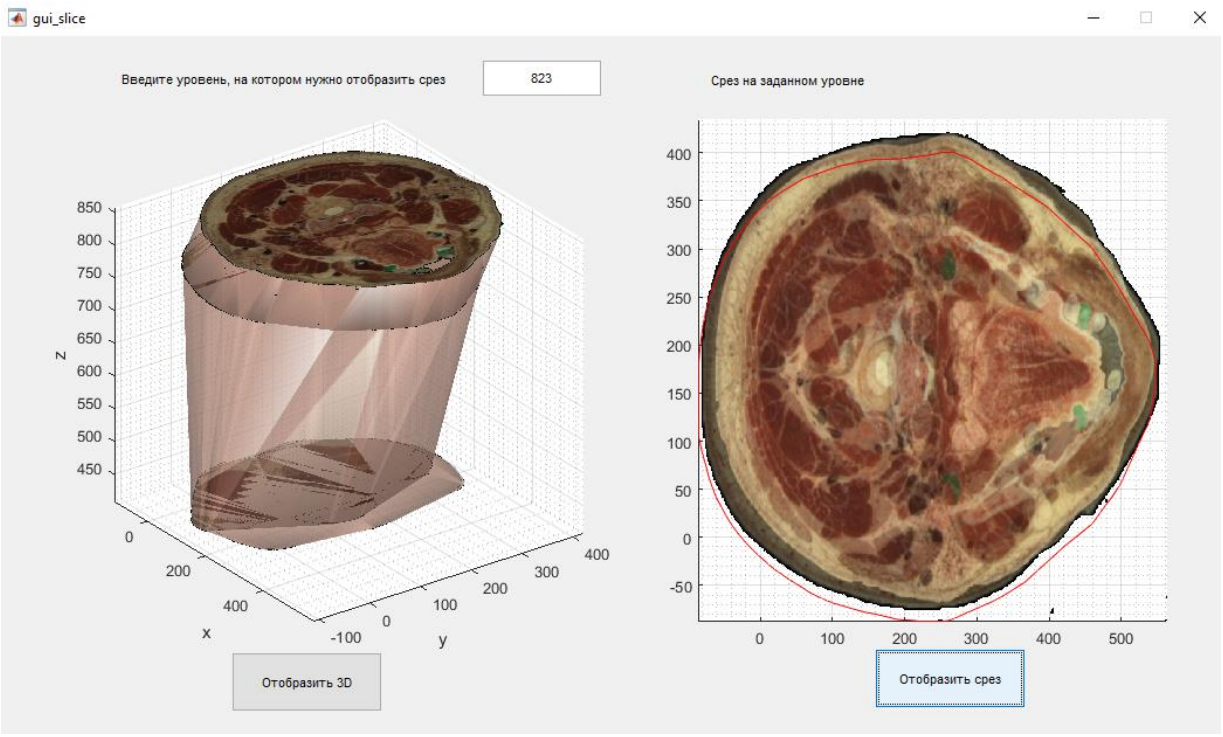


Рис. 34. Проведение реконструкции среза, расположенного между двумя близко расположенными имеющимися срезами. На правой паре осей отображено получившееся изображение с наложенной маской прозрачности. Также отображен контур, который получился в результате пересечения объёмной модели плоскостью на заданном уровне.

Выводы

Для достижения цели работы был разработан один из возможных вариантов решения, который включает в себя две отдельных задачи:

- нахождение пространственных координат и проведение операций деформации над срезами из анатомического атласа;
- построение трёхмерной модели с использованием данных, полученных в первой задаче.

На основании проведённого исследования были сделаны следующие выводы, касающиеся каждого описанного ранее этапа получения объёмной модели исследуемой области тела человека.

Прежде всего, отдельной задачей стояла обработка рентгенограмм. Эта задача требовала изучения методов обработки изображений, в частности рентгенограмм. В результате исследования был подобран подходящий способ обработки рентгеновских снимков.

В ходе реализации в качестве тестовых изображений были взяты снимки, доступные в сети Интернет. Кроме того, каждый тестовый срез из анатомического атласа был предварительно обработан в программе Photoshop, поскольку изображения из анатомического атласа содержали водяные знаки.

Вследствие этого, более точные результаты, необходимые при практическом использовании реализованного программного решения, могут быть получены при наличии хорошего анатомического атласа, который был бы переведён в цифровую форму.

Помимо этого, в сделанной реализации включены данные лишь нескольких аппаратов, без возможности добавления новых со своими параметрами. Это является недостатком и требует внесения изменений в сделанную реализацию.

Предлагаемый вариант реализации поставленной задачи был программно реализован (см. Приложение) и детально описан. Помимо этого, для удобства работы с программой были реализованы пользовательские интерфейсы с использованием средств разработки JavaFX и MATLAB.

Заключение

Построение объёмной модели тела пациента - нетривиальная задача, решение которой может позволить улучшить качество диагностики заболеваний.

В настоящей работе предложен вариант реализации алгоритма получения данных для построения объёмной модели и сам вариант её построения с использованием MATLAB. Предложен способ обработки рентгенограмм с учётом особенностей и проблем, которые возникают при попытках нахождения контура на подобном типе изображений.

Помимо этого, были созданы процедурно-ориентированные интерфейсы для обеих поставленных задач, которые позволяют облегчить работу с программой обычному пользователю.

В заключение, хотелось бы добавить, что данная рассмотренная тема имеет широкий спектр применения, но требует дополнительного исследования, поскольку одной из главных проблем является сложность получения тестовых данных, поскольку рентгенограммы представляют собой персональные данные пациента. Помимо этого, имеет смысл использовать алгоритм, описанный в статье [9], для корректировки контуров внутренних органов.

Более того, в настоящее время есть тенденция отхождения от применения рентгенографии. Несмотря на то, что данный метод лучевой диагностики самый простой и недорогой, он имеет ряд недостатков, таких как облучение, нечёткость получаемых снимков. Следовательно, имеет смысл адаптировать реализованные алгоритмы получения облака точек и построение объёмной модели для иных видов лучевых диагностик, например цифрового рентгена. В сравнении с традиционной технологией (пленочной) используемые цифровые рентгеновские датчики существенно уменьшают дозу рентгеновского облучения (на 50-70%, в отдельных случаях - до 90%) [25].

Список литературы

1. Sergeev S. L., Stuchenkov A. B. An algorithm of deformation of a flat image // 2014 International Conference on Computer Technologies in Physical and Engineering Applications (ICCTPEA). 2014. P. 159.
2. Sergeev S. L., Stuchenkov A. B. Modeling of deformation of an elastic body slice // 2014 International Conference on Computer Technologies in Physical and Engineering Applications (ICCTPEA). 2014. P. 158.
3. Глубокая стимуляция головного мозга в Австрии. https://ru.health-tourism.com/deep-brain-stimulation/austria/
4. Goswami B., S. Kr. Misra. 3D Modeling of X-Ray Images: A Review Baishali. International Journal of Computer Applications .Volume 132 – No.7, 2015 – С. 40-46.
5. BioDigital: 3D Human Visualization Platform for Anatomy and Disease. https://www.biodigital.com/developers
6. 3D Slicer. https://www.slicer.org
7. Lamecker H., Wenckebach T. H., Hege H.-C. Atlas-based 3D shape reconstruction from X-ray images. Proceeding ICPR '06 Proceedings of the 18th International Conference on Pattern Recognition. V.1, 2006 – С. 371-374.
8. Karade V., Ravi B. 3D femur model reconstruction from biplane X-ray images: a novel method based on Laplacian surface deformation. International Journal of Computer Assisted Radiology and Surgery, 2015. 10(4). С. 473–485.
9. Сергеев С. Л., Севрюков С. Ю. Использование метода упругой пленки в программном комплексе построения среза тела пациента // Вестник Санкт-Петербургского университета. Серия 10: Прикладная математика. Информатика. Процессы управления. 2010. № 1. С. 73–79.

10. Launch4j - Cross-platform Java executable wrapper. http://launch4j.sourceforge.net
11. Стученков А.Б., Улитина И.А., Построение изображений по проекциям с использованием анатомического атласа // Процессы управления и устойчивость, 2017. Т.4, №1. – в печати.
12. Доля П.Г. Методы обработки изображений. Харьковский Национальный Университет, Механико – математический факультет. 2013 – 46 с.
13. Гонсалес Р., Вудс Р., Эддинс С. Цифровая обработка изображений в среде MATLAB. Техносфера, 2006 – 616 с.
14. Методы компьютерной обработки изображений / Под ред. В.А.Сойфера, 2-е изд, испр.- ФИЗМАТЛИТ, 2003 − 784 с.
15. Moore neighborhood. https://en.wikipedia.org/wiki/Moore_neighborhood
16. Contour Tracing. http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/moore.html
17. Радиография - Сайт Врачей-Радиологов. Medical Radiologists. http : //radiographia . ru/node/9069
18. Cross Sections - Online Atlas with Pictures and Diagrams. https : //www . kenhub . com/en/atlas?lecture=cross-sect ions
19. Atlas of Human Anatomy in Cross Section. http://www.anatomyatlases.org/HumanAnatomy/CrossSectionAtlas.shtml
20. The Visible Human Server at the EPFL. http://visiblehuman.epfl.ch
21. Характеристики рентгенофлюорографических комплексов для исследования грудной клетки. http : //www.renex.ru/texts/vibor_fluorograph_tabl.htm
22. Скворцов А.В. Триангуляция Делоне и её применение. / Томск: Изд-во Том. ун-та, 2002ю – 128с.

23. Geodesic Computations on 3D Meshes.
http://www.cmap.polytechnique.fr/~peyre/geodesic_computations/
24. Iso2mesh: a 3D surface and volumetric mesh generator.
http://iso2mesh.sourceforge.net/cgi-bin/index.cgi
25. Цифровой рентген.
https://www.mc21.ru/adult/diagnostics/rentgenograpy/

Приложение

Код для решения первой задачи.

SliceMain.cpp

```
#include"SliceMain.h"
using namespace cv;

/**
	Функция, которая сохраняет изображение в др формате
*/
void SliceMain::convertToMat(){
	imageMatrix = cvarrToMat(this->Image);
}
/**
	Функция, которая ищет крайние точки определенного слоя (по оси x)
*/
void SliceMain::findPointsOnTomography(CvSeq* contoursOne, CvSeq*
contoursTwo){

	for (int i = 0; i < 4; i++){
		yImage[i] = level;
	}

	CvSeq *masCont[2] = { contoursOne, contoursTwo };

	CvPoint* p, *p1;
	int count = 0;
	int limitPointsForOnePicture = 1;

	for (int k = 0; k < 2; k++){
		for (CvSeq* seq0 = masCont[k]; seq0 != 0; seq0 = seq0-
>h_next){
			for (int i = 0; i < seq0->total; i++) {
				p = (CvPoint*)cvGetSeqElem(seq0, i);
				//printf(" %d ", p->y);
					if (abs(p->y - this->yImage[count]) == 0
&& count <= limitPointsForOnePicture){
						if (count % 2 == 1 &&
abs(xImage[count - 1] - p->x) > 50 || count % 2 == 0){
							this->xImage[count] = p-
>x;
							count++;
```

```
                                        }
                                }
                        }
                }
                limitPointsForOnePicture = 3;
        }

        // меняем точки на найденном изображении, чтобы
        //    правильно потом построить четырёхугольник
        double tmp = xImage[0];
        xImage[0] = xImage[1];
        xImage[1] = tmp;

}

/**
        Функция, которая ищет точки четырёхугольника
*/
void SliceMain::findQuadrangle(){

        b[0] = xImage[2];
        k[0] = (s1.y - b[0]) / s1.x;
        b[1] = xImage[3];
        k[1] = (s1.y - b[1]) / s1.x;

        k[2] = s2.y / (s2.x - xImage[0]);
        b[2] = -xImage[0] * k[2];
        k[3] = s2.y / (s2.x - xImage[1]);
        b[3] = -xImage[1] * k[3];

        A.x = -(b[0] - b[2]) / (k[0]-k[2]);
        A.y = k[0] * A.x + b[0];
      B.x = -(b[0] - b[3]) / (k[0] - k[3]);
        B.y = k[0] * B.x + b[0];
        C.x = -(b[1] - b[3]) / (k[1] - k[3]);
        C.y = k[1] * C.x + b[1];
        D.x = -(b[1] - b[2]) / (k[1] - k[2]);
        D.y = k[1] * D.x + b[1];
}

/**
        Функция, которая обновляет значения в том
```

```
                        или ином формате записи точек
                                в зависимости от флага
*/
void SliceMain::refreshValueOfPoints(int flag){
      if (flag == 1){
            xSlice[0] = A.x;
            ySlice[0] = A.y;
            xSlice[1] = B.x;
            ySlice[1] = B.y;
            xSlice[2] = C.x;
            ySlice[2] = C.y;
            xSlice[3] = D.x;
            ySlice[3] = D.y;
      }
      else{
            A.x = xSlice[0];
            A.y = ySlice[0];
            B.x = xSlice[1];
            B.y = ySlice[1];
            C.x = xSlice[2];
            C.y = ySlice[2];
            D.x = xSlice[3];
            D.y = ySlice[3];
      }
}

/**
      Функция, которая ищет прямоугольник вокруг четырёхугольника
*/
void SliceMain::findReqtangleAroundQuadr(){

      refreshValueOfPoints(1);

      double maxi[4];
      maxi[0] = xSlice[0];//min x
      maxi[1] = xSlice[1];//max x
      maxi[2] = ySlice[0];//min y
      maxi[3] = ySlice[1];//max y
      // ищем наименьшие вторую и первую координаты,
      //       наибольшие вторую и первую координату
      for (int j = 0; j < 4; j++){
            if (this->xSlice[j] < maxi[0])
```

```
                maxi[0] = xSlice[j];
            if (this->xSlice[j] > maxi[1])
                maxi[1] = xSlice[j];
            if (this->ySlice[j] < maxi[2])
                maxi[2] = ySlice[j];
            if (this->ySlice[j] > maxi[3])
                maxi[3] = ySlice[j];
        }
        // из полученных четырех значений получаем четыре координаты

        this->xQuadr_rect[0] = maxi[0]; this->yQuadr_rect[0] = maxi[2];
        this->xQuadr_rect[1] = maxi[0]; this->yQuadr_rect[1] = maxi[3];
        this->xQuadr_rect[2] = maxi[1]; this->yQuadr_rect[2] = maxi[3];
        this->xQuadr_rect[3] = maxi[1]; this->yQuadr_rect[3] = maxi[2];
}

/**
        Функция, которая ищет прямоугольник вокруг среза
*/
void SliceMain::findReqtangleAroundSlice(){

        double maxi[4];
        CvPoint* p;
        maxi[0] = ((CvPoint*)cvGetSeqElem(this->contour, 0))->x;//min x
        maxi[1] = ((CvPoint*)cvGetSeqElem(this->contour, 1))->x;//max x
        maxi[2] = ((CvPoint*)cvGetSeqElem(this->contour, 0))->y;//min y
        maxi[3] = ((CvPoint*)cvGetSeqElem(this->contour, 1))->y;//max y

        for (int i = 0; i < this->contour->total; i++) {
            p = (CvPoint*)cvGetSeqElem(this->contour, i);
            if (p->x<maxi[0]){
                maxi[0] = p->x;
            }
            if (p->y<maxi[2]){
                maxi[2] = p->y;
            }
            if (p->x>maxi[1]){
                maxi[1] = p->x;
            }
            if (p->y>maxi[3]){
                maxi[3] = p->y;
            }
```

```
	}
	// из полученных четырех значений получаем четыре координаты
	this->xSlice_rect[0] = maxi[0]; this->ySlice_rect[0] = maxi[2];
	this->xSlice_rect[1] = maxi[0]; this->ySlice_rect[1] = maxi[3];
	this->xSlice_rect[2] = maxi[1]; this->ySlice_rect[2] = maxi[3];
	this->xSlice_rect[3] = maxi[1]; this->ySlice_rect[3] = maxi[2];

}

/**
	Функция, которая рисует четырехугольник по заданным массивам
координат
*/
void SliceMain::drawQuadrang(double ArrReqX[], double ArrReqY[],
IplImage* ImageName){
	CvPoint point_first;
	CvPoint point_second;

	for (int i = 0; i < 4; i++){
		if (i < 3){
			point_first.x = ArrReqX[i];
			point_first.y = ArrReqY[i];
			point_second.x = ArrReqX[i + 1];
			point_second.y = ArrReqY[i + 1];
		}
		else{
			point_first.x = ArrReqX[3];
			point_first.y = ArrReqY[3];
			point_second.x = ArrReqX[0];
			point_second.y = ArrReqY[0];

		}
		cvLine(ImageName, point_first, point_second, CV_RGB(255, 0,
0), 2, 8, 0);
	}

}
/**
	Функция, которая находит прямоугольники
		и обновляет данные о них + рисует
*/
void SliceMain::refreshAndDrawReqt(IplImage* ImageName){
```

```
	findReqtangleAroundQuadr();
	findReqtangleAroundSlice();

	drawQuadrang(xQuadr_rect, yQuadr_rect, ImageName);
	drawQuadrang(xSlice_rect, ySlice_rect, ImageName);
}

/**
	Функция, которая показывает изначальное положение
			источников, четырёхугольник и срез,
			  с которым далее будет идти работа
*/
void SliceMain::showStartImage(IplImage* ImageBefore){

	refreshAndDrawReqt(ImageBefore);
	drawContour(ImageBefore);
	cvLine(ImageBefore, Point(this->xImage[0], 0), s2, Scalar(255, 0,
255), 2, 8);
	cvLine(ImageBefore, Point(this->xImage[1], 0), s2, Scalar(255, 0,
255), 2, 8);
	cvLine(ImageBefore, s1, Point(0, this->xImage[2]), Scalar(255, 0,
255), 2, 8);
	cvLine(ImageBefore, s1, Point(0, this->xImage[3]), Scalar(255, 0,
255), 2, 8);
	cvNamedWindow("BeforeResize", CV_WINDOW_AUTOSIZE);
	cvShowImage("BeforeResize", ImageBefore);
}


/**
	Функция, которая показывает текущее состояние изображения
*/
void SliceMain::showProcessImage(String label, IplImage* ImageName){
	String windowName = "ProcessImage" + label;
	drawQuadrang(xSlice, ySlice, ImageName);

	cvNamedWindow(windowName.c_str(), CV_WINDOW_AUTOSIZE);
	cvShowImage(windowName.c_str(), ImageName);
}

/**
	Функция, которая подгоняет размер среза под размер прямоугольника,
```

```
                                        описанного вокруг найденного
четырёхугольника
                                      + переносит четырёхугольник в начало
координат
*/
void SliceMain::resizeImageToRectangle(IplImage* littleSlice){

      vector_sdviga[0] = -xQuadr_rect[0] + xSlice_rect[0];
      vector_sdviga[1] = -yQuadr_rect[0] + ySlice_rect[0];

      //сдвигаем прямоугольник, описанный вокруг четырехугольника
      //                                        в начало координат
      for (int i = 0; i < 4; i++){
            xSlice[i] = xSlice[i] + vector_sdviga[0];
            ySlice[i] = ySlice[i] + vector_sdviga[1];
      }

      refreshValueOfPoints(0);
      cvResize(this->dst, littleSlice, 1);

      this->dst = littleSlice;
      this->gray = cvCreateImage(cvGetSize(littleSlice), IPL_DEPTH_8U,
1);
      this->bin =  cvCreateImage(cvGetSize(littleSlice), IPL_DEPTH_8U,
1);
      cvCvtColor(littleSlice, this->gray, CV_RGB2GRAY);
      cvInRangeS(this->gray, cvScalar(70), cvScalar(180), this->bin);
}

/**
      Функция, которая ищет коэффициенты для
              параллельного четырёхугольника
                                  вокруг среза
*/
void SliceMain::findBkoeffForParallel(){
      CvPoint* p;
      double * masBParal = new double[this->contour->total];
      double * masX = new double[this->contour->total];
      double * masY = new double[this->contour->total];
      // по всем четырём сторонам
      for (int j = 0; j < 4; j++) {
            // по каждой точке контура
```

```
            for (int i = 0; i < this->contour->total; i++) {
                  p = CV_GET_SEQ_ELEM(CvPoint, this->contour, i);
                  masBParal[i] = p->y - k[j] * p->x;
                  masX[i] = p->x;
                  masY[i] = p->y;
            }
            double tmp = 0;

            for (int i = 0; i < this->contour->total; i++) {
                  for (int k = i + 1; k < this->contour->total; k++) {
                        if (masBParal[i] > masBParal[k]) {
                              tmp = masBParal[i];
                              masBParal[i] = masBParal[k];
                              masBParal[k] = tmp;

                              tmp = masX[i];
                              masX[i] = masX[k];
                              masX[k] = tmp;

                              tmp = masY[i];
                              masY[i] = masY[k];
                              masY[k] = tmp;
                        }
                  }
                  switch (j) {
                  case 0:
                        if (masY[0] < masY[this->contour->total - 1]) {
                              bParallel[j]        = masBParal[0];
                              xSliceQuadrPoint[j] = masX[0];
                              ySliceQuadrPoint[j] = masY[0];
                        }
                        else {
                              bParallel[j]        = masBParal[this-
>contour->total - 1];
                              xSliceQuadrPoint[j] = masX[this->contour-
>total - 1];
                              ySliceQuadrPoint[j] = masY[this->contour-
>total - 1];
                        }
                        break;
                  case 1:
                        if (masY[0] > masY[this->contour->total - 1]) {
```

```
                              bParallel[j]        = masBParal[0];
                              xSliceQuadrPoint[j] = masX[0];
                              ySliceQuadrPoint[j] = masY[0];
                        }
                        else {
                              bParallel[j]        = masBParal[this-
>contour->total - 1];
                              xSliceQuadrPoint[j] = masX[this->contour-
>total - 1];
                              ySliceQuadrPoint[j] = masY[this->contour-
>total - 1];
                        }
                        break;
                  case 2:
                        if (masX[0] > masX[this->contour->total - 1]) {
                              bParallel[j]        = masBParal[0];
                              xSliceQuadrPoint[j] = masX[0];
                              ySliceQuadrPoint[j] = masY[0];
                        }
                        else {
                              bParallel[j]        = masBParal[this-
>contour->total - 1];
                              xSliceQuadrPoint[j] = masX[this->contour-
>total - 1];
                              ySliceQuadrPoint[j] = masY[this->contour-
>total - 1];
                        }
                        break;
                  case 3:
                        if (masX[0] < masX[this->contour->total - 1]) {
                              bParallel[j] = masBParal[0];
                              xSliceQuadrPoint[j] = masX[0];
                              ySliceQuadrPoint[j] = masY[0];
                        }
                        else {
                              bParallel[j] = masBParal[this->contour-
>total - 1];
                              xSliceQuadrPoint[j] = masX[this->contour-
>total - 1];
                              ySliceQuadrPoint[j] = masY[this->contour-
>total - 1];
                        }
```

```
                        break;
                    }
                }
            }
}


/**
      Функция, которая возвращает отношение сторон
                              двух прямоугольников
*/
double SliceMain::calculateNewSize(int i, int j){
      double up   = pow((xSlice_rect[i] - xSlice_rect[j]),2) +
pow((ySlice_rect[i] - ySlice_rect[j]),2);
      double down = pow((xQuadr_rect[i] - xQuadr_rect[j]),2) +
pow((yQuadr_rect[i] - yQuadr_rect[j]),2);
      return sqrt(up) / sqrt(down);
}

/**
      Функция, предназначенная для отправки данных типа string в Matlab
      На вход:                                                  имя сервера,
                                    желаемое название переменной в
Matlab среде,
                              значение, которое хочешь присвоить этой
переменной
*/
void SliceMain::sendMatrixDataToMatlab(Engine *ep, char* name, double
mas[], int sz){
      mxArray *Koeff = mxCreateDoubleMatrix(1, sz, mxREAL);
      double  *dataPointer = mxGetPr(Koeff);
      memcpy(dataPointer, mas, 1 * sz * sizeof(double));
      engPutVariable(ep, name, Koeff);
}

/**
      Функция, которая отправляет данные в Matlab и возвращает решение
СЛАУ,

полученное в Matlab МНК

*/
```

```
double* SliceMain::findLinearTransformation(){
	double *solution = new double[4];

	Engine *ep = engOpen(NULL); // соединяемся

	// указываем путь до текущей папки Matlab,
	//		где расположены скрипты и функции
	std::string PATH = "cd " + pathMatlab;
	char *MatlabCodeDir = new char[PATH.length() + 1];
	strcpy(MatlabCodeDir, PATH.c_str());
	engEvalString(ep, MatlabCodeDir);

	sendMatrixDataToMatlab(ep, "bParalKoeff", bParallel, 4);
	sendMatrixDataToMatlab(ep, "kKoeff", k, 4);
	sendMatrixDataToMatlab(ep, "bKoeff", b, 4);

	engEvalString(ep, "findSolutionMLS(bParalKoeff,kKoeff,bKoeff)");
	mxArray *x0_y0_k_h = mxCreateDoubleMatrix(1, 4, mxREAL);
	x0_y0_k_h = engGetVariable(ep, "ans");

	double  *resultPointer1 = mxGetPr(x0_y0_k_h);
	double  *resultPointer2 = solution;
	memcpy(solution, resultPointer1, 1 * 4 * sizeof(double));

	// Запускаем вторую часть
	sendMatrixDataToMatlab(ep, "kKoeff", k, 4);
	sendMatrixDataToMatlab(ep, "bKoeff", b, 4);
	sendMatrixDataToMatlab(ep, "bParalKoeff", bParallel, 4);

	sendMatrixDataToMatlab(ep, "xSlice", xSlice, 4);
	sendMatrixDataToMatlab(ep, "ySlice", ySlice, 4);

	sendMatrixDataToMatlab(ep, "xSlicePer", xSlicePer, 4);
	sendMatrixDataToMatlab(ep, "ySlicePer", ySlicePer, 4);

	sendMatrixDataToMatlab(ep, "xSliceQuadrPoint", xSliceQuadrPoint,
4);
	sendMatrixDataToMatlab(ep, "ySliceQuadrPoint", ySliceQuadrPoint,
4);

	sendMatrixDataToMatlab(ep, "x0_y0_k_h", solution, 4);
```

```
	sendMatrixDataToMatlab(ep, "vector_sdviga", vector_sdviga, 2);

	engEvalString(ep, "laststep");
	return solution;
}

void SliceMain::findQuadrangleNext(){

	A1.x = -(bParallel[0] - bParallel[2]) / (k[0] - k[2]);
	A1.y = k[0] * A1.x + bParallel[0];
	B1.x = -(bParallel[0] - bParallel[3]) / (k[0] - k[3]);
	B1.y = k[0] * B1.x + bParallel[0];
	C1.x = -(bParallel[1] - bParallel[3]) / (k[1] - k[3]);
	C1.y = k[1] * C1.x + bParallel[1];
	D1.x = -(bParallel[1] - bParallel[2]) / (k[1] - k[2]);
	D1.y = k[1] * D1.x + bParallel[1];

	xSlicePer[0] = A1.x;
	ySlicePer[0] = A1.y;
	xSlicePer[1] = B1.x;
	ySlicePer[1] = B1.y;
	xSlicePer[2] = C1.x;
	ySlicePer[2] = C1.y;
	xSlicePer[3] = D1.x;
	ySlicePer[3] = D1.y;
}

/**
	Функция, которая занимается вписыванием среза в
                                        четырёхугольник
*/
void SliceMain::inscribeSlice(){

	 IplImage* ImageBefore = cvCloneImage(this->Image);
	 showStartImage(ImageBefore);
	 //cvWaitKey(0);
	// ищем, во сколько раз надо изменить длину и ширину,
	//        найдя отношение сторон двух прямоугольников
	 double wid = calculateNewSize(1, 2);
	 double hei = calculateNewSize(0, 1);
```

```
      IplImage* littleSlice = cvCreateImage(cvSize(this->dst-
>width/(wid), this->dst->height /(hei)), this->dst->depth, this->dst-
>nChannels);

      resizeImageToRectangle(littleSlice);
      cvSaveImage("littleSlice.png", littleSlice);

      // ищем контур для обновленного среза
      findContour();
      drawContour(littleSlice);
      // вот этот четырехугольник потом используется как канонный!
      //refreshAndDrawReqt(littleSlice);


      // находим b_ чтоб ограничить параллельными прямыми наше
изображение из атласа
      findBkoeffForParallel();
      findQuadrangleNext();
      drawQuadrang(xSlicePer, ySlicePer, littleSlice);

      showProcessImage("1", littleSlice);
      // ищем линейное проебразование для каждой точки изображения,
      //       которое переведёт наше изображение в четырехугольник
      double *x0_y0_k_h = new double[4];
      x0_y0_k_h = findLinearTransformation();
}
```

ImageProcessing.m

```
 I = imread(fileName);
 J = imadjust(I, [0 75]/255, [ ], 1);
 bw = im2bw(J);
 BW_filled = imfill(bw,'holes');

 boundaries = bwboundaries(BW_filled);
 b = [1;1];
 for loopmy=1:size(boundaries)
    bb = boundaries{loopmy};
    if(size(b,1)<size(bb,1))
        b = bb;
    end
 end
binaryImage = BW_filled;
```

```
grayImage = 255 * uint8(binaryImage);
RGB = cat(3, grayImage, grayImage, grayImage);
RGB = padarray(RGB,[3 3],'both');
figure (1)
imshow(RGB,'Border','tight');
hold on
plot(b(:,2)+3,b(:,1)+3,'y','LineWidth',2);
r = 150; % pixels per inch
set(gcf, 'PaperUnits', 'inches', 'PaperPosition', [0 0 size(RGB,2)
size(RGB,1)]/r);
print(gcf,'-dpng',sprintf('-r%d',r), newName);
close(figure (1));
clear all;
```

findSolutionMLS.m

```
function x0_y0_k_h = findSolutionMLS(bParalKoeff,kKoeff,bKoeff)
% находим углы из коэффициентов k
    fi = [rad2deg(atan(kKoeff(1))) rad2deg(atan(kKoeff(2)))
rad2deg(atan(kKoeff(3))) rad2deg(atan(kKoeff(4)))];

    B1 = bKoeff(1)*cosd(fi(1));
    B2 = bKoeff(2)*cosd(fi(2));
    B3 = bKoeff(3)*cosd(fi(3));
    B4 = bKoeff(4)*cosd(fi(4));

    B_1 = bParalKoeff(1)*cosd(fi(1));
    B_2 = bParalKoeff(2)*cosd(fi(2));
    B_3 = bParalKoeff(3)*cosd(fi(3));
    B_4 = bParalKoeff(4)*cosd(fi(4));

    A1 = B_2*sind(fi(4)-fi(3)) + B_3*sind(fi(2)-fi(4)) +
B_4*sind(fi(3)-fi(2));
    A2 = B_1*sind(fi(3)-fi(4)) + B_3*sind(fi(4)-fi(1)) +
B_4*sind(fi(1)-fi(3));
    A3 = B_1*sind(fi(4)-fi(2)) + B_2*sind(fi(1)-fi(4)) +
B_4*sind(fi(2)-fi(1));
    A4 = A1*B1 + A2*B2 + A3*B3 - B4;

    s2 = sign(A1*A2);
    s3 = sign(A2*A3);
    s4 = sign(A3*A4);

    A = [ - sind(fi(1)) cosd(fi(1)) B_1 1;
```

```
          - sind(fi(2)) cosd(fi(2)) B_2 s2;
          - sind(fi(3)) cosd(fi(3)) B_3 s3;
          - sind(fi(4)) cosd(fi(4)) B_4 s4];

    b=[ B1; B2; B3; B4];
    x0_y0_k_h = (A' * A) \ A' * b;
end
```

Код для решения второй задачи.

modelob.m

```
%% Список всех файлов в текущей директории
cd DataInsideYou;
f=dir('*.png');
files={f.name};
allX = [];
allY = [];
Pointscell = {};

%% считываем срезы, накладываем альфа канал поправленный,
%  считываем их снова + находим границы срезов
for k=1:numel(files)
    [img,map,alpha] = imread(files{k});
    A1 = ones(size(img));
    A2 = ones(size(img));
    A3 = ones(size(img));
    A1(img(:,:,1)==0)=0;
    A2(img(:,:,2)==0)=0;
    A3(img(:,:,3)==0)=0;

    A = A1+A2+A3;
    A= A(:,:,1);

    imwrite(img,'test.png','alpha',A);
    [img,map,alphaChannel{k}] = imread('test.png');
    Im{k}=img;
    J = imadjust(img, [0 75]/255, [ ], 1);
    bw = im2bw(J);
    bw = imfill(bw,'holes');

    b = bwboundaries(bw);
    boundar = b{1};
    allX = [allX, boundar(:,1)'];
    allY = [allY, boundar(:,2)'];
```

```
    Pointscell{1,k} = [boundar(:,1)'; boundar(:,2)'];
end
delete 'test.png';
%% считываем данные из файлов
% сдвиг пока поставлен в ноль
f=dir('*.txt');
files_params={f.name};
for k=1:numel(files_params)
    fid = fopen(files_params{k});
    str = fgetl(fid);
    sep_str = strsplit(str);
    vecX(k) = str2num(sep_str{1,1});
    %vecX(k) = 0;
    vecY(k) = str2num(sep_str{1,2});
    %vecY(k)=0;
    vecLevel(k) = str2num(sep_str{1,3});
    fclose(fid);
end

%% сортируем данные в порядке возрастания
for l1=1:numel(files_params)-1
    for l2=l1+1:numel(files_params)
        if(vecLevel(l1)>vecLevel(l2))
            tmp = vecLevel(l1);
            vecLevel(l1) = vecLevel(l2);
            vecLevel(l2) = tmp;
            tmp = vecX(l1);
            vecX(l1) = vecX(l2);
            vecX(l2) = tmp;
            tmp = vecY(l1);
            vecY(l1) = vecX(l2);
            vecY(l2) = tmp;
            tmp = Im{1,l1};
            Im{1,l1} = Im{1,l2};
            Im{1,l2} = tmp;
            tmp = alphaChannel{1,l1};
            alphaChannel{1,l1} = alphaChannel{1,l2};
            alphaChannel{1,l2}= tmp;


            tmp = Pointscell{1,l1};
            Pointscell{1,l1} = Pointscell{1,l2};
```

```
            Pointscell{1,l2}= tmp;
        end
    end
end

hobj=findobj('Type','axes','Tag','axes1');
axes(hobj)
hobj=rotate3d;
set(hobj,'Enable','on');
DT = {};
%% строим 3D модель
for k=1:size(vecLevel,2)-1
    Xx = [];
    Yy = [];
    Zz = [];
    tm = [];
    Points = Pointscell{1,k};
    Xx(1:size(Points,2)) = Points(2,:)-vecY(k);
    Yy(1:size(Points,2)) = Points(1,:)-vecX(k);
    tm(1:size(Points,2)) = vecLevel(k);
    Zz(1:size(Points,2)) = tm;
    tm = [];
    Points = Pointscell{1,k+1};
    Xx(end+1:end+size(Points,2)) = Points(2,:)-vecY(k+1);
    Yy(end+1:end+size(Points,2)) = Points(1,:)-vecX(k+1);
    tm(1:size(Points,2)) = vecLevel(k+1);
    Zz(end+1:end+size(Points,2)) = tm;

    Dtri = [Xx' Yy' Zz'];
    DT{k} = DelaunayTri(Xx',Yy',Zz');
    hullFacets = convexHull(DT{k});
    hh =
trisurf(hullFacets,DT{k}.X(:,1),DT{k}.X(:,2),DT{k}.X(:,3),'FaceColor',[
1,.75,.65],'FaceAlpha',.5);
    set(hh,'EdgeColor','none');
    hold on;
end
%title('Tetrahedral mesh');
view(3)
axis tight
lightangle(45,30); lighting phong
```

```
%% отображаем срезы
for l=1:numel(files)
    % the image data you want to show as a plane.
    planeimg = Im{1,l};

    % desired z position of the image plane.
    imgzposition = vecLevel(l);
    %происходит наложение модели и рисунков
    % поэтому маленький лайфхак при отображении
    if l==1
        imgzposition = imgzposition - 0.1;
    end
    if l==numel(files)
        imgzposition = imgzposition + 0.1;
    end
    hold on;
    %% если отображаем срезы целиком
    hobj = surf([-vecY(l) -vecY(l)+size(planeimg,2)],...
        [-vecX(l) -vecX(l)+size(planeimg,1)],...
        repmat(imgzposition, [2 2]),...
        planeimg,...
        'FaceColor','texturemap',                              ...
        'EdgeColor','none',                                    ...
        'FaceAlpha','texture',                                ...
        'AlphaData', alphaChannel{l});
    end
hold off
%% set a colormap for the figure.
colormap(jet);
% set the view angle.
view(30,30);
grid on;
grid minor;
% labels
xlabel('x');
ylabel('y');
zlabel('z');
CountourPoint = cell(1,vecLevel(end));
```

find_slice_between.m

```
if exist('hPlotData','var') ~= 0
```

```
    delete(hPlotData);
    delete(hhPlotData);
end

h=findobj('Type','axes','Tag','axes2');
lev=findobj('Tag','edit1');
levelint = str2num(lev.String);
axes(h)
axis tight

if ~isempty(find(vecLevel==levelint))
    ls = find(vecLevel==levelint);
    hPlotData = imagesc([-vecY(ls) -vecY(ls)+size(Im{1,ls},2)], [-
vecX(ls) -vecX(ls)+size(Im{1,ls},1)], Im{1,ls});
    set( hPlotData,'AlphaData', alphaChannel{ls});
    hold on;
    hhPlotData = plot([0 0.01],[0 0.01],'w');
    grid on;
    grid minor;
else
    if(isempty(CountourPoint{vecLevel(end)-1}))
        vecXma = max(vecX);
        vecXmi = min(vecX);
        vecYma = max(vecY);
        vecYmi = min(vecY);

        Xmax = max(allX)-vecXma+1500;
        Xmin = min(allX)-vecXmi-1500;
        Ymax = max(allY)-vecYma+1500;
        Ymin = min(allY)-vecYmi-1500;

        Zmax = max(vecLevel);
        Zmin = min(vecLevel);
        u=linspace(Xmin,Xmax,500);
        v=linspace(Ymin,Ymax,500);
        [x,y]=meshgrid(v,u);
        for k=1:size(vecLevel,2)-1
            F =
TriScatteredInterp(DT{k}.X(:,1),DT{k}.X(:,2),DT{k}.X(:,3));
            qz = F(x, y);
            hold on;
```

```
            %contour3(x, y, qz, vecLevel(k+1)-vecLevel(k)-
1,'visible','off');
            [C,ha] = contour(x, y, qz, vecLevel(k+1)-vecLevel(k)-
1,'visible','off');
            [xr,yr,zr] = C2xyz(C);
            Xrr = [];
            Yrr = [];
            parfor lev = vecLevel(k)+1:vecLevel(k+1)-1
                jj= find(abs(zr-lev)<0.1);
                for n = 1:size(jj,2) % only loop through the z =
values.
                    if n==1
                        Xrr(1:size(xr{jj(n)},2)) = xr{jj(n)};
                        Yrr(1:size(yr{jj(n)},2)) = yr{jj(n)};
                    else
                        Xrr(end+1:end+size(xr{jj(n)},2)) = xr{jj(n)};
                        Yrr(end+1:end+size(yr{jj(n)},2)) = yr{jj(n)};
                    end
                end

                CountourPoint{lev}= [Xrr ; Yrr];

                if (vecLevel(k+1) - vecLevel(k) < 100)
                    sliceFirst = Im{1,k};
                    sliceSecond = Im{1,k+1};
                    param_one =
min(size(sliceFirst,1),size(sliceSecond,1));
                    param_two =
min(size(sliceFirst,2),size(sliceSecond,2));

                    sliceFirst  = imresize(sliceFirst,[param_one
param_two]);
                    sliceSecond  = imresize(sliceSecond,[param_one
param_two]);
                    Ima =
imfuse(sliceFirst,sliceSecond,'blend','Scaling','joint')
                    CountourImage{lev} = Ima;
                end

                Xrr = [];
                Yrr = [];
            end
```

```
            disp('Loading...')
        end
    end


    for ff=levelint:levelint
        Countour = CountourPoint{ff};
        k = boundary(Countour(1,:)',Countour(2,:)');
        minX = min(Countour(1,:));
        minY  = min(Countour(2,:));
        try
            sliceIm = CountourImage{1,ff};
            A1 = ones(size(sliceIm));
            A2 = ones(size(sliceIm));
            A3 = ones(size(sliceIm));
            A1(sliceIm(:,:,1)==0)=0;
            A2(sliceIm(:,:,2)==0)=0;
            A3(sliceIm(:,:,3)==0)=0;

            A = A1+A2+A3;
            A= A(:,:,1);

            imwrite(sliceIm,'test.png','alpha',A);
            [sliceIm,map,alphaChannelslice] = imread('test.png');
            hPlotData = imagesc([-abs(minX) -
abs(minX)+size(sliceIm,2)], [-abs(minY) -abs(minY)+size(sliceIm,1)],
sliceIm);
            set( hPlotData,'AlphaData', alphaChannelslice);
            hold on;
        catch me
            fprintf('Image cannot be created because distance between
slices is enormous!\n');
        end
        hhPlotData = plot(Countour(1,k),Countour(2,k),'r');
        grid on;
        grid minor;
    end
end
```