СОДЕРЖАНИЕ

Введение ... 8
1 Анализ предметной области ... 9
1.1 Теоретические основы построения панорам ... 9
1.2 Склейка изображений ... 10
1.2.1 Модели трансформаций ... 11
1.2.2 Выравнивание сегментов ... 13
1.2.3 Метод RANSAC ... 15
1.2.4 Глобальная регистрация ... 16
1.2.5 Ретуширование ... 17
1.3 Методы нахождения и описания особых точек ... 18
1.3.1 Алгоритм SIFT ... 18
1.3.2 Алгоритм SURF ... 19
1.3.3 Детекторы углов ... 20
1.4 Способы отображения панорам ... 20
1.5 Примеры реализации задачи синтеза панорам ... 22
1.6 Постановка цели и задач дипломного проекта ... 25
2 Модели и спецификация требований к программному средству ... 26
2.1 Функциональная модель программного средства ... 26
2.2 Спецификация требований к программному средству ... 27
2.2.1 Загрузка и сохранение изображений ... 27
2.2.2 Процесс генерации панорамы ... 27
2.2.3 Редактирование изображений ... 27
2.2.4 Пользовательский интерфейс программного средства ... 27
2.2.5 Совместимость программного средства ... 28
2.3 Модель предметной области ... 28
2.4 Математическая модель программного средства ... 29
2.4.1 Модель преобразования координат ... 29
2.4.2 Прямая подгонка сегментов ... 29
2.4.3 Поиск особых точек ... 30
2.4.4 Построение дескрипторов ... 33
2.4.5 Соотнесение особых точек ... 34
2.4.6 Корректировка ... 36
3 Проектирование программного средства ... 38
3.1 Архитектура программного средства ... 38
3.1.1 Выбор типа программного средства ... 38
3.1.2 Программная архитектура ... 38
3.2 Проектирование пользовательского интерфейса ... 41
3.3 Алгоритмы программного средства ... 43
3.3.1 Схема работы программы ... 43
3.3.2 Алгоритм генерации панорамы ... 44
3.3.3 Алгоритм сравнения сегментов панорамы ... 47
4 Конструирование программного средства ... 51

4.1 Выбор средств разработки 51
4.2 Описание компонента синтеза панорамы 52
4.2.1 Класс Stitcher 53
4.2.2 Класс Segment 55
4.2.3 Класс PanoramaImages 56
4.2.4 Класс PanoramaRelations 56
4.2.5 Класс PanoramaTransformations 57
4.2.6 Интерфейс IImagesRelation 57
4.2.7 Интерфейс ITransformation 58
4.2.8 Интерфейс IAnalyzer 58
4.2.9 Интерфейс IBuilder 58
4.2.10 Интерфейс IPresenter 58
4.2.11 Интерфейс IRelationControl 59
4.2.12 Класс DefaultFactory 59
5 Тестирование программного средства 61
5.1 Модульное тестирование программного средства 61
5.2 Интеграционное тестирование программного средства 61
6 Руководство пользователя программного средства 68
7 Технико-экономическое обоснование эффективности разработки программного средства синтеза панорамных изображений 72
7.1 Функции, назначение и потенциальные пользователи ПС 72
7.2 Расчет затрат на разработку и реализацию ПС 72
7.3 Оценка эффекта от использования ПС 75
7.4 Расчет показателей эффективности инвестиций в разработку ПС 75
8 Эргономическая экспертиза программного средства синтеза панорамных изображений 78
8.1 Сущность информационной совместимости 78
8.2 Характеристика трудового процесса пользователя при работе с программным средством. Проектирование информационной архитектуры 79
8.3 Оценка эргономической эффективности человеко-машинного взаимодействия с помощью модели GOMS 81
8.3.1 Описание модели GOMS 81
8.3.2 Оценка задачи генерации панорамы 82
Заключение 86
Список использованных источников 88
Приложение А Исходный код библиотеки синтеза панорам и ее расширений 89
Приложение В Исходный код основных классов приложения 109

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие понятия.

Библиотека – сборник подпрограмм и объектов, используемых для разработки программного обеспечения.

Интерфейс – способ взаимодействия двух систем (человека, устройств, программ и др.).

Приложение (прикладная программа) – программа, предназначенная для выполнения определенных задач и рассчитанная на непосредственное взаимодействие с пользователем.

Программа – последовательность инструкций, предназначенных для исполнения устройством управления вычислительной машины.

Программное обеспечение – полный набор программ, процедур, правил и связанной с ними документации системы обработки информации.

Программное средство – ограниченная часть программного обеспечения системы обработки информации, имеющая определенное функциональное назначение.

Сегмент (панорамы) – изображение, используемое для синтеза панорамы.

Синтез – процедура соединения или объединения ранее разрозненных объектов в одно целое.

Склейка изображений (image stitching) – синтез панорамы из набора изображений.

Тест-кейс (тестовый случай) – набор условий, позволяющих определить, работает ли программа согласно требованиям к ней.

Фреймворк – программная платформа, определяющая структуру программной системы.

ОС – операционная система.

ПС – программное средство.

ВВЕДЕНИЕ

Панорама (от греч. πᾶν «все» и ὅραμα «видение») – это любое представление пространства с широким углом и большой глубиной обзора, будь то рисунок, картина, фотография, видео, карта или трехмерная модель. Понятие впервые стало использоваться в живописи, для описания особого типа картин. Сегодня панорамы встречаются повсеместно как средства наглядного отображения больших пространств со множеством деталей. Существует особый вид панорам, охватывающих все возможные углы обзора из определенной точки, т.е. позволяющих создавать виртуальные трехмерные сцены. В том или ином виде, панорамы используются для научных, развлекательных и рекламных целей.

Подавляющее большинство панорам имеют вид фотографий. Современные объективы фотоаппаратов жестко ограничены в плане углов обзора, и часто неспособны охватить всю требуемую картину. Для решения этой проблемы при съемке получается набор фотографий с различным углом поворота камеры, затем при пост-обработке они объединяются в панораму.

Процесс пост-обработки существенно варьируется по сложности. При использовании специальной аппаратуры и техники съемки, генерация панорамы может происходить «на лету», с помощью самого фотографирующего устройства. Такой метод применяется повсеместно, однако имеет ряд существенных недостатков:

- дороговизна аппаратуры для качественной съемки;
- практическое отсутствие настроек процесса генерации, что в некоторых условиях может приводить к неизбежности дефектов в получаемой панораме.

Более сложная пост-обработка происходит вне фотоаппарата, в удобной и настраиваемой среде. Такой метод ослабляет зависимость качества получаемой панорамы от исходных фотографий, позволяя использовать в этой роли произвольные изображения - как полученные непосредственно для этой цели, так и доступные из других источников. Это избавляет от необходимости в специальном процессе съемки, делая создание панорам легкодоступным процессом. Такой подход предоставляет большой простор для творчества, однако ставит новую проблему: ручная подгонка и редактирование изображений может быть весьма трудозатратным процессом. Он поддается автоматизации, но любое ограничение контроля пользователя над процессом увеличивает вероятность получения неудовлетворительного результата. Необходим компромисс, и используемые средства автоматизации должны позволять изменять баланс для конкретной ситуации. Создание таких средства является нетривиальной задачей, использующей знания нескольких научных дисциплин.

Назначение разработки данного программного средства – развитие научной области путем решения актуальных проблем и новой реализации существующих решений.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Теоретические основы построения панорам

Автоматизация построения панорам относится к таким научным областям, как компьютерное зрение и обработка изображений. Компьютерное зрение – теория и технология создания машин, которые могут производить обнаружение, слежение и классификацию объектов [1]. Она относится к теории создания искусственных систем, которые получают информацию из изображений. Обработка изображений – форма обработки информации, для которой входные данные представлены двухмерным изображением. Хотя по определению эта область и является производной от области компьютерного зрения, вторая часто упоминается как отдельная, сосредотачивающаяся на обработке трехмерных сцен (работа же с панорамами подразумевает оперирование терминами как двух-, так и трехмерного пространства).

Всю область можно охарактеризовать как молодую, разнообразную и динамично развивающуюся. Интенсивное её изучение началось лишь в конце 1970-х гг., когда компьютеры смогли управлять обработкой больших наборов данных, какими являются изображения. И сейчас нет стандартной формулировки этой области, а многие методы и приложения всё ещё находятся на стадии фундаментальных исследований. В последнее время наблюдается повышение активности изучения области, ввиду всё большего применения её методов в коммерческих продуктах. Важнейшей прикладной областью, где используется компьютерное зрение, является медицина. Компьютерная обработка позволяет отобразить данные, получаемые с помощью микроскопии, рентгенографии и томографии, в наиболее оптимальном для исследования экспертами виде. В промышленности

Цель компьютерного зрения – имитация биологического зрения, чрезвычайно сложного и многоаспектного явления. Так, процесс человеческого видения зависит как от биологических особенностей субъекта, так и от его психики, всего накопленного опыта и инстинктов. Точное воспроизведение подобных явлений с помощью искусственных артефактов практически невозможно. В то же время, попытки автоматизации зрительного процесса уже дали ощутимые плоды, что говорит о перспективности дальнейших исследований в этой области.

Существует два способа хранения цифровых изображений: векторный и растровый. Векторная графика представляет набор элементарных геометрических объектов, иным словом, описывается с помощью математических функций. Подобное представление обычно подразумевает предрасположенность изображений к семантическому (содержательному, смысловому) анализу. С другой стороны, растровая графика оперирует лишь упорядоченным множеством атомарных единиц графической информации – пикселей. Так как понятие изображения подразумевает двухмерность, пиксели объединены в матрицу размерности x*y. Каждый пиксель, помимо

уникальных координат в матрице, характеризуется также значениями таких свойств, как интенсивность и цвет.

Любое изображение, полученное из реального мира, имеет изначально растровый характер. Биологическое зрение связано с хорошо развитой нейронной сетью смотрящего, которая с высокой скоростью и точностью объединяет поступающую с сетчатки глаза информацию в узнаваемые образы. Вычислительная техника по своей сложности еще очень далека от органического мозга, поэтому компьютерное восприятие столь больших объемов данных имеет весьма ограниченный характер.

Однако, одно из преимуществ современной вычислительной техники над человеческих мозгом – скорость вычислений. Статистический анализ изображений используется повсеместно для имитации и замены семантических операций над ними.

К числу таких операций относится воссоздание изображений по их сегментам. Тривиальная для человека задача соотнесения частей целого может решаться автоматически с помощью сложного математического механизма. Статистический аспект механизма придает его работе некоторую долю погрешности, однако постоянное развитие технологий и расширение баз данных постепенно сводит эту погрешность к минимуму.

1.2 Склейка изображений

Задача склейки нескольких изображений в общее их представление – одна из старейших в области компьютерного зрения [2]. Уже несколько десятилетий специальные алгоритмы используются для составления карт и картин высокого разрешения (наиболее известное их применение – обработка спутниковых и телескопных фотографий). За такой срок было разработано множество различных подходов. К примеру, изначально изображения соотносились друг с другом по их пиксельному сравнению и минимизации различий. Сегменты вручную выстраивались в необходимом порядке, а вычислительные машины подгоняли их один под другой (см. рисунок 1.1). Позднее был разработан более быстрый подход, оперирующий уже не каждым отдельным пикселем, но множеством локальным особенностей изображения. Появилась возможность генерации панорам по неупорядоченному набору изображений.

Определение процесса синтеза панорамы состоит в решении следующих задач [3]:

1) Описание отношений координат пикселей одного изображения по отношению к другому;

2) Нахождение и оценка корректности вариантов относительного расположения сегментов;

3) Совместное позиционирование всех сегментов;

4) Выбор поверхности или холста, на котором будет располагаться скомпонованное сегменты;

5) Сочетание сегментов в виде цельной панорамы.

Рисунок 1.1 – Склейка сегментов в панораму

Общий алгоритм генерации или синтеза панорам на основании множества сегментов состоит из трех этапов, каждый из которых имеет специфические проблемы и методы их решения:

1) Регистрация.

2) Калибровка.

3) Сочетание.

Регистрация подразумевает анализ исходных сегментов панорамы. Этот этап пропускается при использовании метода прямой подгонки. Калибровка – соотнесение сегментов друг с другом. Сочетание – объединение сегментов в одно изображение и ликвидация признаков склейки.

1.2.1 Модели трансформаций

Одной из фундаментальных операций при склейке изображений является их пространственное преобразование. Исходные сегменты представляют собой проекции некоторой сцены на плоскость объектива в разных позициях, углах наклона и поворота. Сегменты не хранят подобную информацию, однако её наличие является необходимым условием для воссоздания сцены в виде панорамы.

В этом контексте задачу объединения можно выразить как нахождение такого преобразования для каждого сегмента, которое позволит корректно разместить его на панораме. Для практического оперирования понятием преобразования, необходимо определить используемую математическую

модель (*motion model*). В общем случае она отображается как матрица параметров размерностью 3 x 3 и используется в составе оператора, определяющего однозначное соответствие пары точек в двухмерном пространстве координат. Расчеты с использованием всех восьми параметров (девятый всегда принимается равным единице) могут требовать значительных вычислительных затрат, поэтому зачастую матрицу ограничивают, приспосабливая для конкретных типов преобразований [4].

Двухмерные перемещения используются при обработке видеоданных, когда разница между положением объектов на двух соседних кадрах невелика. Эта модель оперирует всего двумя параметрами.

Двухмерное евклидово преобразование – трехпараметровая модель, добавляющая к перемещению эффект вращения. Используется при работе с плоскими вращениями, например, при частичном сканировании большого изображения на малом сканере.

Масштабируемое вращение, или преобразование подобия, прибавляет четвертый параметр. Эффективно при склейке сегментов, полученных с помощью с большим фокусным расстоянием. Важно отметить, что данная модель сохраняет углы на изображении неизменными.

Аффинное преобразование - шестипараметровая модель, использующая верхние два ряда универсальной матрицы. Модель не влияет на свойство параллельности объектов.

Гомографическое, или перспективное, преобразование – самое общее из двумерных, определяющееся восемью параметрами. Оно сохраняет прямоту линий на изображении и считается достаточным для эффективной обработки фотографий. Однако, определение параметров гомоморфной модели заключается в решении системы уравнений с восемью неизвестными, что может пагубно отразиться на скорости работы вычислительной системы. Поэтому, в зависимости от конкретной ситуации рекомендуется использовать модель с наименьшим количеством параметров.

Все двухмерные преобразования координат наглядно продемонстрированы на рисунке 1.2.

Существую и более сложные модели – такие, как преобразование в трехмерной плоскости, использующее матрицу размерностью 4 на 4. Подобные модели не принято использовать при склейке изображений из-за чрезмерной сложности вычислений, поэтому гомоморфная модель используется как универсальная, но с рядом допущений:

- исходные фотографии сделаны неперемещающейся камерой с вращением по фиксированной оси;
- фотографируемая сцена считается плоской, что приемлемо при достаточном удалении фотокамеры от объектов сцены.

За определением используемой модели преобразований следует выбор метода нахождения параметров этой модели. Процедура нахождения называется выравниванием, и для неё существуют два принципиально разных подхода.

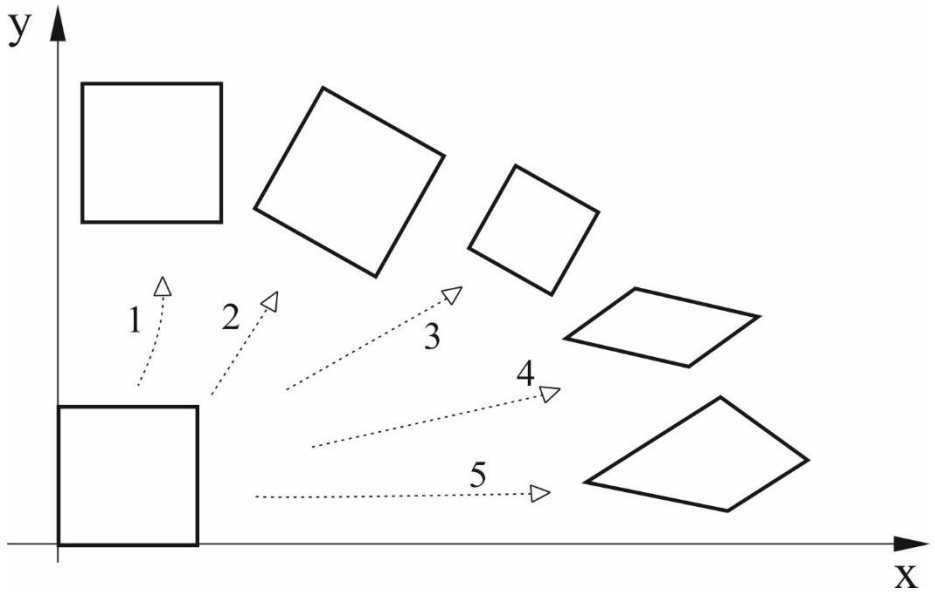

Рисунок 1.2 – Двухмерные преобразования:
1 – перемещение, 2 – евклидово, 3 – подобие, 4 – аффинное, 5 – перспективное

1.2.2 Выравнивание сегментов

Исходные данные: два изображения (сегмента), для которых необходимо определить преобразование, объединяющее их в одно изображение. Прямое выравнивание заключается в последовательном применении различных преобразований и выбора оптимального, по некоторому критерию (метрика ошибок). Простейший пример – полный поиск, применяющий все возможные варианты преобразования. Очевидным недостатком такой реализации является большое время работы. Способы оптимизации:

- иерархический поиск на основе пирамид изображений;
- использование преобразования Фурье для ускорения вычислений;
- инкрементный поиск с разложением изображения в ряд Тейлора.

Прямое выравнивание основано на вычислениях в пиксельном масштабе, когда каждый пиксель является компонентом итерирования. В противовес этому низкоуровневому подходу существуют более абстрактные, самый известный из которых – использование *локальных особенностей*. Он заключается в оперировании не всем множеством пикселей, а лишь небольшим набором элементов, относящихся к объектам на изображении. Если сегменты относятся к одной сцене, есть вероятность наличия одного объекта на нескольких из них, и каждый сегмент будет иметь характерную этого для объекта особенность. Также, если у объекта имеется несколько особенностей, их относительное расположение не должно существенно меняться от сегмента к сегменту. Таким образом, эти особенности относятся в больше степени к фотографируемой сцене, чем к фотографиям (см. рисунок

1.3). Различие между представлением одной особенности на разных сегментах выражается в форме оператора преобразования, который затем может быть применен к каждому пикселю сегмента при его вставке в панораму.

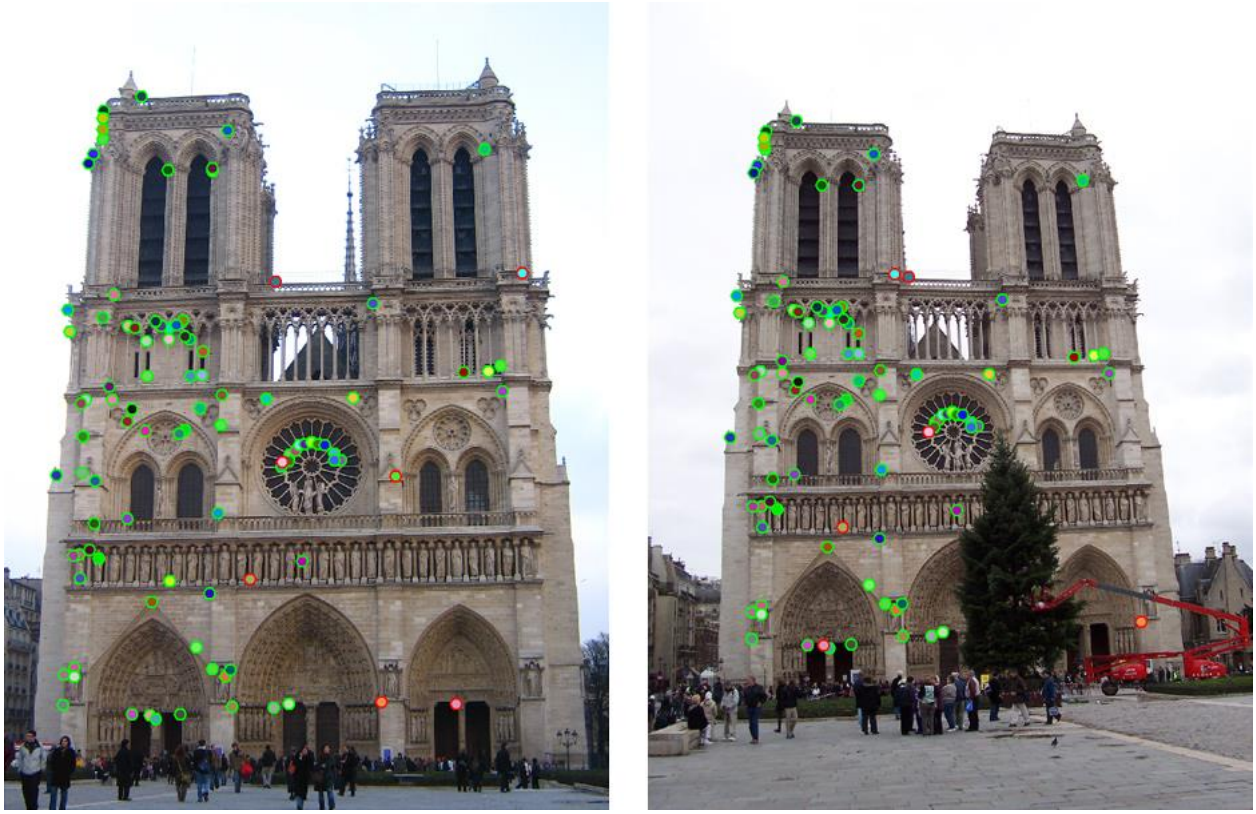

Рисунок 1.3 – Совпадение ключевых точек на различных изображениях

Миколайчик и Туителаарс [5] определили набор необходимых свойств, которыми должна обладать особая точка:

- повторяемость – особая точка находится в одном и том же месте в изображаемой сцене или объекте, несмотря на изменения точки обзора и освещенности;
- различаемость – окрестности особых точек должны иметь существенные отличия друг от друга, достаточные для сопоставления этих точек;
- локальность – особая точка должна занимать небольшую область изображения для минимизации геометрических и фотометрических искажений;
- количество – число и плотность особых точек должны быть достаточными для обнаружения малых объектов (варьируется в зависимости от области применения);
- точность – обнаруженные точки должны точно локализовываться, в исходном или ином масштабе;
- эффективность – время обнаружения особых точек должно быть в допустимых для приложения пределах.

Задача поиска локальных особенностей на изображении является нетривиальной. Необходимым свойством особенности является стойкость к преобразованиям. Стандартом сегодня является независимость, в той или иной мере, ко всем двумерным трансформациям вплоть до аффинного преобразования. Это обеспечивается подходящим дескриптором особенности.

Существует ряд комплексных методик для поиска и описания локальных особенностей. Наиболее популярные из них будут рассмотрены далее в подразделе 1.4.

Выравнивание изображений при помощи особых точек может быть дополнено прямым выравниванием на конечном этапе, когда преобразования сегментов вычислены, но содержат некоторую долю погрешности. Т.е. пиксельное выравнивание эффективно при уточнении значений преобразований на ограниченном интервале значений.

1.2.3 Метод RANSAC

После описания особенностей каждого сегмента выполняется процедура их сравнения, имеющая целью нахождение совпадений и вычисление соответствующих им преобразований. Идентичные дескрипторы локальной особенности на практике встречаются крайне редко, поэтому сравнение допускает определенную величину отклонения. Это порождает новую проблему - нахождение ложных совпадений. Для определения достоверных пар связанных сегментов используются статистические алгоритмы, к примеру, RANSAC.

RANSAC (Random Sample Consensus) – статистический метод оценки параметров модели на основе случайных выборок. Предложен Фишлером и Боллесом в 1981 г. Алгоритм предназначен для построения и оценки моделей по выборке данных, содержащей выбросы. Точные методы (типа метода наименьших квадратов) в таком случае приведут к выведению некорректной модели, так как она будет удовлетворять всем исходным данным, независимо от их корректности. Статистический метод основан на предположении, что количество некорректных данных значительно ниже, и они могут быть расценены как «выбросы». Так, RANSAC выбирает из исходных данные некоторые наборы ограниченной величины и строит модель («гипотезу») на их основе. Каждая полученная таким образом модель оценивается с применим всего набора исходных данных методом подсчета совпадений. Выбирается та гипотеза, которую подтверждает наибольшая доля данных (см. рисунок 1.4).

Метод RANSAC способен дать надежную оценку параметров модели, то есть с достаточно высокой точностью, даже если в исходном наборе данных присутствует значительное количество выбросов. Недостатком метода является отсутствие верхней границы времени, необходимого для вычисления параметров модели. Если использовать в качестве некоторой границы времени максимальное число итераций, полученное решение может быть неоптимальным, а также существует очень малая вероятность того, что ни

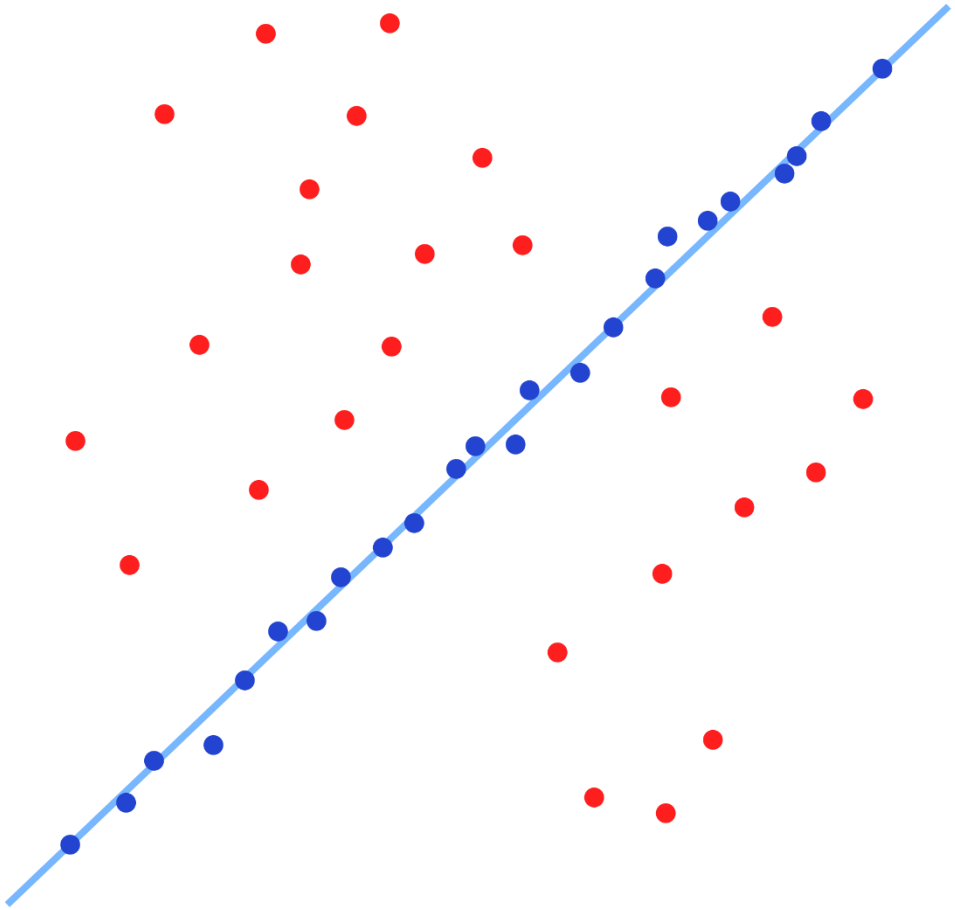

Рисунок 1.4 – Применение алгоритма RANSAC для построения прямой по набору точек, содержащему сдвиги и выбросы

одна модель не будет соответствовать исходным данным. Точная модель может быть определена с некоторой вероятностью, которая становится тем больше, чем большее количество итераций используется. Другим недостатков RANSAC является то, что для выполнения алгоритма необходимо задавать конкретные пороговые значения. Наконец, методом RANSAC можно определить только одну модель для определенного набора данных. Соответственно, если данные предполагают наличие нескольких наличий, может быть не найдена ни одна.

1.2.4 Глобальная регистрация

В предыдущих подразделах было определено, как вычисляются парные преобразования, объединяющие один сегмент с другим. Однако, цель процесса – единое изображение, совмещающее в себе множество сегментов. То есть, необходима глобальная система, в которой каждому сегменту будет

соответствовать определенное пространственное преобразование, безотносительно других сегментов.

Простейшее решение – последовательно добавлять сегменты на панораму, выравнивая каждый новый с ранее установленными и определяя их перекрывающиеся области. В случае, если строится панорама с углом обзора 360, аккумулированная ошибка неизбежно приведет к образованию пробела между двумя концами изображения. Более универсальная альтернатива такого метода – одновременное выравнивание всех сегментов с равномерным распределением общей ошибки. Такой способ называется методом наименьших квадратов и заключается в нахождении такой системы расположений всех сегментов, которая минимизирует суммарный квадрат разницы между текущим положением пикселей и предполагаемым. Предполагаемая позиция вычисляется преобразованиями координат точки между трехмерным и двумерным пространствами.

1.2.5 Ретуширование

Когда все сегменты расположены на панораме, необходима дополнительная обработка для гарантии целостного восприятия получаемого изображения. В общем случае после подгонки сегментов остаются видны дефекты, обусловленные качеством исходных изображений или спецификой применяемых алгоритмов (см. рисунок 1.5). Наиболее часто встречаемые дефекты: видимые швы (из-за разницы в яркости сегментов), нечеткость (из-за ошибок при регистрации сегментов) и «призраки» (движущиеся объекты на фотографиях). Четкое изображение достигается путем сокрытия лишних пикселей и размытия. Для выполнения этих действий разработано множество методик.

Медианный фильтр и сглаживание (*feathering*) решают проблемы шума и локальных дефектов. Одним из направлений развития этих фильтров является центрированное взвешивание пикселей. Так, в случае панорам многие проблемы решаются путем фокусировки сглаживающего фильтра на центр изображения (то есть, чем ближе пиксель находится к центру, тем большую роль он играет при корректировке интенсивности других пикселей). При перекрытии одной области несколькими сегментами их границы могут быть слишком заметны, возникает отчетливый «шов». Существует несколько методов решения проблемы. Один из таких – нахождение идеального шва, на котором границы двух сегментов имеют наименьшее различие. Такой шов прокладывается из одного конца стыка в другой, путем оптимального выбора каждого следующего шага (пикселя).

Для ликвидации «призраков» Uyttendaele разработал алгоритм [3], заключающийся в сканировании области перекрытий на наличие регионов с наибольшей разницей между сегментами и заменой его тем или иным вариантом (в зависимости от конкретной реализации алгоритма).

Рисунок 1.5 – Видимые дефекты при склейке панорам

Существует большое множество других алгоритмов, но практически все они предоставляют лишь частичное решение проблем. Дальнейшие улучшения требуют разработки специальных алгоритмов или оптимизации процесса склейки.

1.3 Методы нахождения и описания особых точек

1.3.1 Алгоритм SIFT

SIFT (Scale-Invariant Feature Transform) – алгоритм из области компьютерного зрения для детектирования и описания локальных особенностей изображения. Опубликован Д. Лоуи в 1999 г. Применяется для распознавания объектов, в робототехнике, трехмерном моделировании, распознавании жестов и отслеживании объектов на видео. SIFT способен точно идентифицировать объекты в сложных ситуациях, например, при перекрытии. Дескриптор SIFT полностью инвариантен к вращению и масштабированию, частично – к аффинным преобразованиям и смене освещения.

Особые точки определяются путем построения пирамиды масштабов для каждого изображения и нахождения локальных экстремумов интенсивности их пикселей (разница гауссиан) [6]. Затем они последовательно

уточняются по направлению градиента разницы гауссиан [7]. При помощи местных градиентов интенсивности определяется «направленность» особой точки. В конечном виде дескриптор имеет вид вектора.

При последовательном итерировании по исходным изображениям, для каждого определяется набор дескрипторов и сравнивается (по метрике евклидова расстояния) с ранее найденными наборами. Из совпадающих пар выделяется подмножество, характеризуемое одинаковым преобразованием позиции, масштаба и поворота. Нахождение таких подмножеств зачастую оптимизируется с помощью преобразования Хафа. Выделяются кластеры мощностью более трёх пар и так далее, до тех пор, пока не будет достигнута заданная достоверность совпадений.

Хотя метод SIFT подвержен статистическим ошибкам, точность детектирования относительно высока, что обусловило его распространенность.

1.3.2 Алгоритм SURF

SURF (Speeded-Up Robust Features) – алгоритм детектирования особенностей изображения, разработанный Г. Бэем в 2006 г. Применяется для распознавания объектов и трехмерной реконструкции. Метод частично основан на SIFT.

Упор делается на повсеместном использовании интегральных изображений (см. рисунок 1.6). Интегральным представлением изображения называется матрица той же размерностью, в которой каждая ячейка содержит сумму яркости соответствующего ей пикселя и значений всех ячеек слева и сверху от неё. Предварительный расчет такой матрицы позволяет значительно ускорить ряд вычислений при работе с изображением.

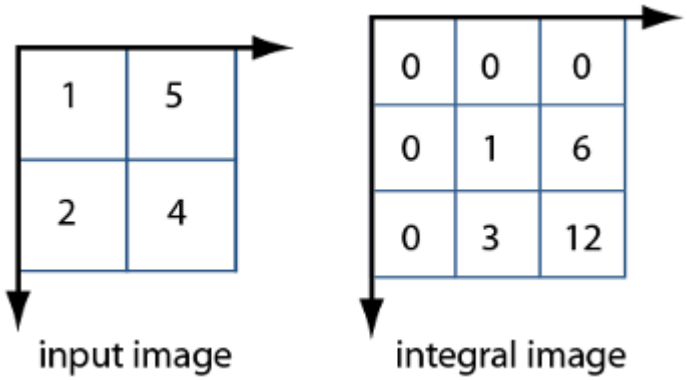

Рисунок 1.6 – Интегральное изображение

В SURF при детектировании локальных особенностей используется лапласиан гауссиана (LoG) и его гессиан. Но фильтр не используется в классическом виде, а заменяется приближенным линейным фильтром (box filter), легко вычисляемым при помощи интегрального изображения. Для

описания ориентаций используется отклик вейвлета Хаара, который также может быть рассчитан из интегральной матрицы.

В общем случае SURF работает в несколько раз быстрее, чем SIFT. Однако, его применение неэффективно при обработке фотографий из разных точек съемки, а также при различных уровнях освещения.

1.3.3 Детекторы углов

Широкое применение среди детекторов особых точек находят детекторы углов. Углы – особые точки, которые формируются из двух или более граней, определяющих границы между объектами или их частями. Иначе говоря, угол – точка, в окрестности которой интенсивность изменяется относительно центра в двух доминирующих градиентах. Градиент – векторная величина, показывающая направление наискорейшего возрастания функции (здесь – функции интенсивности пикселей на изображении). Так как изображение дискретно, вектор градиента определяется через частные производные по осям *x* и *y* через изменения интенсивностей соседних точек. Большинство детекторов рассматривают угловатость, зависящую от производной второго порядка, поэтому в общем случае чувствительность к шуму высока.

Детектор Харриса (1988) – наиболее эффективный детектор L-связных углов. Характеризуется анизотропией по всем направлениям, следовательно, и инвариантностью к поворотам [8]. Недостаток детектора – сильная чувствительность к шуму и изменениям масштаба.

FAST (Features from Accelerated Segment Test) – другой распространенный детектор углов, введенный Е. Ростеном и Т. Драммондом в 2005 г. Алгоритм рассматривает относительную интенсивность пикселей на фиксированной окружности. В 2008 г. алгоритм дополнен и назван FAST-ER (FAST with Enhanced Repeatability). Главным новшеством стало введение машинного обучения, сделавшего найденные особые точки повторяемыми (одинаково распознаваемыми на разных изображениях).

Детекторы углов определяют лишь алгоритмы поиска особых точек, но не способ их описания. Поэтому нередко встречаются комбинации типа детектор углов Харриса и SIFT, сочетающие быстроту детектирования углов с инвариантностью дескриптора.

1.4 Способы отображения панорам

Всё поле зрения может быть представлено как сферическая поверхность. Это свойство относится как к биологическому зрению, так и к фотоаппаратам. Из него следует, что при проецировании сферического изображение на плоскую поверхность неизбежны искажения. При малых углах обзора эти искажения могут быть практически незаметны, однако при больших – к примеру, на панорамах – они значительны (см. рисунок 1.7), что может вызывать неудобство восприятия таких изображений. Решением проблемы

является применение различных видов проецирования на двухмерную плоскость.

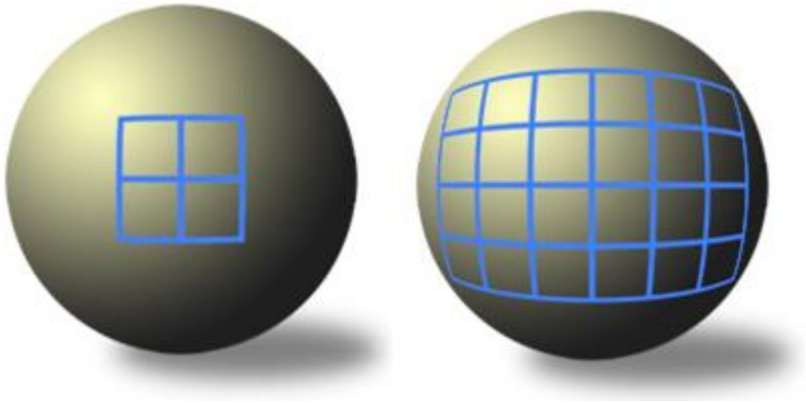

Рисунок 1.7 – Искажения при различных углах обзора

Метод равных прямоугольников заключается в нанесении на сферу координатной сетки в терминах параллелей и меридиан (взаимно ортогональных). Сеть проецируется на своё двухмерное представление, на котором все ячейки имеют равные размеры и форму. Эта проекция имеет горизонтальную длину, в два раза большую, чем её высота. Она характеризуется горизонтальным искажением, сильнее всего проявляющимся на полюсах сферы (верх и низ изображения).

Цилиндрическая проекция схожа с проекцией равных прямоугольников, однако она вносит дополнительное вертикальное искажение, нулевое на экваторе и бесконечно большое на полюсах (верхние и нижние ячейки сети имеют всего три грани). Это позволяет улучшить способность к сохранению размеров и пропорций объектов и изображении, однако вносит дополнительное искажение прямых линий и не позволяет оперировать большими вертикальными углами обзора. Цилиндрический метод взят за стандарт в панорамных камерах.

Метод «рыбий глаз» (fisheye) создает проекции, в которых расстояние от центра сетки пропорционально покрываемому углу обзора, что придает сходство с отражением на металлической сфере. Горизонтальный и вертикальный углы обзора ограничены 180. Характерная особенность проекции – усиление искажений по мере отдаления от центра изображения. Такой вид проекции редко используется для отображения панорам, однако чрезвычайно полезен при их создании, так как позволяет сократить количество необходимых сегментов буквально до нескольких штук.

Проекция Меркатора – компромисс между цилиндрической и равными прямоугольниками. Характеризуется небольшим вертикальным растяжением и способностью охватывать большие вертикальные углы. Недостаток, по сравнению с цилиндрической проекцией, - большая степень искажения прямых линий. Наиболее широко известное применение проекции Меркатора – плоская карта Земли.

Синусоидальная проекция имеет специфическую особенность – она сохраняет площади при переносе на двухмерную плоскость. Проекции широт являются идеально прямыми линиями. При этом вся имеющаяся информация о сферическом представлении сохраняемся, что позволяет делать обратное проецирование.

Стереографическая проекция схожа с «рыбьим глазом», но характеризуется лучшей сохранностью перспективных свойства. Это достигается путем растяжения объектов в сторону от точки перспективы.

Сравнительные параметры различных видов проекции сферической поверхности даны в таблице 1.1.

Таблица 1.1 – Сравнение различных видов проекции сферической поверхности на двухмерную плоскость

Тип проекции	Поле зрения		Сохранение прямых линий	
	Горизонтальное	Вертикальное	Горизонтальных	Вертикальных
Прямолинейная	<120	<120	Да	Да
Цилиндрическая	120-360	<120	Нет	Да
Проекция Меркатора	120-360	<150	Нет	Да
Прямые прямоугольники	120-360	120-180	Нет	Да
«Рыбий глаз»	<180	<180	Нет	Нет
Синусоидальная	360	180	Да	Нет

1.5 Примеры реализации задачи синтеза панорам

1.5.1 В популярном средстве для профессиональной обработки изображений Photoshop имеется специальный компонент для объединения изображений – Photomerge. Одним из его частных применений является создание панорам.

Достоинства:

- высокая степень интеграции с другими средствами Photoshop.

Недостатки:

- громоздкость (привязка к Photoshop);
- малый набор настроек;

- ограничения на входные изображения: неизменная яркость и наклон снимков, большая доля перекрытий.

1.5.2 Image Composite Editor (ICE) – программа для компиляции панорамных изображений, созданная Microsoft Research Computational Photography Group. На основе набора перекрывающихся фотографий одной сцены, сделанных из одной точки, приложение собирает панораму в высоком разрешении. В качестве основы могут также использоваться видеоданные. Возможен трехмерный обзор полученных панорам и их сохранение в популярных форматах (JPEG, TIFF, PSD/PSB).

Достоинства:

- бесплатность;
- малый размер и высокая скорость работы;
- минималистичный стилизованный интерфейс (см. рисунок 1.8);
- поддержка множества форматов для экспорта результата, популярных и специфичных;
- возможность преобразования панорам в трехмерные сцены.

Недостатки:

- относительно небольшой набор настроек;
- высокие требования к количеству и степени перекрываемости входных изображений.

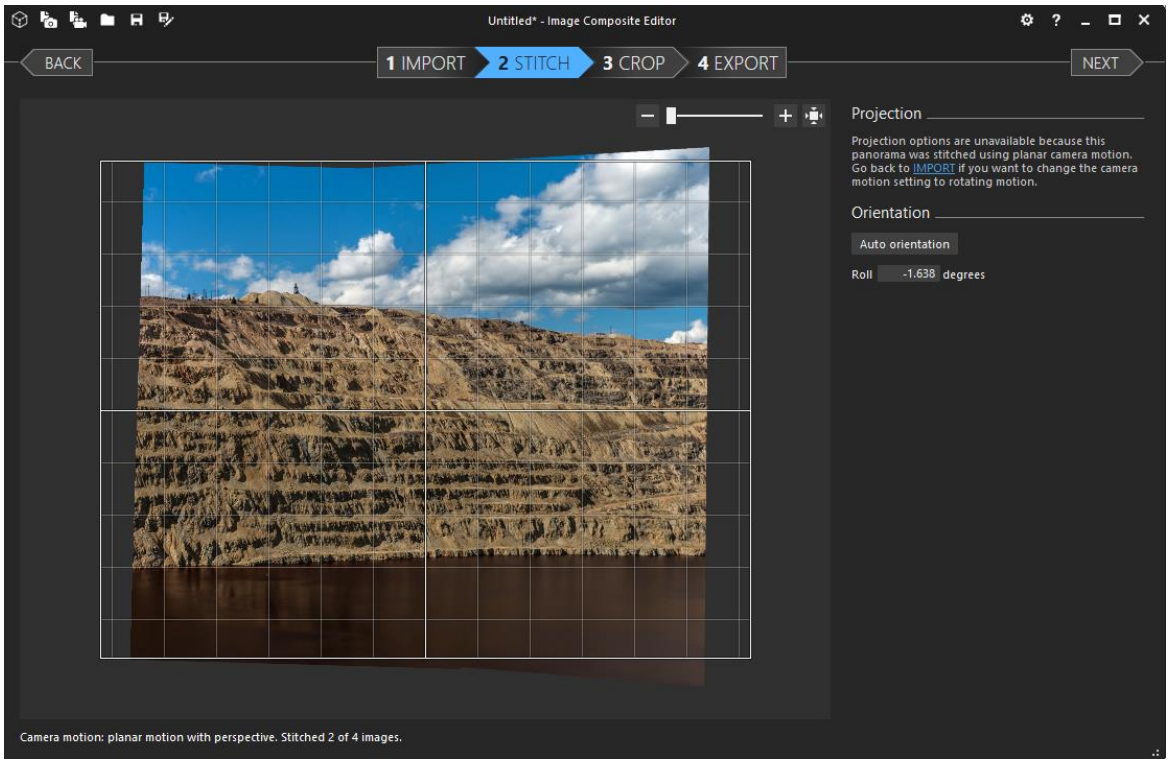

Рисунок 1.8 – Пример использования Image Composite Editor

1.5.3 PTGui (Graphical User Interface for Panorama Tools) – специализированное средство для склейки изображений. Изначально разработанное как дополнение к Panorama Tools, ныне является самостоятельным приложением. Предназначается для профессиональной деятельности, поэтому предлагает широкий набор опций и настроек. Среди специфических функций – возможность ручного задания особых точек и их соответствий на разных изображениях (см. рисунок 1.9).

Достоинства:

- высокая степень настраиваемости исходных и результирующих данных, а также процесса склейки;
- встроенный редактор изображений (входных и выходных).

Недостатки:

- ограничения на входные изображения: одинаковые размеры и ориентации, яркость, большая доля перекрытий;
- низкое удобство использования относительно других рассмотренных приложений;
- относительно низкое качество панорамы при настройках «по умолчанию».

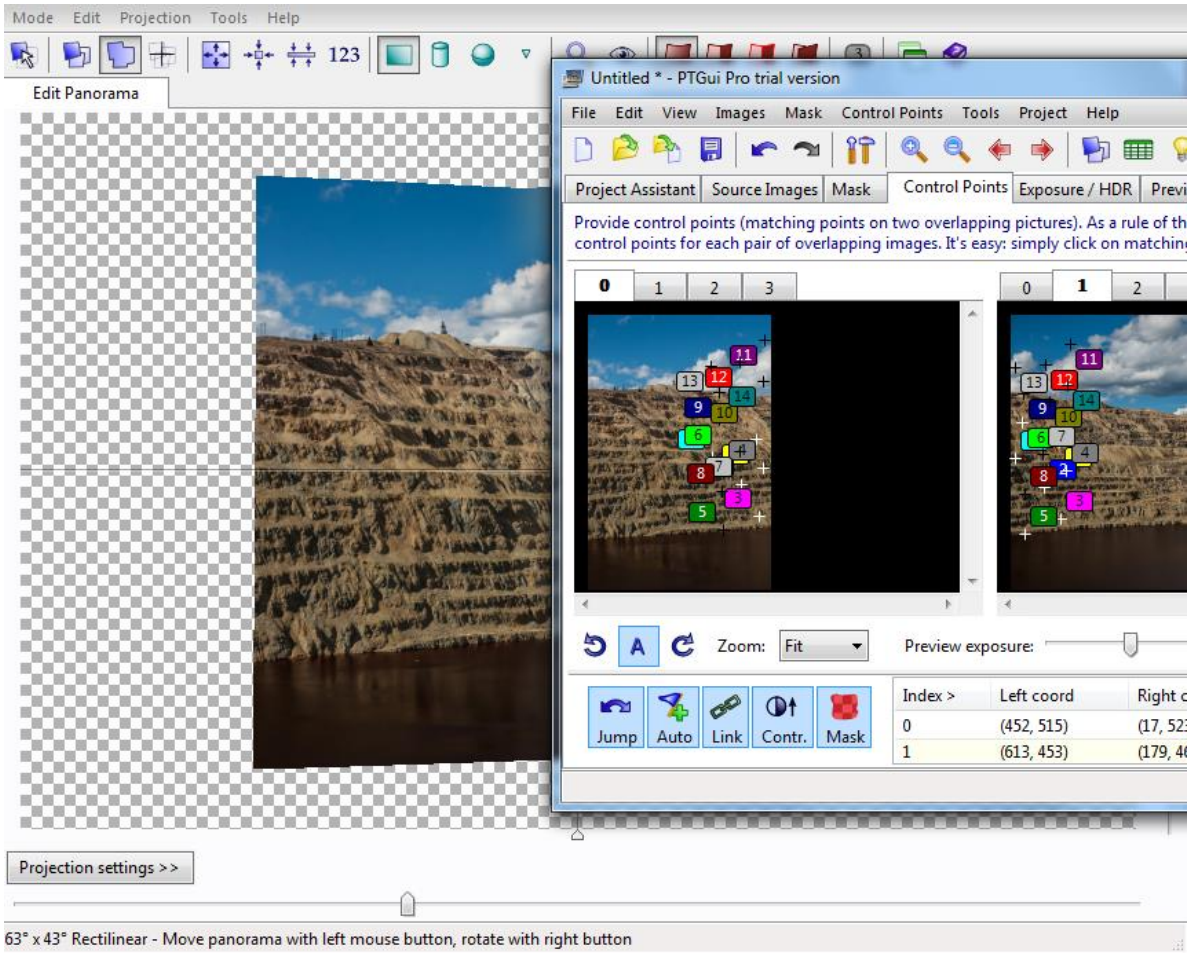

Рисунок 1.9 – Пример использования PTGui

1.5.4 Autopano Pro и Autopano Giga - средства для профессионального создания панорам, различающиеся полнотой функционала (Pro – облегченная версия).

Достоинства:

- предварительное автоопределение принадлежности исходных изображений к панораме;
- встроенный редактор входных и выходных изображений, включающий гомоморфные преобразования;
- большое количество настроек;
- возможность преобразования панорам в трехмерные сцены и 3D-туры.

Недостатки:

- высокая стоимость;
- стандартные ограничения для исходных изображений.

1.5.5 Все рассмотренные приложения имеют идентичный принцип использования и схожие требования к исходным изображениям:

- значительная доля перекрытий (рекомендуется 20-60%);
- фото сделаны из одной точки нахождения камеры методом поворота вокруг фиксированной оси;
- фото сделаны с одним и тем же уровнем яркости.

Следовательно, применяются схожие методы склейки и обработки изображений. Это говорит о потенциальной возможности полной или частичной ликвидации вышеперечисленных ограничений путем использования иных, возможно, экспериментальных, методов.

1.6 Постановка цели и задач дипломного проекта

Цель – разработка программного средства для автоматизированной генерации панорамных (широкоформатных) изображений из произвольных взаимосвязанных сегментов.

Задачи:

- разработка архитектуры ПС;
- разработка алгоритмов;
- выбор платформы для разработки программного средства;
- проектирование пользовательского интерфейса;
- создание базового проекта в среде программирования;
- создание пользовательского интерфейса;
- программирование и тестирование модулей;
- сборка и тестирование ПС.

2 МОДЕЛИ И СПЕЦИФИКАЦИЯ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ

2.1 Функциональная модель программного средства

При рассмотрении программного средства в самом общем виде, его функции могут быть представлены в виде соответствия входных и выходных данных. Входные данные - набор частично перекрывающихся изображений, вводимые пользователем параметры. Выходные данные - широкоформатное изображение, панорама.

С оглядкой на существующие прикладные реализации рассматриваемой задачи, определяются варианты использования программного средства (см. рисунок 2.1).

Рисунок 2.1 – Диаграмма вариантов использования

На основе вариантов использования для разрабатываемого программного средства формируется следующий набор функциональных требований:

1) Загрузка набора исходных изображений. Возможность выбора нескольких файлов, отображение уже загруженных, возможность добавления и удаления файлов из набора.

2) Генерация панорамы на основе набора исходных изображений. Отображение текущего статуса в процессе генерации. Возможность перезапуска генерации с изменением исходных данных или настроек.

3) Графическое представление панорамы, возможность его редактирования (масштабирование, позиционирование, обрезка краев).

4) Сохранение панорамы в файл на диске.

2.2 Спецификация требований к программному средству

2.2.1 Загрузка и сохранение изображений

Требования:

- поддержка популярных форматов изображений (JPEG, PNG, BMP) при загрузке сегментов и сохранении панорамы;
- поддержка кириллических символов в названии и в пути файлов;
- возможность сохранения текущего вида панорамы до или после изменения параметров её отображения.

2.2.2 Процесс генерации панорамы

Требования:

- невозможность запуска генерации при отсутствии или недостаточном количестве загруженных сегментов;
- отображение актуального состояния процесса, с периодом обновления текста не более минуты;
- наличие настроек по умолчанию.

2.2.3 Редактирование изображений

Требования:

- выбор инструментов из стандартного набора (масштабирование, сглаживающие фильтры);
- возможность выполнения автоматических операций (корректировка яркостей, удаление «призраков»);
- возможность отмены нескольких последних операций.

2.2.4 Пользовательский интерфейс программного средства

Требования:

- отображение приложения в перемещаемом окне;

- возможность сворачивания и разворачивания окна в любое время, независимо от статуса генерации.

2.2.5 Совместимость программного средства

Требуется совместимость с операционными системами Windows 7/8.

2.3 Модель предметной области

Разработанная модель предметной области представлена диаграммой на рисунке 2.2.

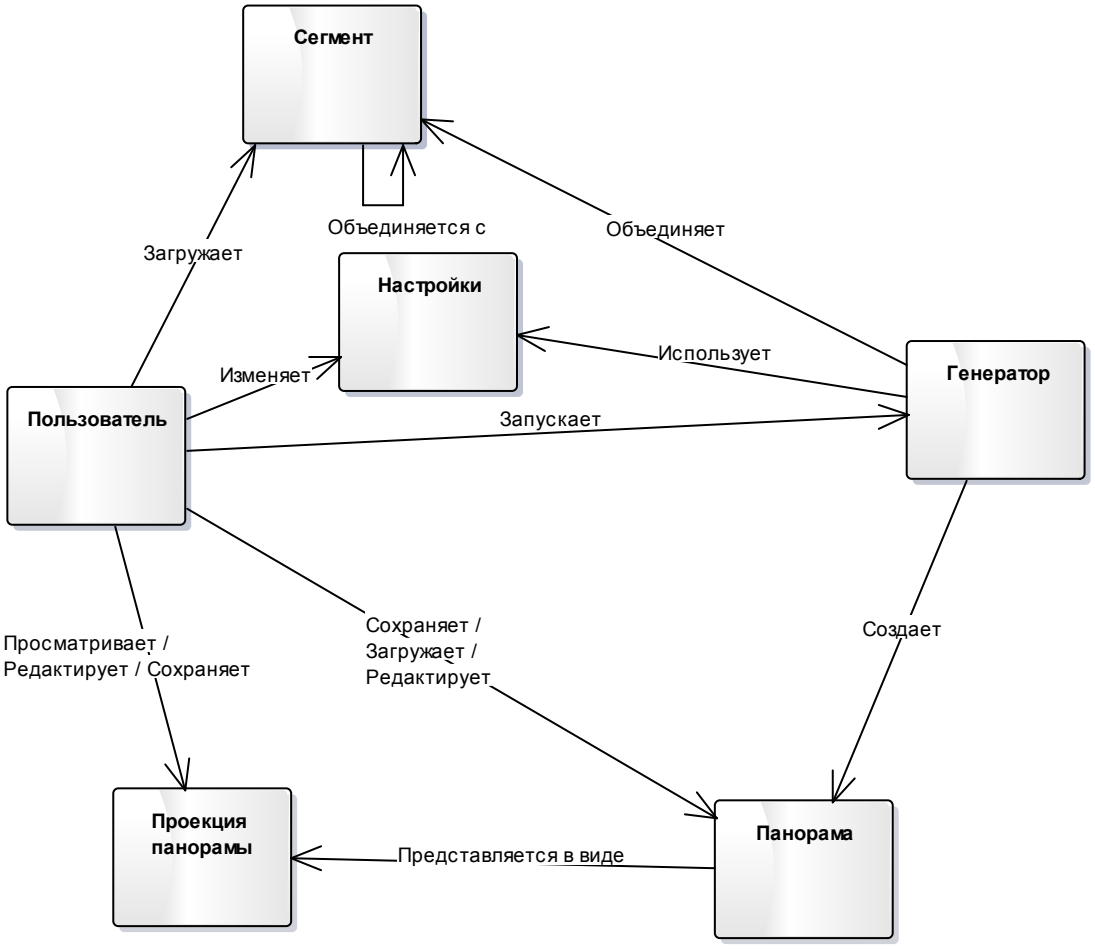

Рисунок 2.2 – Диаграмма модели предметной области

Модель предметной области основывается на требованиях к системе и представляет собой приближенный набор основных понятий и связи между ними. В данном случае объект «Пользователь» и его связи отображают функциональные требования. Его связь «сохраняет проекцию панорамы» означает экспорт в виде изображения, а «сохраняет панораму» - её сохранение в виде файла специального формата, предназначенного для дальнейшего

переиспользования в данном программном средстве. Также разделены операции по редактированию самой панорамы и её представления. Первая влияет на все последующие представления панорамы, вторая – только на текущее.

Модель «Генератор» - главный активный компонент программного средства, может служить опорным примитивом при разработке архитектуры.

2.4 Математическая модель программного средства

2.4.1 Модель преобразования координат

Перспективное преобразование имеет вид

$$\boldsymbol{x}` \sim \boldsymbol{M}\boldsymbol{x} = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{2.1}$$

где $\boldsymbol{x} = (x, y, 1)$ и $\boldsymbol{x}` = (x`, y`, 1)$ – гомогенные и проекционные координаты;
~ определяет равенство с точностью до масштаба.

Операция может быть переписана как

$$x` = \frac{m_0 x + m_1 y + m_2}{m_6 x + m_7 y + 1}, \tag{2.2}$$

$$y` = \frac{m_3 x + m_4 y + m_5}{m_6 x + m_7 y + 1} \tag{2.3}$$

(в модели перемещения лишь параметры m_2 и m_5 используются).

Такое преобразование предполагает чистое вращение камеры (на фиксированной оси), что является справедливым приближением при съемке отдаленных объектов. При условии, что известно фокусное расстояние, модель сокращается до четырех неизвестных параметров.

2.4.2 Прямая подгонка сегментов

Прямое выравнивание изображения $I_1(\boldsymbol{x})$ по отношению к изображению $I_0(\boldsymbol{x})$ заключается в опробывании различных вариантов вектора смещения **u** и нахождении такого, которое соответствует минимальной сумме квадратов ошибок (sum of squared differences, SSD):

$$E_{SSD}(\boldsymbol{u}) = \sum_i [I_1(\boldsymbol{x_i} + \boldsymbol{u}) - I_0(\boldsymbol{x_i})]^2 = \sum_i {e_i}^2, \tag{2.4}$$

где e_i – остаточная ошибка.

В общем случае, модуль вектора $\boldsymbol{u}$ может быть дробной величиной, и из-за этого к дискретным значениям интенсивности картинки I должна быть применена некоторая интерполяция.

Интерполяция, или интерполирование – в вычислительной математике способ нахождения промежуточных значений величины по имеющемуся набору дискретных значений. Линейная - приближение линейной функции по двум её значениям, билинейная – функции двух аргументов. При обработке изображений лучшие результаты (гладкую поверхность) дает бикубическая интерполяция, представляющее функцию как полином третьего порядка (с двумя неизвестными).

2.4.3 Поиск особых точек

Среди универсальных детекторов особых точек решено использовать метод SIFT, как наиболее точный. Далее будет рассмотрено его применение на некотором изображении.

Первый этап – построение пирамиды масштабов изображения при помощи фильтра Гаусса. Это широко применяемый фильтр сглаживания, определяемый формулой

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \qquad (2.5)$$

где x и y – координаты текущего пикселя относительно сглаживаемого (ядра размытия);

σ – коэффициент стандартного отклонения.

Из значений функции $G(x,y)$ образуется матрица свертки, налагаемая на окрестность изменяемого пикселя:

$$I`(x,y) = \sum_{dx,dy} I(x+dx, y+dy) * G(dx,dy). \qquad (2.6)$$

Для обеспечения максимальной точности матрица свертки должна покрывать всё изображение, однако, ввиду специфики функции Гаусса, вклад наиболее удаленных пиксели практически равен нулю в сравнении со вкладом ближайших, поэтому величину матрицы обычно ограничивают длиной стороны, равной 6σ.

Для нахождения особых точек на основе исходного изображения строится пирамида гауссианов, а затем – пирамида разностей гауссианов (DoG, Difference of Gaussian). Гауссианом условно принято называть изображение, к которому применен сглаживающий фильтр Гаусса. Пирамида гауссианов – многоступенчатое упорядоченное множество изображений, полученных с помощью последовательных применений фильтра Гаусса и масштабирований исходного изображения. Пирамида состоит из октав (см. рисунок 2.3), каждая определяется по некоторому значению масштаба. Все октавы включают в себя по N значимых изображений и 2 вспомогательных, каждое получено одно из другого с помощью размытия.

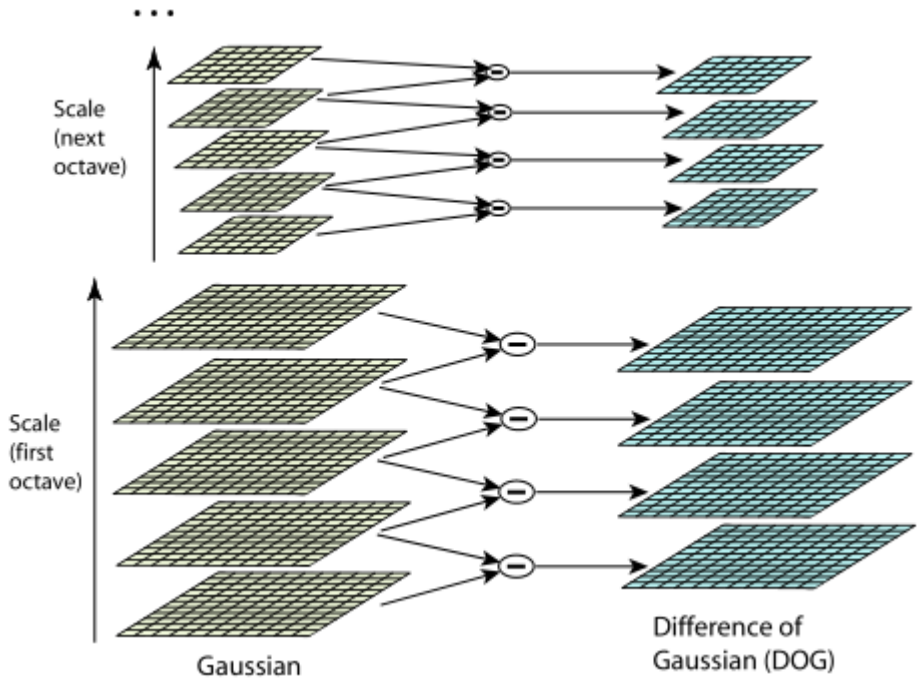

Рисунок 2.3 – Пирамида гауссианов и пирамида их разностей

Разность гауссианов – результат вычитания пиксельных значений одного гауссиана из другого, относящихся к одному изображению, но определяемых разным значением стандартного отклонения σ. Подобная операция позволяет локализовать градиенты интенсивности по их модулям, иначе говоря – разбивать изображение на частотные составляющие. Одним из применений разностей гауссианов является выделение флуктуаций (граней) на картинке. Также они позволяют определять особые точки, по их частотным свойствам.

Таким образом, построенная пирамида DoG служит для поиска локальных экстремумов значения разности. При этом пиксельная точка считается экстремумом (и потенциальной особой точкой), если 8 окружающих ее пикселей, а также 9 пикселей на изображениях выше и ниже по октаве, имеют меньшие (большие) значения разности. Дополнительные две изображения в каждой октаве пирамиды масштабов используются для полноты этой проверки.

Не все из найденных точки пригодны в роли ключевых. Для отбора каждая из них пропускается через ряд проверок. Предварительно следует уточнить координаты каждой точки, так как при масштабировании они получают некоторое отклонение. Для этого применяется интерполяция функции DoG в виде квадратичного многочлена Тейлора с ядром в точке экстремума [9]:

$$D(\boldsymbol{x}) = D + \frac{\partial D^t}{\partial \boldsymbol{x}}\boldsymbol{x} + 0.5\boldsymbol{x}^{\boldsymbol{T}}\frac{\partial^2 D}{\partial \boldsymbol{x}^{\boldsymbol{2}}}\boldsymbol{x}\,, \tag{2.7}$$

где $\boldsymbol{x} = (x, y, \sigma)$.

Точное расположение экстремума находится далее путем приравнивания производной функции приближения D($\boldsymbol{x}$) к нулю. При расчете координат следует учесть, что функция D(x) оперирует не абсолютными координатами, а координатами относительно ядра – рассматриваемого экстремума. И полученные значения x и y будут представлять собой корректирующий сдвиг ядра $\widehat{\boldsymbol{x}}$. Если одна из его компонент превышает значение 0.5, значит, рассматриваемый экстремум – ложный, и следует перейти к рассмотрению другого, в направлении $\widehat{\boldsymbol{x}}$ (если координаты не превышают размер изображения) [10]. Если значение функции интерполяции D(**x**) в точке $\widehat{\boldsymbol{x}}$ меньше 0.03, точка характеризуется низкой величиной контраста, и также отбрасывается.

Следующая проверка ликвидирует точки на гранях объектов. Подобные точки являются ярко выраженными экстремумами функции DoG, однако весьма чувствительны к шумам. Граничные точки характеризуются тем, что взаимно перпендикулярные градиенты в них значительно различаются по величине. Это определяется при помощи матрицы Гессе:

$$\boldsymbol{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix}, \tag{2.8}$$

где

$$D_{xy} = \frac{\partial^2 D}{\partial x \partial y}\,. \tag{2.9}$$

Если определить неизвестные α и β через систему

$$\begin{aligned} Tr(\boldsymbol{H}) &= D_{xx} + D_{yy} = \alpha + \beta \\ Det(\boldsymbol{H}) &= D_{xx}D_{yy} - \left(D_{xy}\right)^2 = \alpha\beta \end{aligned} \tag{2.10}$$

и представить r как

$$r = \frac{\alpha}{\beta}\,, \tag{2.11}$$

то условие отбраковки рассматриваемого экстремума:

$$\frac{Tr(\boldsymbol{H})}{Det(\boldsymbol{H})} \geq \frac{(r+1)^2}{r} \tag{2.12}$$

Оставшиеся уточненные экстремумы считаются особыми точками.

2.4.4 Построение дескрипторов

Дескриптор особой точки обеспечивает её стойкость к различным преобразованиям, например, к вращению и масштабированию. Дескрипторы SIFT зарекомендовали себя на практике и часто используются отдельно от прочих компонентов метода.

Для построения дескриптора необходима операция по вычислению компонент градиента произвольной точки:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}, \quad (2.13)$$

$$\theta(x,y) = tg^{-1}\frac{L(x+1,y) - L(x-1,y)}{L(x,y+1) - L(x,y-1)}\ , \quad (2.14)$$

где $L(x,y)$ – значение гауссиана с фиксированным значением σ;

m – величина градиента;

θ – его направление.

Сперва определяется окрестность точки, образующая дескриптор. Она выбирается из гауссиана с масштабом σ`, в полтора раза большим, чем у ключевой точки, и имеет вид окружности радиусом $3\sigma$`. Она делится на 36 равных секторов (по 10^o). Градиент каждой точки внутри окружности суммируется с другими из того же сектора, таким образом строится гистограмма направлений. Общее направление может уточняться с помощью интерполяции лидирующего и соседних с ним и нахождения максимума. Если имеются несколько отстоящих от главного направлений, близких к нему по величине (более 80%), они используются для создания новой ключевой точки, с теми же координатами и другой направленностью. Результат всех этих вычислений – направление ключевой точки. Перед выполнением дальнейших операций всё окружение точки (изображение) ориентируется на это направление. Так достигается инвариантность дескриптора к вращению.

На окрестности ключевой точки (на масштабе, ближайшем к точному) формируется квадратная окрестность, состоящая из блоков пикселей размером 4 x 4. Размер окрестности варьируется, в данном примере принят равным 16 (т.е. область имеет стороны длиной 16 пикселей и вмещает 16 блоков 4 x 4)[11].

На окрестности вычисляется дополнительный гауссиан – свертка с центром в ключевой точке. Затем для каждого блока в окрестности строится гистограмма направлений, по уже рассмотренному принципу, но на 8 направлений и с взвешиванием градиентов по соответствующим значениям новой свертки. Векторы гистограмм нормализуются по общей величине.

Результирующая совокупность шестнадцати гистограмм является дескриптором ключевой точки. Фактически он представляет собой массив 128

(4 x 4 x 8) чисел. Пример дескриптора типа 2 x 2 x 8 представлен на рисунке 2.4.

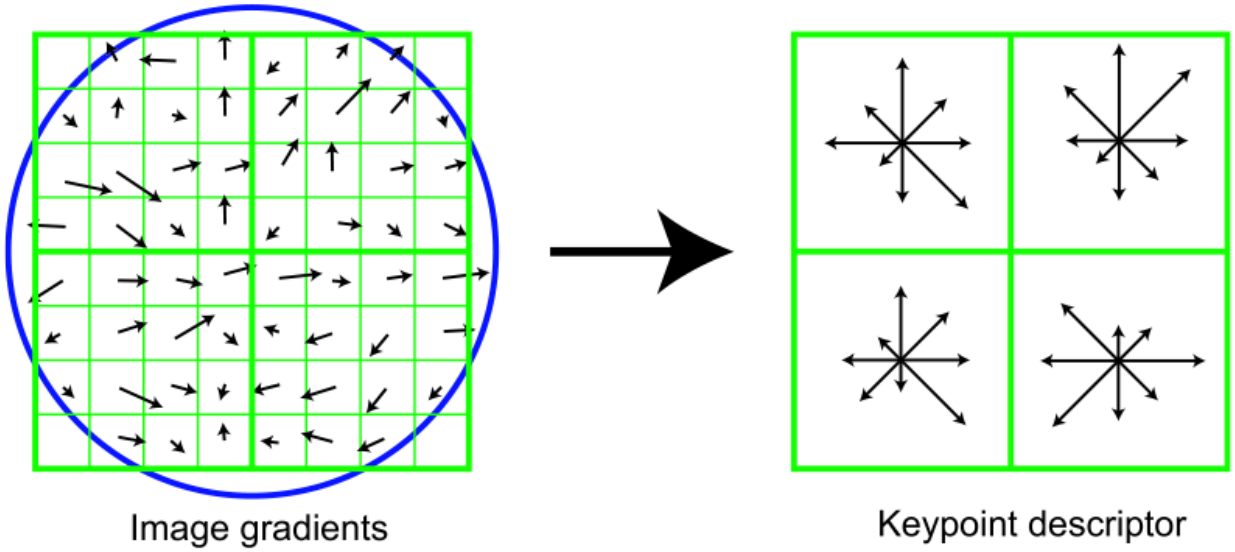

Рисунок 2.4 – Дескриптор особой точки в виде совокупности гистограмм направлений

2.4.5 Соотнесение особых точек

Имеется набор исходных сегментов с описанными особыми точками, необходимо соотнести их между собой оптимальным образом. Решение этой задачи должно быть выражено оператором преобразования координат, для которого необходимо определение восьми параметров (см. 2.2.1). Для вычисления четырех из них (перемещение, вращение, масштабирование) достаточно было бы одной пары соответствующих ключевых точек (по одной на каждом сегменте). Однако, для определения полного преобразования, необходимы четыре пары.

Нахождение соответствий между ключевыми точками может быть реализовано простым перебором с нахождением квадратов ошибок:

$$E_{SSD}(\boldsymbol{x_i}, \boldsymbol{y_j}) = \sum_k \left[\boldsymbol{x_{ik}} - \boldsymbol{y_{jk}}\right]^2 = \sum_k {e_k}^2. \qquad (2.15)$$

Выбираются пары со значением E_{SSD} меньше некоторого порога. Если таких оказывается слишком мало, изображения считаются несовместимыми, и рассматривается следующая пара сегментов.

Если количество ключевых точек велико, полный перебор может оказаться весьма длительным процессом. Применение метода поиска ближайших соседей с вейвлетной индексацией позволяет сравнивать ключевую точку только с теми точками, которые близки к ней по градиентным характеристикам. Для этого строится трехмерная матрица, в которой каждая ключевая точка регистрируется по трем координатам - c_1, c_2 и c_3:

$$c_1 = \frac{\partial I}{\partial x}; c_2 = \frac{\partial I}{\partial y}; c_3 = \frac{\partial^2 I}{\partial x \partial y}. \quad (2.16)$$

Пространство каждой меры в матрицу разбивается на b частично перекрывающих друг друга интервалов (*bins*). Матрица используется при итерировании по ключевым точкам, для сокращения количества их комбинаций. Минус такого метода заключается в возможности ложного отсеивания.

Высока вероятность нахождения случайных пар ключевых точек, которые приведут к конфликту при вычислении трансформации. Во избежание таких ситуаций используется алгоритм RANSAC. Варианты преобразований рассматриваются как модели, а пары соответствующих ключевых точек – зашумленные исходные данные. Согласно алгоритму, среди пар выбираются случайные комбинации (в данном случае – по 4 пары), и по ним строятся модели. Модель проверяется на корректность путем пропускания через неё остальных данных и проверки совпадения значений. Так как исходные данные (координаты точек) обладают некоторой долей погрешности, для проверки используется диапазон значений отклонения, в пределах которого оно считается удовлетворительным (см. рисунок 2.5). Точный размер диапазона выбирается в результате проб и экспериментов.

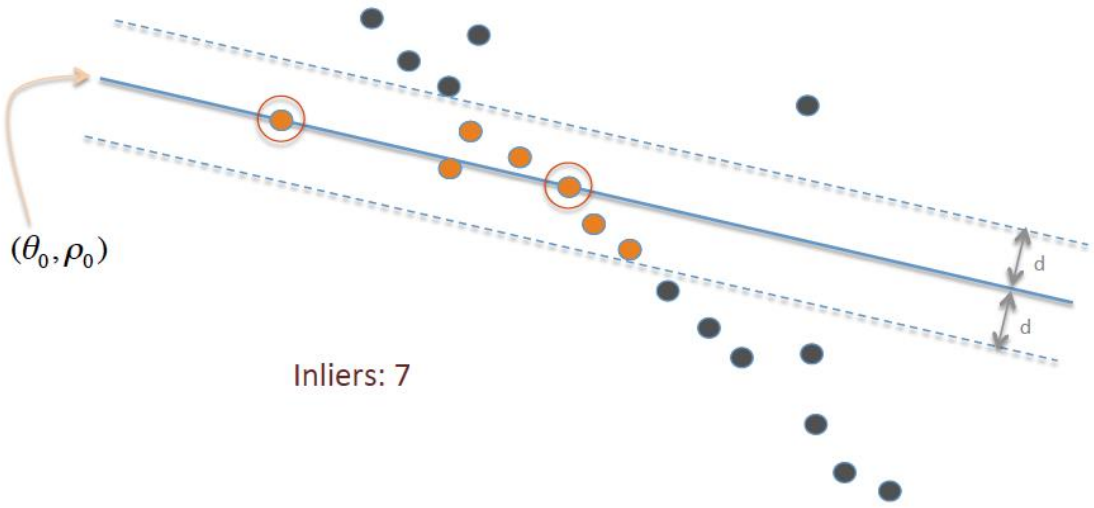

Рисунок 2.5 – Диапазон корректных значений в алгоритме RANSAC

Результат работы RANSAC – достоверный оператор преобразования координат, определяющий положение одного сегмента панорамы относительно другого. Параметры преобразования могут быть уточнены

После перебора всех комбинаций сегментов возникает совокупность относительных преобразований, которые необходимо представить в виде глобальной системы.

2.4.6 Корректировка

При склейке изображений зачастую образуются области перекрытия, не совпадающие на 100%. Существует два пути решения этого конфликта: смешение пикселей разных сегментов или проведение четкой границы между ними.

Простейшее решение при реализации последнего варианта – выбирать один из сегментов по какой-либо признаку и замещать его пикселями пиксели другого. Это создает риск возникновения видимых швов ввиду разности освещений или кривизны проекций. Чтобы шов был незаметен, его следует провести особым образом. Для этого используется алгоритм Graphcut [14]. Перекрываемая область представляется в виде связного графа пикселей (см. рисунок 2.6), в котором вес дуги равен разности между пикселями смежных сегментов, а именно:

$$M(s,t,\boldsymbol{A},\boldsymbol{B}) = \|\boldsymbol{A}(s) - \boldsymbol{B}(s)\| + \|\boldsymbol{A}(t) - \boldsymbol{B}(t)\|, \qquad (2.17)$$

где $\boldsymbol{A}$ и $\boldsymbol{B}$ – сегменты;

t и s – смежные пиксели (вершины графа).

В таком контексте нахождение шва становится задачей нахождения минимального среза, которая решается с помощью графового алгоритма поиска максимального потока. Две особые вершины обозначают грань области перекрытия. Вес их связей со смежными вершинами равен бесконечности, таким образом, они не участвуют в поиске. После нахождения оптимального среза, граф используется как маска для выбора пикселей из разных сегментов.

Смешение пикселей производится быстро путем замены интенсивностей средним значением конфликтующей пары. Результирующее изображение будет хранить информацию обоих сегментов, что может быть полезным свойством, но порождает ряд проблем.

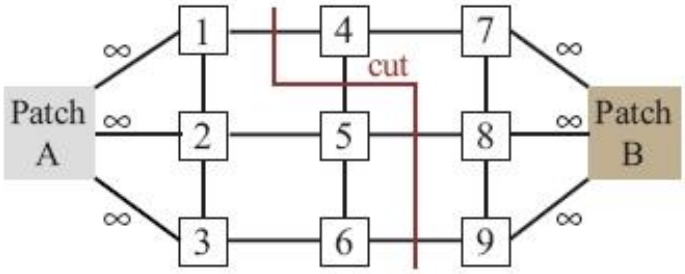

Рисунок 2.6 – Граф для определение оптимального шва на перекрываемой области

Одна из проблем – шум. Малый уровень шума легко ликвидируется с помощью медианного пространственного фильтра. Из упорядоченного набора интенсивностей всех пикселей в окрестности рассматриваемого выбирается среднее значение. Оно применяется как новая интенсивность пикселя. Размер окрестности варьируется в зависимости от требуемой мощности фильтра.

Другая проблема – «призраки». При создании панорамы на основе фотографий, следует учесть, что они, скорее всего, сделаны не одновременно. Если в кадр попадает движущийся объект, он будет присутствовать лишь на некоторых сегментам, что вызовет видимый дефект при их склеивании. Проведение шва зачастую исключает такую проблему, ей подвержено смешение пикселей. Для решения проблемы можно использовать алгоритм Uyttendaele [3]. Он заключается в поиске областей, характеризуемых большой разностью между пикселями двух сегментов. В результате последовательного прохода по области перекрытия определяется конфликтный регион (по наибольшим значениям разностей интенсивности). Он представляется в виде связного графа, в котором вершины – конфликтующие пиксели. В ходе досконального прохода по всем вершинам графа, соответствующие им пиксели панорамы получают смещение интенсивности (т.е. выбираются из одного сегмента). Таким образом можно либо удалить призрачный объект с картинки, либо сделать его более четким – в зависимости от того, пиксели какого сегмента будут выбраны.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Архитектура программного средства

Архитектура программного средства – его высокоуровневое структурное описание, определяющее роли основных компонентов системы и связи между ними. Она должна обеспечивать эффективное решение по реализации требований к системе, оптимальным образом отражать варианты использования.

3.1.1 Выбор типа программного средства

Фундаментальные требования к разрабатываемому программному средству были поставлены в разделе 2.1. Из них следует, что ключевыми операциями программного средства будут манипулирование и обработка графической информации. У системы четко определены входные и выходные данные - изображения, причем первые почти однозначно определяют последние. Эта связь дает возможность сделать систему в высокой степени замкнутой, минимизировать ее внешние зависимости, приблизив к концепции «черного ящика». Данный подход тем более целесообразен, если процесс работы ПС имеет комплексный характер. Так, генерация панорамы состоит из ряда этапов и подэтапов обработки данных различного характера.

Однако, полноценный «черный ящик» в данном случае неприменим, так как пользователь должен иметь возможность управлять и вносить изменения в процесс работы ПС. Необходимо предоставлять ему информацию о процессе наиболее оптимальным образом – графическим (ввиду характера работы системы). В современных операционных системах наиболее распространенным средством ввода и вывода информации являются окна – модульные элементы интерфейса, объединенные общей системой управления. Для разработки приложений, использующих собственные окна, существует множество эффективных программных решений. Это говорит о целесообразности разработки программного средства в виде оконного приложения.

Среди требований присутствуют работы с файлами изображений, что означает взаимодействие с файловой системой действующего компьютера. К счастью, все современные платформы для разработки имеют средства абстрагирования подобных взаимодействий, а также их оптимизации. К последним относится использование системных диалоговых окон для удобной работы пользователя с файлами. Это еще один аргумент в пользу оконного приложения.

3.1.2 Программная архитектура

Процесс генерации панорамы состоит из ряда сложных процедур анализа и преобразования данных различного вида. Это означает наличие высокого риска создания недостаточно гибкой, хрупкой архитектуры, при

которой внесение любого изменения в общую структуру может приводить к непредсказуемым последствиям. Поэтому ключевой характеристикой архитектуры должна быть простота, выражаемая в легкости, с которой разработчик сможет ориентироваться в существующей системе и абстрагировать отдельные ее части.

Элементарным приемом для упрощения сложных программных решений является принцип модульности – разделения кода на цельные компоненты. Такой прием позволяет, по возможности, изолировать несвязные фрагменты программы друг от друга, что дает множество преимуществ, среди которых:

- разграничение ответственности;
- возможность параллельной работы нескольких команд разработчиков;
- возможность переиспользования кода;
- простота тестирования.

Разграничение ответственности требует особого способа разбиения кода на модули. Если за некоторый набор функций отвечает четко определенный модуль, работающая над этими функциями команда разработчиков не будет конфликтовать с другими, отвечающими за другие функции. Также ограниченность набора функций модуля означает ограниченный набор способов его использования, и простоту тестирования. Наконец, модульность подразумевает внутреннее разбиение крупных компонент на более мелкие, с отношениями включения. Если схожий код требуется во многих местах, он может быть вынесен в единый модуль, с общей реализацией для всех вариантов использования. Это сокращает общее количество программного кода и избавляет от ошибок при его модификациях.

Модульность упрощает программирование в пределах малого модуля, однако с продвижением вверх по иерархии существенно возрастает уровень ответственности. Современным решением для эффективного абстрагирования и делегирования ответственностей является объектно-ориентированное программирование. Оно основано на представлении программы в виде набора объектов – структур, содержащих информацию и определяющих некоторые операции (методы). Объекты абстрагируются в виде классов, интерфейсов и модулей. Они ссылаются друг на друга и могут составлять иерархии неограниченной сложности. Отношение между двумя классами, когда изменение одного приводит к неизбежному изменению другого, называется зависимостью. Одним из главных достоинств объектно-ориентированного дизайна является то, что он выявляет подобные зависимости и позволяет управлять ими – сокращать или оборачивать вспять. Этой цели служит принцип полиморфизма – использование разных сущностей с помощью одного интерфейса. Использование интерфейсов также означает ограничение взаимодействий с объектом – наличие у него индивидуального пространства для хранения и использования кода (принцип инкапсуляции). Область влияния этого скрытого кода четко определена, что сильно упрощает отладку программы.

Помимо всего вышеназванного, очевидным достоинством объектно-ориентированной архитектуры является ее способность отражать предметную область. Логично организовывать функции и информацию в классы сообразно сущностям, упомянутым в вариантах использования – будь то изображение, панорама или сам пользователь. Оперирование программными абстракциями, имеющими реальные аналоги, облегчает понимание кода и делает разработку отчасти интуитивной. Не все классы в приложении описывают сущности из предметной области. По мере снижения уровня абстракции появляются классы, имеющие специфические роли уже в терминах конкретной программы, платформы разработки или операционной системы.

Таким образом, определен общий вид программы как системы связанных и взаимодействующих объектов и их абстракций. Согласно принципу модульности, каждый из объектов должен отвечать за некую определенную для него долю функций программного средства, напрямую или косвенно. Эффективное распределение ролей между модулями – одна из главных архитектурных задач.

Всякое модульное приложение использует какую-либо модель отношений между его компонентов в приложении, практически всегда – набор моделей разного уровня абстракции. Распространенность принципа модульности привела к стандартизации наиболее эффективных из них в виде «паттернов» (приемов проектирования). Проектирование большой системы практически невозможно без использования этих приемов.

Один из наиболее общих приемов проектирования - модель EBC (Entity-Boundary-Control). Она определяет модель из элементов трех категорий:

- сущности (entity);
- граничные элементы (boundary);
- управляющие элементы (control).

Сущности – элементы, предназначенные для хранения информации, чаще всего относящиеся к предметной области. Граничные элементы являются интерфейсом всего приложения, они взаимодействуют с пользователем (или другим актером). Элементы управления определяют внутреннюю работу системы по генерации выходных данных на основании входных, полученных из граничных классов. Фактически, элементы управления являются непосредственными реализаторами вариантом использования ПС.

Для проектируемого приложения в разделе 2.3 разработана модель предметной области. Она может быть использована для конкретизации категорий элементов в контексте модели EBC. Так, ключевое понятие – «Панорама» - представляет собой наиболее очевидный кандидат на роль сущности. «Сегменты», из которых она состоит, являются контейнерами графической информации – это также сущность в приложении. «Генератор» - активный функциональный компонент, следовательно, это должен быть элемент управления. «Настройки», которыми он пользуется – еще один контейнер. «Проекция панорамы» - панорама в виде простого изображения.

Платформа разработки предоставит стандартный класс, который заменит эту сущность в приложении. Последнее понятие из модели - «Пользователь». Функции не предполагают хранения какой-либо информации о пользователе. Это понятие отображает инициативную сторону – агента, внешнего по отношению к программному средству. В подобном элементе нет необходимости.

Кандидаты на роль граничных классов были описаны в разделе 3.1.1 – это окно и диалог для управления файлами.

Функции по загрузке/сохранению файлов, запуску генератора и редактирования изображений, будут реализованы элементами управления. Функция панорамы «представляется в виде проекции» реализуется элементом управления «Представитель Панорамы».

Все разработанные элементы модели, их роли и зависимости представлены на диаграмме на рисунке 3.1.

3.2 Проектирование пользовательского интерфейса

Прежде чем начать детализацию программного средства как системы компонентов, необходимо уделить внимание его внешнему представлению – тому, с чем непосредственно будет работать пользователь. Эффективное выполнение своей роли – главная задача любого программного средства, а в случае, если это пользовательское приложение, удобство использования является одним из важнейших условий эффективности.

Пользовательский интерфейс (UI, User Interface) – это способ взаимодействия человека и машины, включающий совокупность действий, совершаемых пользователем, и результатов этих действий. Критериями качества интерфейса является его эффективность и удобство пользования. Структурно интерфейс является совокупностью определенных средств и методов. Средства в данном контексте – элементы, предназначенные для ввода информации в устройство или ее вывода для отображения пользователю. Методы – динамическая составляющая интерфейса, набор правил по использованию имеющихся средств.

Интерфейс разрабатываемого программного средства должен быть двунаправленным. Программе для выполнения своей задачи необходимы данные, определяемые пользователем – сегменты панорамы. Пользователь для возможности предоставления этих данных должен получить и понять запрос от программного средства. Когда программа завершила процесс генерации панорамы, тот предоставляется пользователю, вместе с возможностью редактирования и дальнейшего использования в каких-либо целях.

Ввиду сильной зависимости выходных данных (панорамы) от входных (сегментов), возможность вмешательства в работу может быть излишней в тех случаях, когда система самостоятельно вырабатывает оптимальный результат. Поэтому следует минимизировать необходимость любого вмешательства и

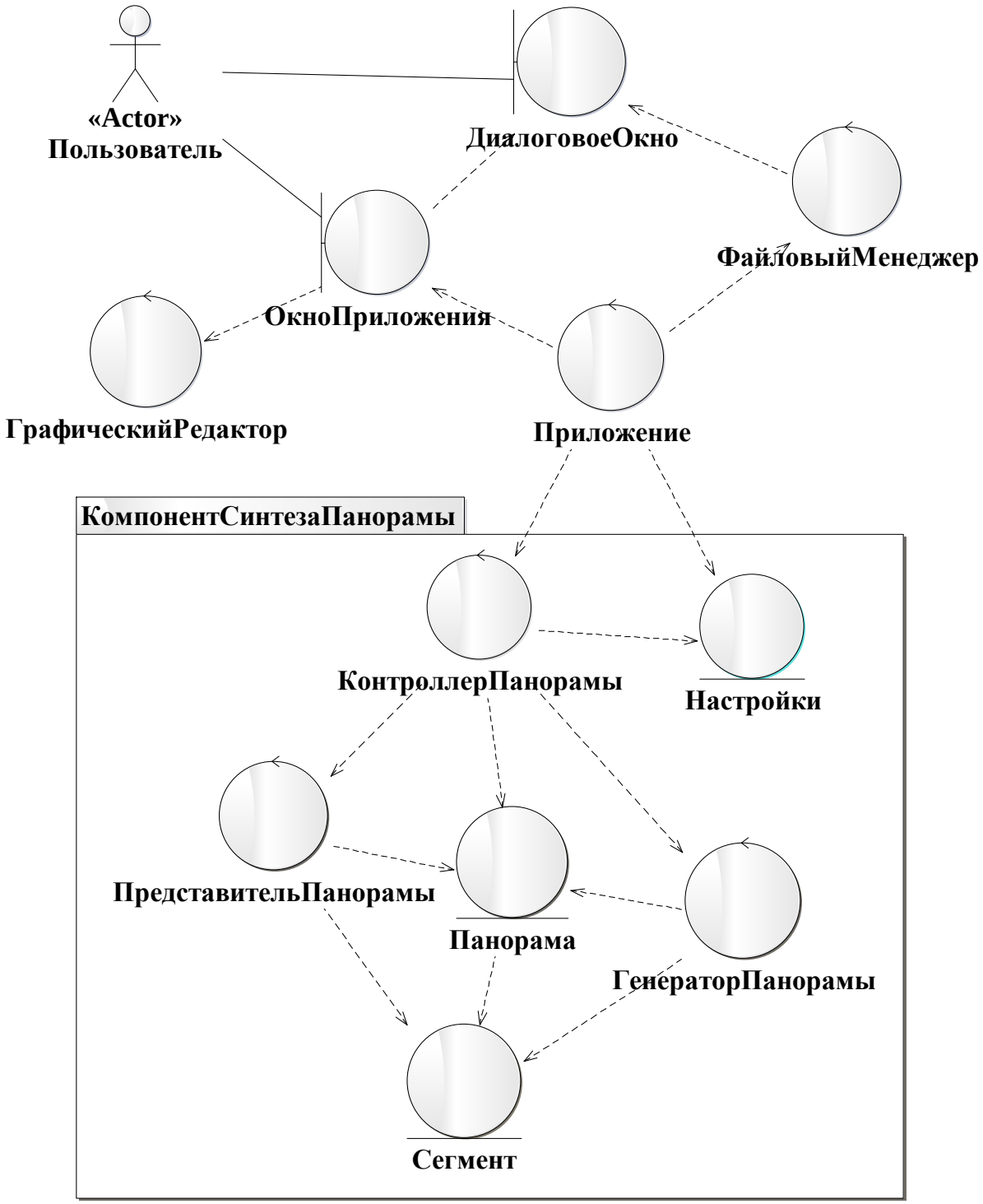

Рисунок 3.1 – Аналитическая модель программного средства

предоставить пользователю предварительный результат. В случае неудовлетворительной оценки он должен иметь возможность изменить результат непосредственно или определить этап, на котором дефект может быть устранен, внести изменения в ход этого этапа и оценить их влияние. В случае, если система может самостоятельно оценить качество генерируемой

панорамы как неприемлемое, от пользователя сразу потребуется выполнение некоторых корректирующих действий.

Требования к пользователю должны быть ему понятны и очевидны. Следовательно, необходим механизм доставки сообщений, который будет помогать ему в навигации по интерфейсу и при этом не являться помехой для эффективного использования этого интерфейса. К примеру, на окне приложения может быть выделена постоянная область, предназначенная для отображения сообщений пользователю. Так как содержание области будет меняться по ходу работы приложения, и поэтому не всегда отвечать на для пользователя актуальные вопросы, следует дать ему возможность задания вопроса. Частой реализацией этого является кнопка «Справка» или ссылки на документацию в контекстном меню окна.

Предоставляемые пользователю инструменты должны быть целесообразны ожидаемым от него действиям и удобны в использовании. Имеет смысл взять за основу средства, получившие широкое распространение среди других популярных приложений, которые будут наиболее близки и понятны пользователю.

3.3 Алгоритмы программного средства

В подразделах 3.1-3.2 велось поэтапное проектирование общих внешнего и внутреннего вида программы. Результат – структурное описание, которое послужит основой для реализации в виде программного кода. Однако, это описание является неполным. Оно не учитывает динамического аспекта функционирования программного средства, иначе говоря, способа приведения всей структуры в действие.

3.3.1 Схема работы программы

Весь процесс функционирования программы может быть описан структурно – как дискретное множество операций, связанных между собой порядком и условиями выполнения.

Программа начинает процесс выполнения в результате запуска пользователем. Она загружает текущие настройки, которые могут быть заданы пользователем, и использует их при последующей за этим инициализацией компонентов системы. На данном этапе необходимым компонентом является окно приложения, которое предоставит пользователю возможность работы с программой. Для функционирования по назначению программе требуются входные данные, поэтому после запуска окно отобразит интерфейс для передачи этих данных (изображений). Интерфейс состоит из стандартных диалоговых окон для выбора файлов на компьютере.

Получив входные данные, программа способна самостоятельно выработать выходные. Для сопоставления друг с другом изображения проходят через стадию анализа, на которой извлекаются их статистические атрибуты. Пройдя через несколько стадий обработки, эти атрибуты далее

служат основой для синтеза выходного результата в ходе процедуры генерации панорамы. Результат генерации – первоначальный вариант панорамы - подается на главное окно и предоставляется пользователю для оценки.

Если результат удовлетворителен по мнению пользователя, он может экспортировать его в виде файла (сохранить изображение). В этом случае программа достигла своей цели и может быть завершена (или использована для новой задачи – на усмотрение пользователя).

Если же результат оказался неудовлетворительным, пользователь может приступить к исправлению выделенных им дефектов. Дефекты могут порождаться на различных стадиях работы программы, поэтому применение изменений, для наибольшего удобства использования, должно быть возможно с использованием инструментов различных уровней. Простейшие средства редактирования изображения являются первой из доступных возможностей. Пользователь уведомляется об этом и применяет инструменты. Если они оказываются недостаточны для исправления дефекта, совершается переход к более раннему этапу процесса, в данном случае – к объединению сегментов в панораму. Графический интерфейс должен удобно отображать принятые программой решения касательно отношения между сегментами и их свойствами и позволять редактировать эти данные. далее пользователь может перезапустить генератор панорамы и оценить измененный результат. Если и эти средства оказываются недостаточными, остается возможность изменения наборы входных данных. Отображение отношений между сегментами может помочь пользователю выявить отдельные изображения, являющиеся причиной дефекта, и исключить их из исходного набора. Другим вариантом является добавление новых сегментов, что является обычным способом повышения качества панорамы. Обо всех этих возможностях пользователь будет уведомлен посредством сообщений.

Алгоритм работы программы в схематическом виде приведен на рисунке 3.2.

3.3.2 Алгоритм генерации панорамы

В результате теоретических исследований, описанных разделе 1, было найдено множество алгоритмов для сравнения пары изображений между собой и их склейки в одно, однако реальная связь между этими двумя этапами в рамках процесса генерации панорамы неопределенна. Между тем, она является одним из определяющих моментов всего процесса. Открытым остается вопрос, каким образом связи между парами сегментов преобразуются в связи сегментов с глобальной картиной (панорамой). В пункте 1.2.3 были упомянуты два метода: аккумулирующая регистрация и регистрация с помощью распределения общей ошибки. Под ошибкой здесь понимается

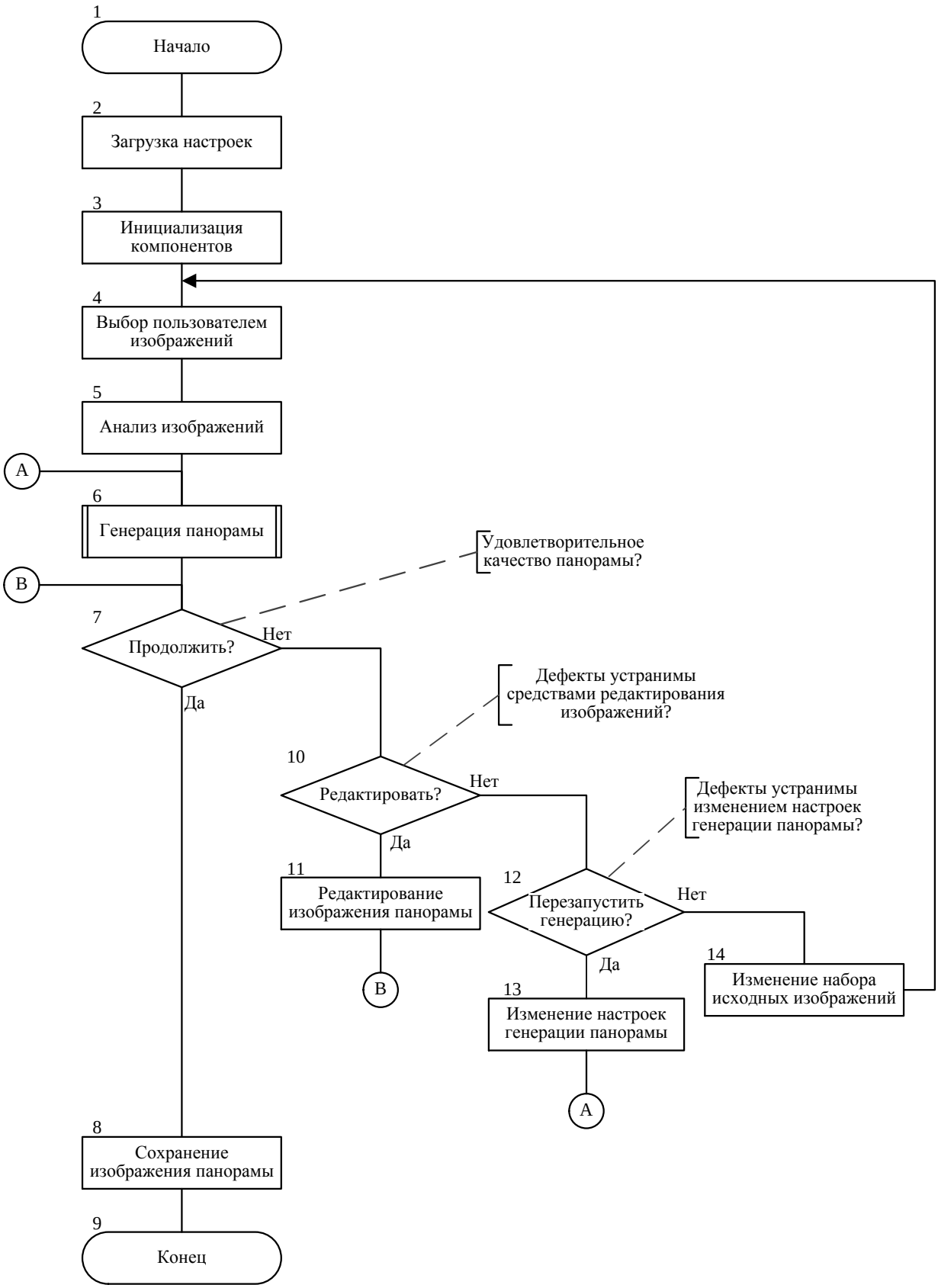

Рисунок 3.2 – Схема работы программы

разница между корректным и вычисленным расположением сегмента в контексте всей панорамы. Первый из методов имеет существенный недостаток, заключающийся в последовательном накоплении ошибки, которое в итоге может в сильной степени исказить результат. Второй метод абстрактен и имеет недостаточное количество примеров конкретных реализаций. Следовательно, необходима доработка и приспособление вышеуказанных методов.

В разрабатываемом ПС решено использовать метод глобальной регистрации сегментов, основанный на аккумулирующем методе. Разница состоит в том, что связи между изображениями составляет не линейную цепь, а структуру данных типа «дерево». Регистрация сегментов осуществляется рекурсивно, начиная с корня дерева, и далее по всем его потомкам. Таким образом, ошибка при вычислении расположения каждого сегмента зависит не от всех предшествующих ему, но лишь от стоящих на его пути к корню дерева. Это дает возможность строить дерево, используя те связи между сегментами, которые характеризуются наименьшим значением ошибки. На практике ошибка напрямую зависит от схожести сегментов, т.е. степени их перекрытия и специфики области перекрытия. Чем более схожи изображения, тем большей точностью обладает процесс определения их связи.

Генерация панорамы возможна сразу после получения системой входных данных. Согласно описанному выше методу, необходимой информацией для процесса генерации являются оценки схожести между сегментами. Когда все отношения определены, начинается процесс создания дерева. Структура типа дерево всегда имеет строго один элемент в роли корня. Согласно описанному ранее алгоритму регистрации, ошибка при выводе всех сегментов зависит от суммарной ошибки их связи с корнем дерева. Следовательно, логично выбрать корнем тот сегмент, который характеризуется наибольшей степенью схожести с другими сегментами. Данная метрика использует ранее сделанные оценки схожести.

Далее осуществляется переход на второй уровень иерархии. Среди всего множества еще не зарегистрированных в панораме сегментов выбираются те, для которых корень является наиболее схожим сегментом. Они непосредственно связаны с корнем, и все оставшиеся сегменты будут зависеть от ошибок этих связей. Для каждого полученного элемента второго уровня иерархии выполняется та же операция, что и для корня – поиск и присоединение ближайших. То же выполняется для этих ближайших, и так далее. Данная процедура может быть просто реализована с помощью рекурсии. Рекурсия – прием структуризации, при котором объект включается в самого себя. В контексте алгоритмизации, это вызов некоторой процедурой самой себя. Обязательным требованием к реализации рекурсии является задание корректного условия, при котором цикл вызовов прекращается. Так, при построении дерева процедура регистрации в нем сегмента вызывает саму себя в отношении некоторых других, еще не зарегистрированных, сегментов.

Если множество сегментов конечно, на каком-то этапе процедура не станет вызывать себя из-за отсутствия сегментов.

Важно отметить, что рекурсивная процедура регистрации, вызванная для некоторого сегмента, использует именно те из незарегистрированного множества, по отношению к которым он является ближайшим. Таким образом, созданная связь между двумя сегментами – наилучшая для дочернего, так как сильная схожесть обеспечивает малую величину ошибки. Однако, подобный подход порождает проблему. Например, случай, когда все оставшиеся сегменты слишком схожи между собой, чтобы быть включенными в дерево. Решение этой ситуации – парный перебор сегментов каждого из множеств с поиском наиболее схожей пары. Она обеспечит наилучшую связь между деревом и незарегистрированными элементами. Для свободного элемента найденной пары вызывается рекурсивная процедура регистрации.

Процедуры совмещения множеств и регистрации повторяются до тех пор, пока все сегменты не зарегистрированы в дереве.

Алгоритм создания дерева и рекурсивная процедура регистрации сегмента в нем представлены на рисунке 3.3.

3.3.3 Алгоритм сравнения сегментов панорамы

Для работы процедуры генерации панорамы, описанной в пункте 3.3.2, необходимыми данными являются оценки схожести всех исходных изображений. От фактической схожести сегментов зависит точность их склейки в панораме, следовательно, процедуры сравнения двух сегментов и определения трансформации между ними должны быть связаны общей основой.

В пункте 1.2.2 был описан метод выравнивания изображений, основанный на поиске их локальных особенностей. Так, каждый сегмент панорамы может быть описан конечным множеством особых точек, которые могут присутствовать и на других сегментах. Даже при наиболее оптимальном способе выбора этих точек, вероятность их полного совпадения на двух разных сегментах (с перекрывающейся областью) практически равна нулю, что порождает необходимость использовать статистические методы (RANSAC) для определения самого вероятного способа преобразования одной группы точек в другую.

Однако, использование локальных особенностей дает возможность оценить схожесть изображений на основе схожести их особенностей. Так, эквивалентные особые точки на разных сегментах панорамы означают присутствие на них обоих одного объекта (либо весьма схожих объектов). Поэтому, особые точки, которые будут использоваться для склейки сегментов, имеет смысл вычислить заранее для их сравнения.

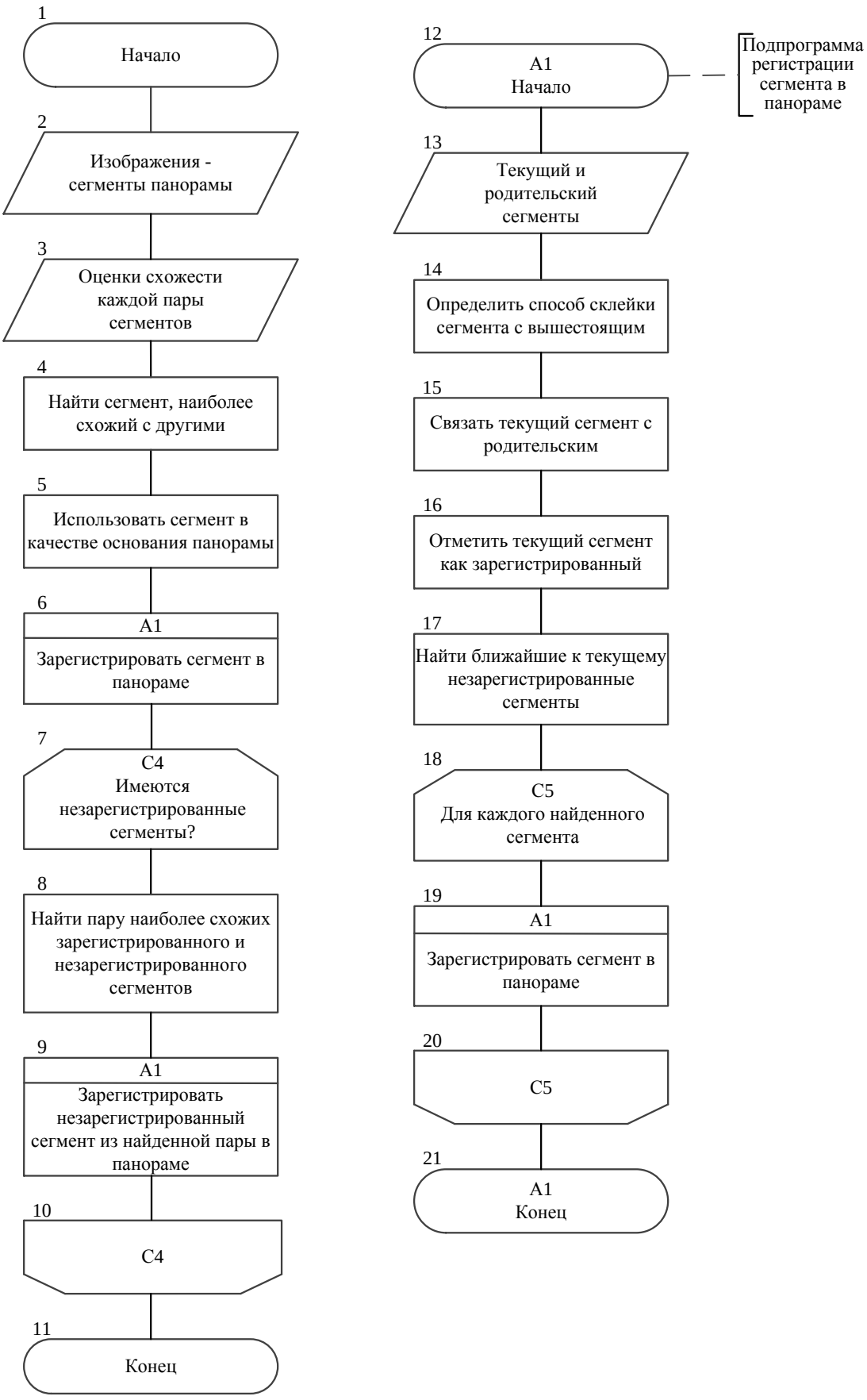

Рисунок 3.3 – Схема алгоритма генерации панорамы

Эта процедура осуществляется в ходе анализа загруженных программой изображений. С помощью эффективного метода SIFT для каждого из них определяется набор особых точек (в дальнейшем – точек). В ходе попарного сравнения всех сегментов для каждой их точки находится наиболее схожая с ней на другом изображении методом поиска ближайшего соседа. Все пары «соседей» запоминаются и сортируются по их схожести (расстоянию) между собой.

Метрика расстояния между точками должна быть выбрана таким образом, чтобы пары, описывающие один объект на двух изображениях, имели расстояние, близкое к нулю. В таком случае среди упорядоченного множества пар можно выделить те, которые с наибольшей вероятностью описывают общую часть двух изображений. В дальнейшем, использование этих данных может также существенно ускорить выполнение метода RANSAC при склеивании сегментов.

Метрика расстояния между сегментами должна характеризоваться прямой зависимостью от количества таких пар точек и обратной – от схожести этих пар.

Таким образом, в результате выполнения процедуры будет определено значение схожести каждой пары сегментов. Это значение, а также использованные пары особых точек (ближайших соседей) сохраняются для дальнейшего использования в процедуре генерации панорамы.

Алгоритм сравнения представлен в полном виде на рисунке 3.4

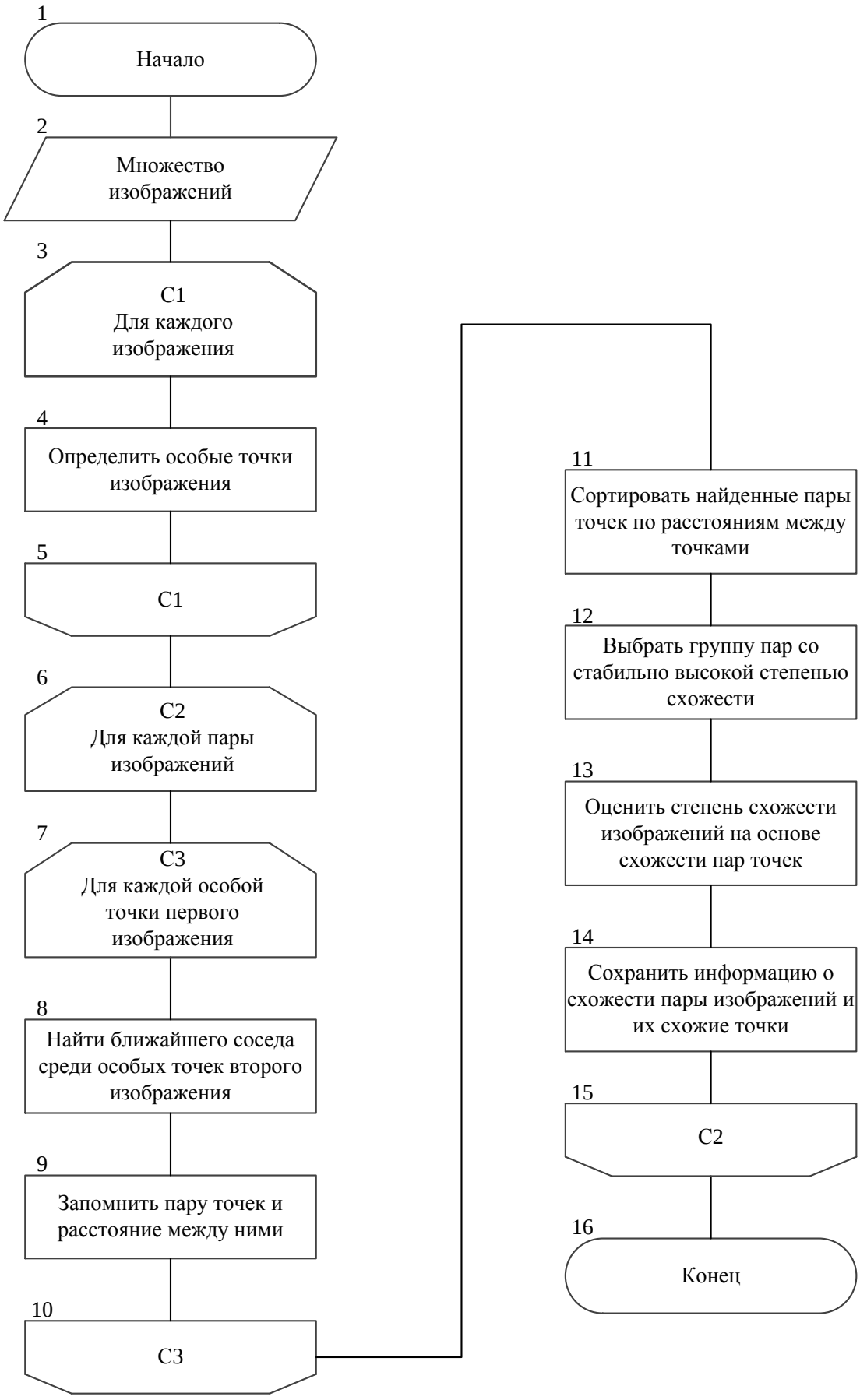

Рисунок 3.4 – Схема алгоритма анализа схожести изображений

4 КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В разделе 3 описана разработка архитектуры программного средства, включающая общий вид его структуры и порядок функционирования. Таким образом, определена основа, достаточная для реализации программы в виде исполняемого кода. Обычным средством для реализации является платформа разработки, позволяющая программистам работать с удобным набором команд вместо бинарного машинного кода.

4.1 Выбор средств разработки

В подразделе 3.1 был определен общий вид программы - оконное приложение, а также характер выполняемых работ – обработка графической информации. Из этого следует ряд требований, определяющих выбор средств разработки:

- возможность создания графического интерфейса;
- поддержка объектно-ориентированного программирования;
- высокая скорость выполнения программного кода;
- наличие библиотек для работы с изображениями.

Первоочередной задачей является выбор языка программирования. Среди большого множества современных языков существует разделение на две группы - компилируемые и интерпретируемые, различающиеся методом исполнения кода. Для разрабатываемого ПС выбрана группа компилируемых языков, так как производительность программы находится в числе важнейших требований. Также язык должен быть объектно-ориентированным, что приводит к ограниченной группе популярных языков: C++, C#, Java, Visual Basic. Java использует особую среду JVM, позволяющую программам выполняться на машинах разных конфигураций, однако требует дополнительных вычислительных затрат, что делает этот язык нежелательным. C++, C# и Visual Basic объединены тем, что могут быть реализованы на одной платформе – Microsoft .NET. Она характеризуется высокой степенью интеграции различных технологий с помощью единого байт кода CIL (Common Intermediate Language) и библиотек DLL (Dynamic Link Library). Так, например, библиотека, написанная на языке Visual C++, может быть использована при написании программ на C# и Visual Basic. Это обеспечивает большую общую базу компонентов для повторного использования. Из трех языков C# наиболее прост в использовании и обеспечивает быструю реализацию объектно-ориентированных архитектур. По производительности он несколько уступает C++, однако этот недостаток может быть преодолен путем использования высокопроизводительных библиотек.

Еще одной характерной особенностью платформы .NET является привязанность к единой интегрированной среде разработки – Microsoft Visual Studio. Помимо редактора исходного кода, она предоставляет широкие

возможности для построения схем, анализа кода, управления версиями, визуального программирования и многих других современных методик.

В стандартном пакете библиотек и фреймворков платформы .NET существует один, приспособленный для создания оконных приложений. Windows Forms представляет собой событийно-ориентированный фреймворк. Большая часть времени своей работы приложение ждет действий пользователя, предоставляя для этого графический интерфейс – «форму». В результате какого-либо действия – «события» в терминах фреймворка – исполняется ответственный за его обработку код. Windows Forms предлагает эффективные средства визуального программирования, позволяющие реализовывать графический оконный интерфейс за короткое время с написанием минимального количества кода (для обработки событий).

Платформа .NET имеет ряд встроенных библиотек для работы с графическими данными, однако они ориентированы на удобство и универсальность использования, и поэтому являются недостаточно производительными для обработки больших объемов данных. Для этих целей существует популярная платформонезависимая библиотека OpenCV (Open source Computer Vision). Она состоит из наборов высокопроизводительных функций для обработки изображений в реальном времени. Библиотека написана на языке C++, однако посредством оберточных интерфейсов доступна для языков C#, Perl, Ch, Ruby, Java и Python. Подобной оберткой в случае C# является Emgu CV. Так, через высокоуровневый интерфейс осуществляется доступ к эффективным функциям для обработки графических и статистических (что важно для разрабатываемого ПС) данных.

4.2 Описание компонента синтеза панорамы

На аналитической модели приложения, представленной ранее на рисунке 3.1, видны основные элементы архитектуры и их зависимости друг от друга. Элемент «Приложение» - центр всех зависимостей, наиболее специфичный компонент программного средства. Исходящие из него зависимости имеют звездообразную топологию и почти не пересекаются. Это отражает уровни изоляции разных компонентов программы. При эффективном управлении зависимостями в архитектуре ее составные компоненты могут быть реализованы в виде модулей и библиотек, взаимозаменяемых и пригодных для повторного использования. В данной архитектуре можно четко выделить несколько изолированных групп элементов:

- компонент, отвечающий за синтез панорамы из набора изображений;
- компонент, управляющий файлами;
- окно приложения, обеспечивающее основной графический интерфейс;
- графический редактор;
- компонент, управляющий настройками программы.

Среди перечисленных наиболее сложным и значимым для программного средства является компонент синтеза панорамы, так как он реализует основную функцию программы. Ввиду комплексного характера процесса построения панорамы и большого количества используемых структур данных и алгоритмов их обработки, важными условиями получения эффективного результата являются гибкость и простота реализации. Поэтому компонент разрабатывается в наиболее абстрактном виде, со строгими связями между его составляющими и возможностью для расширения. В рамках объектно-ориентированного программирования этой цели служат интерфейсы. В сочетании с приемом проектирования «Фабрика», они позволяют разделять разрабатываемые модули на две составляющие: связи между абстрактными сущностями и независимые друг от друга реализации этих сущностей.

Основная часть компонент синтеза панорамы представлена библиотекой Panoramas, хранящей связный набор интерфейсов. Дополнительно присутствует ряд классов, реализующих простейшие структуры данных и определяющих порядок генерации результата. Доступ к ним по интерфейсам реализует «Фабрика» библиотеки. Все интерфейсы класса и их зависимости представлены на рисунке 4.1.

Далее следуют описания каждого модуля библиотеки.

4.2.1 Класс Stitcher

Класс Stitcher выполняет роль «фасада» библиотеки, принимая вызовы от приложения. Его интерфейс IStitcher устроен таким образом, чтобы приложение не знало о способе выполнения синтеза панорамы, т.е. не имело доступа к другим внутренним класса и интерфейсам библиотеки. Исключение представляет лишь интерфейс IRelationControl (см. пункт 4.2.11), являющийся средством влияния приложения на процесс синтеза. В иных случаях принимаемые и отдаваемые объектом класса Stitcher значения должны представлять стандартные или общедоступные структуры данных. Интерфейс IStitcher определяет два метода:

- Метод MatchBetween. Принимает имена изображений, участвующие в построении панорамы (они известны вызывающей метод программе, так передаются ею генератору панорамы в качестве исходных данных). Метод возвращает объект интерфейса IRelationControl, отображающий информацию о связи между изображениями и позволяющий редактировать ее в качестве настройки генерации. Вспомогательный метод, необязателен для нормального функционирования программы.

- Метод StitchAll. Не принимает параметров, возвращает представление панорамы в виде стандартного изображения (экземпляр класса Bitmap). Таким образом, заключает в себе главную функцию библиотеки.

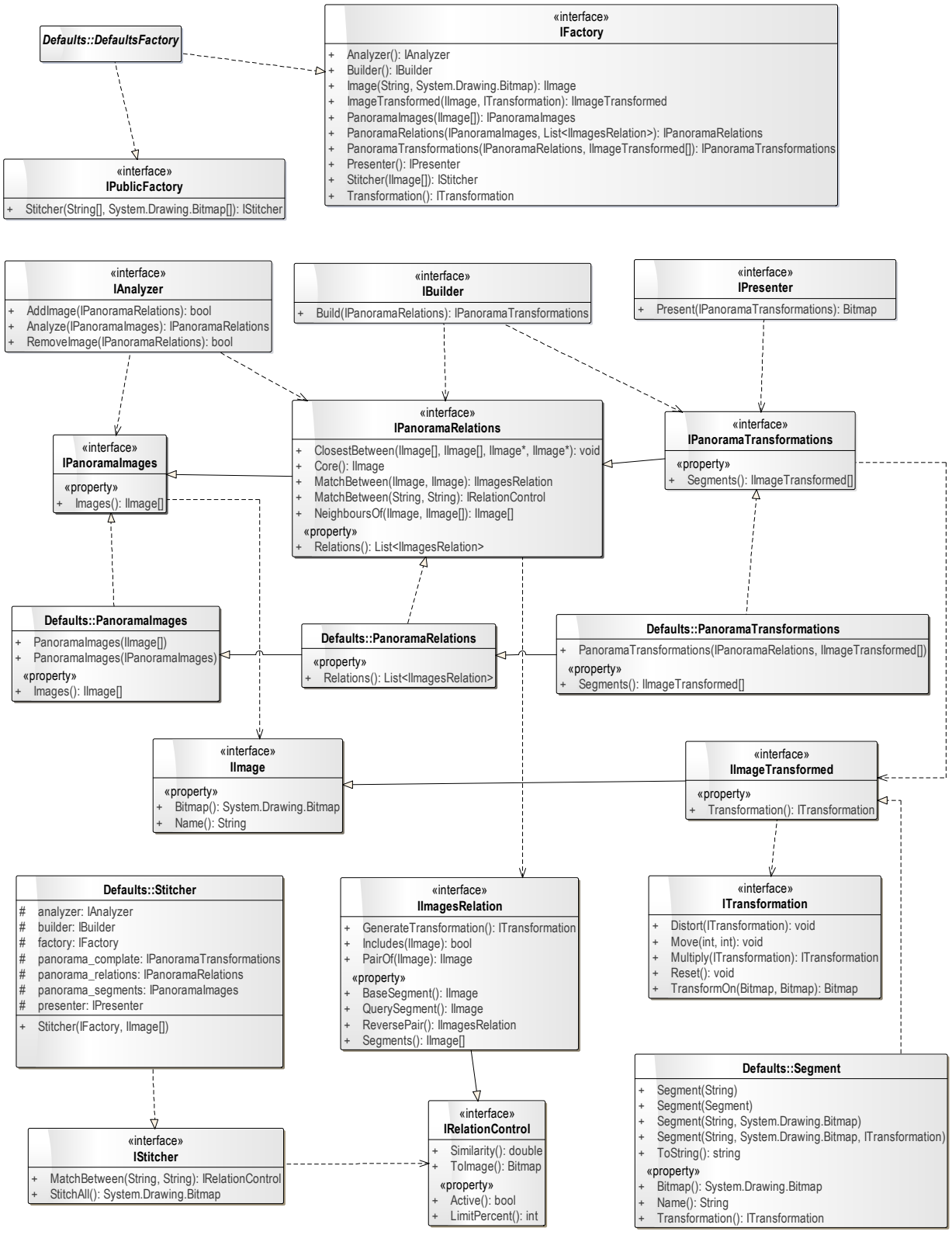

Рисунок 4.1 – Диаграмма классов библиотеки Panoramas

Интерфейс реализован классом Stitcher, доступным через стандартную фабрику DefaultFactory библиотеки Panoramas. Его атрибуты – основные компоненты библиотеки:

- объекты интерфейсов IAnalizer, IBuilder и IPresenter;
- представления панорамы в виде объектов IPanoramaImages, IPanoramaRelations и IPanoramaTransformations;
- объект фабрики IFactory.

Конструктор класса принимает экземпляр фабрики и массив изображений. Последние принадлежат нестандартному интерфейсу IImage, за реализацию которого отвечает фабрика (экземпляр Stitcher сам конструируется фабрикой, а не внешним приложением). По набору атрибутов класса прослеживается череда преобразований, через которые проходят данные при выполнении процесса склейки панорамы: анализ, сборка и представление результата. Через интерфейс класса доступны данные, являющиеся результатом анализа входных изображений, следовательно, нет причин выполнять анализ в отдельном от конструктора методе. Так, объекту класса Stitcher в дальнейшем всегда доступен экземпляр IPanoramaRelations, представляющий результаты анализа. При вызове MatchBetween осуществляется обращение к этому экземпляру. В результате обращения данные могут быть изменены, что следует учитывать при выполнении дальнейших этапов процесса синтеза панорамы.

Вызов метода StitchAll продолжает процесс синтеза, используя результаты анализа для сборки панорамы и ее представления в виде Bitmap. Переиспользовать результаты синтеза нельзя, так как данные могут быть изменены приложением через интерфейс IRelationControl. Если судить по фактическому набору методов класса Stitcher, обязательными атрибутами для него являются лишь объекты IBuilder, IPresenter и IPanoramaRelations. Тем не менее, наличие полного набора атрибутов наглядно отражает логику работы класса, а также позволяет менять ее с помощью расширения класса.

4.2.2 Класс Segment

Segment - второй фундаментальный класс библиотеки, представляющий сегмент панорамы. При инициализации библиотеки любой сегмент представляет собой стандартное изображение класса Bitmap и его имя (идентификатор). С течением процесса генерации определяется место каждого сегмента на панораме и затем используется при сборке панорамы в виде Bitmap. То есть, начальные данные дополняются новыми, доступными на поздних этапах процесса. Поэтому класс Segment имеет два интерфейса: IImage и IImageTransformed. Первый определяет доступ к строковому имени Name и к изображению Bitmap. Второй расширяет IImage объектом интерфейса ITransformation, определяющим преобразование над изображением для его помещения на панораму. Реализация ограниченного и расширенного интерфейсов одним классом в данном случае оправдана малым

объемом его содержания и высокой степенью изолированности (о существовании класса Stitcher знает лишь реализация DefaultFactory фабрики IFactory, все остальные модули используют его через интерфейсы).

4.2.3 Класс PanoramaImages

Класс отвечает за хранение исходных данных процедуры построения панорамы – простых изображений. Как и Segment, класс доступен из фабрики DefaultFactory через его интерфейс IPanoramaImages. Данный интерфейс описывает лишь одно свойство – массив объектов интерфейса IImage. Объект класса PanoramaImages осуществляет инициализацию и хранение этого массива. Класс предельно прост, его роль – контроль доступа к коллекции исходных изображений. В контексте процесса синтеза его можно считать первой формой панорамы. В дальнейшем эта форма будет расширена.

4.2.4 Класс PanoramaRelations

PanoramaRelations отвечает за хранение и доступ к результатам анализа исходных изображений. Его интерфейс IPanoramaRelations обусловлен вариантами дальнейшего использования этих результатов при построении панорамы. Он расширяет интерфейс IPanoramaImages и включает в себя:

- Список объектов Relations интерфейса IImagesRelation. Это вспомогательный элемент интерфейса, обеспечивающий его легкую расширяемость.
- Метод Core, возвращающий одно из изображений IImage, играющее роль «ядра» панорамы по критерию схожести сегментов между собой.
- Две перегрузки метода MatchBetween. Одна из них принимает в качестве параметров имена двух изображений и возвращает объект интерфейса IRelationControl, описывающий связь между ними. Такой вариант метода используется элементов Stitcher для предоставления дополнительного интерфейса пользователю библиотеки. Второй вариант метода принимает два объекта изображений IImage и возвращает экземпляр IImagesRelation. Функция метода та же, однако предоставляемый им объект предназначен для использования внутри библиотеки.
- Метод NeighboursOf. Принимает изображение IImage и массив изображений, из которых выбираются наиболее схожие с данным. При отсутствии массива выбор осуществляется из всего множества сегментов панорамы.
- Метод ClosestBetween. Имеет сложный набор параметров: два набора изображений и две ссылки IImage для возврата результатов. Предназначен для нахождения пары наиболее схожих изображений из двух предоставленных множества.

Класс PanoramaRelations представляет результаты операции анализа над исходной коллекцией изображений, и поэтому является наследником класса PanoramaImages, расширяя его дополнительными данными (массив объектов IImagesRelation) и операциями над ними. Также наследование обеспечивает

беспрепятственный доступ к набору исходных данных. В контексте процесса синтеза, PanoramaRelations – второе представление панорамы, дополненное связями между ее сегментами.

Выбор «ядра» панорамы в методе Core осуществляется путем поиска изображения с наименьшим суммарным расстоянием от него до других. Подобные операции легко реализуются на языке C# с помощью стандартной библиотеки LINQ. Расстояние для каждого изображение определяется с помощью закрытого метода distancesFor. Схожим образом реализованы методы MatchBetween: последовательным перебором находится экземпляр связи, соответствующий переданным параметрам.

Метод ClosestBetween с помощью операций LINQ находит все связи, к которым принадлежат изображения первого из переданных ему наборов, затем выбирает из них связи со вторым набором и сортирует в порядке увеличения схожести. Первый объект в этой коллекции – искомая связь между двумя наборами изображений.

Метод NeighboursOf выбирает те исходные изображения, самым схожим для которых является данное.

4.2.5 Класс PanoramaTransformations

PanoramaTransformations отвечает за хранение и доступ к результатам построения панорамы. Его интерфейс IPanoramaTransformations включает лишь массив объектов Segments интерфейса IImageTransformed (см. пункт 4.2.2), то есть сегменты панорамы с определенными расположениями на ней. Объект класса инициализирует свой массив и предоставляет доступ к сегментам. Он расширяет PanoramaRelations, представляя собой последнюю итерацию данных панорамы, готовую к представлению.

4.2.6 Интерфейс IImagesRelation

Интерфейс представляет связь между двумя изображениями, предназначенную для использования при построении панорамы. Элементы интерфейса:

- Экземпляры BaseImage и QueryImage интерфейса IImage. Представляют два изображения, отношение между которыми описывает объект интерфейса.
- Экземпляр ReversePair того же интерфейса IImagersRelation. Ссылается на объект, описывающий обратную связь изображений. Такое разделение по направлению связи необходимо ввиду неравнозначности их отношений друг к другу.
- Метод Includes. Принимает объект IImage и возвращает логическое значение, говорящее, состоит ли изображение в данной связи.
- Метод PairOf. Принимает объект IImage и возвращает связанный с ним данным соотношением.
- Метод GenerateTransformation. Возвращает объект интерфейса ITransformation, как средство реализации связи двух изображений.

4.2.7 Интерфейс ITransformation

Описывает преобразование какого-либо изображения. В контексте процедуры синтеза панорамы, является способом размещения отдельного сегмента на ней. Методы интерфейса:

- Метод Reset. Служит для установки преобразования в свое изначальное состояние. Изначально объект интерфейса должен описывать такое преобразование, которое не изменяет положение изображения.

- Метод Move. Принимает две целочисленные величины, определяющие смещение изображения в пределах плоскости двухмерного пространства.

- Метод Multiply. Принимает другое преобразование и возвращает результат его объединения с текущим. Метод необходим для работы с последовательностями преобразований.

- Метод Distort. Принимает другое преобразование и объединяет текущее с ним.

- Метод TransformOn. Применяет преобразование к данному изображению стандартного класса Bitmap и размещает его на другом. Ожидаемый результат – второй параметр (холст) с нанесенным на него изображением.

4.2.8 Интерфейс IAnalyzer

IAnalyzer отвечает за процедуру анализа исходных изображений и представление его результатов в виде отношений. Включает метод Analyze, который принимает набор изображений (IPanoramaImages) и возвращает коллекцию результатов анализа (IPanoramaRelations). Модуль не имеет стандартной реализации в библиотеке Panoramas и предназначен для использования при расширении библиотеки. Причина этого – отсутствие стандартного алгоритма или метода анализа.

4.2.9 Интерфейс IBuilder

IBuilder отвечает за процедуру построения панорамы, выражаемую в определении положения каждого из связанных между собой изображений. Включает метод Build, принимающий набор изображений и их связей (IPanoramaRelations) и возвращающий набор позиционированных изображений (IPanoramaTransformations).

4.2.10 Интерфейс IPresenter

IPresenter – последний обработчик данных панорамы, представляющий ее в доступном для приложения варианте – в виде экземпляра класса Bitmap. Аналогично IAnalyzer и IBuilder, не имеет стандартной реализации ввиду отсутствия универсального способа представления сегментов панорамы на одном изображении. Интерфейс состоит из одного метода – Present, принимающего набор позиционированных изображений (IPanoramaTransformations) и возвращающий объект Bitmap.

4.2.11 Интерфейс IRelationControl

Представляет дополнение к «фасаду» библиотеки – IStitcher - для управления настройками генерации панорамы и их наглядного представления. В качестве объекта настроек выступает связь между двумя исходными изображениями. Состав интерфейса:

- Метод ToImage. Возвращает представление связи между изображениями в виде экземпляра стандартного класса Bitmap.
- Метод Similarity. Возвращает целочисленное значение используемой метрики схожести двух изображений. Значение должно отображать результаты изменения настроек приложением.
- Целочисленное значение LimitPercent. Предоставляет доступ к параметру, описывающему в процентах некоторую характеристику связи (в конкретном случае – полноту используемого набора особых точек).
- Флаг Active. Простейший элемент интерфейса, позволяет исключать связь из использования при построении панорамы.

Реализация интерфейса относится к конкретным методам анализа и сборки панорамы, поэтому в библиотеке отсутствует.

4.2.12 Класс DefaultFactory

DefaultFactory – фабрика, используемая для расширения библиотеки Panoramas и реализации ее компонентов. Фабрика доступна внутренним модулям библиотеки через интерфейс IFactory, включающий методы:

- Метод Image. Возвращает объект интерфейса IImage, полученный на основе данного строкового имени и изображения Bitmap.
- Метод Transformation. Возвращает объект интерфейса ITransformation (в исходном состоянии).
- Метод ImageTransformed. Возвращает объект IImageTransformed, образованный данным объектом IImage и описывающим его положение объектом ITransformation.
- Метод PanoramaImages. Принимает массив исходных изображений и IImage возвращает использующий их объект интерфейса IPanoramaImages.
- Метод PanoramaRelations. Принимает контейнер изображений IPanoramaImages и список описывающих их связей IImagesRelation. Возвращает объект представления связных изображений IPanoramaRelations.
- Метод PanoramaTransformations. Принимает контейнер IPanoramaRelations и массив позиционированных изображений IImageTransformed. Возвращает контейнер позиционированных изображений IPanoramaTransformations.
- Метод Stitcher. Принимает массив изображений IImage и возвращает экземпляр IStitcher.
- Методы Analyzer, Builder и Presenter. Не принимают параметров, возвращают экземпляры интерфейсов IAnalyzer, IBuilder и IPresenter соответственно.

Класс DefaultFactory включен в состав библиотеки Panoramas, однако реализации некоторых элементов его интерфейса в ней не определены. В языке C# существует механизм частичной реализации интерфейсов – абстрактные классы. Подобные классы не имеют конструкторов и требуют своего расширения. Как интерфейсы, они способы определять методы для расширения (абстрактные), однако также могут самостоятельно реализовывать другие методы. Следовательно, целесообразно сделать DefaultFactory абстрактным классом с реализацией методов Image, ImageTransformed, PanoramaImages, PanoramaRelations, PanoramaTransformations и Stitcher, так как требуемые ими классы присутствуют в библиотеке (описаны выше). Реализации представляют собой инициализацию объектов этих классов с передачей им параметров. Остальные методы интерфейса – IAnalyzer, IBuilder, IPresenter и ITransformation – описаны классом как абстрактные, т.е. требующие реализации при наследовании.

Описанный интерфейс IFactory предназначен для использования внутренними модулями библиотеки и их расширениями. Предоставление его объекта приложению было бы нарушением принципа изолированности реализаций. Поэтому, определен еще один интерфейс фабрики – IPublicFactory. Он предназначен для внешнего использования, и содержит только один метод – Stitcher. Параметры метода – массив строковых имен и массив изображений Bitmap. Возвращаемое значение – объект интерфейса IStitcher. Таким образом, интерфейс IPublicFactory позволяет ограничить использование фабрики инициализацией главного интерфейса библиотеки. Реализация метода в классе DefaultFactory проверяет соответствие переданных массивов по их размерам, конструирует на их основе объекты IImage и использует метод Stitcher внутреннего интерфейса IFactory для инициализации IStitcher.

5 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Тестирование программного обеспечения – процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта. Обычным разделением тестирования являются уровни:

- компонентное, или модульное (unit testing);
- интеграционное (integration testing);
- системное (system testing).

Они различаются по объему тестируемого функционала и степени автоматизации. Так, модульные тесты наиболее просты для реализации, и поэтому используются чаще остальных. Системное тестирование – наиболее сложное, проверяет соответствие ПС самым общим требованиям. Все три уровня дополняют друг друга и используются совместно.

5.1 Модульное тестирование программного средства

Модульные тесты акцентируют внимание на отдельных модулях (классах) программы. Обычным методом реализации является метод черного ящика, при котором реализация модуля не принимается во внимание, а осуществляется проверка его способности выполнять свои функции – т.е. соответствия скрытой реализации общедоступному интерфейсу.

Модульные тесты применялись при разработке библиотеки синтеза панорам. Это позволило контролировать последствия любых практически изменений в коде, уменьшая количество дефектов. Также тесты косвенно обеспечили разделенность модулей и документирование их функций.

5.2 Интеграционное тестирование программного средства

Интеграционное тестирование состоит в выявлении дефектов во взаимодействии группы модулей. Обычно такие тесты основаны на детализированных функциональных требованиях к ПС, и поэтому употребляются как синоним «функциональных». Структурно интеграционные тесты разделены на сценарии (test scenario) и тест-кейсы (test case). Тест-кейс определяет набор входных данных для программы или компонента, условия и результаты их обработки, таким образом описывая конкретное свойство или поведение ПС.

Для разрабатываемого ПС был создан ряд тестовых сценариев и тест-кейсов, определяющих степень его функциональной готовности. Далее ведется их перечень.

5.2.1 Тестовый сценарий «Управление окном» включает действия, которые пользователь может совершить, используя окно приложение как обычное окно. Тест-кейсы приведены в таблице 5.1.

Таблица 5.1 – Описание тест-кейсов сценария «Управление окном»

Идентификатор	Последовательность действий	Результаты действий
app.start	1. Запуск приложения	1. Появление окна приложения. Фокус на кнопке «Add images..». Кнопка окна для разворачивания заблокирована. Изменение размера окна недоступно. Сворачивание окна доступно.
app.close	1. Запуск приложения 2. Нажатие на кнопку закрытия окна приложения	1. Появление окна приложения 2. Завершение работы приложения. Исчезновение окна

5.2.2 Тестовый сценарий «Загрузка файлов» включает загрузку файлов изображений в приложение. Тест-кейсы приведены в табл. 5.2.

Таблица 5.2 – Описание тест-кейсов сценария «Загрузка файлов»

Идентификатор	Последовательность действий	Результаты действий
load.none	Запуск приложения 1. Нажать кнопку «Add images..» 2. Нажать кнопку «Cancel» диалога	Кнопки «Next», «Remove selected» и «Remove all» заблокированы. Область просмотра файлов пуста. 1. Появление стандартного диалога выбора файлов 2. Закрытие диалога. Кнопки «Next», «Remove selected» и «Remove all» заблокированы. Область просмотра файлов пуста.
load.one	1. Нажать кнопку «Add images..» 2. Выбрать одно изображение JPEG и нажать «Open»	1. Появление стандартного диалога выбора файлов 2. Закрытие диалога. Кнопки «Next», «Remove selected» заблокированы. Кнопка «Remove all» разблокирована. Появление иконки выбранного изображения в области просмотра файлов. Наведение фокуса на кнопку «Next».

Продолжение таблицы 5.2

load.incorrect	1. Нажать кнопку «Add images..» 2. Выбрать не изображение (аудиофайл) и нажать «Open»	1. Появление стандартного диалога выбора файлов 2. Закрытие диалога. Появление окошка с сообщением о неверном формате изображения. Область просмотра файлов пуста.
load.images	1. Нажать кнопку «Add images..» 2. Выбрать два изображения JPEG и нажать «Open»	1. Появление стандартного диалога выбора файлов 2. Закрытие диалога. Кнопки «Next» и «Remove all» разблокированы. Кнопка «Remove selected» заблокирована. Появление иконок обоих изображений в области просмотра файлов. Наведение фокуса на кнопку «Next».
load.more	1. Нажать кнопку «Add images..» 2. Выбрать два изображения JPEG и нажать «Open» 3. Нажать кнопку «Add images..» 4. Выбрать одно изображение JPEG из другой директории и нажать «Open»	1. Появление стандартного диалога выбора файлов 2. Появление иконок обоих изображений в области просмотра файлов 3. Появление стандартного диалога выбора файлов 4. Добавление иконки нового изображения к имеющимся в области просмотра файлов. Кнопки «Next» и «Remove all» разблокированы. Кнопка «Remove selected» заблокирована.

5.2.3 Тестовый сценарий «Управление файлами» и последующие будут использоваться тест-кейс load.images в качестве действия. Тест-кейсы представлены в таблице 5.3.

Таблица 5.3 – Описание тест-кейсов сценария «Управление файлами»

Идентификатор	Последовательность действий	Результаты действий
files.clear	1. Загрузить изображения 2. Нажать кнопку «Remove all»	1. Отображение файлов в области просмотра Кнопка «Remove all» разблокирована. 2. Кнопки «Next», «Remove selected» и «Remove all» заблокированы. Область просмотра файлов пуста

Продолжение таблицы 5.3

files.remove	1. Загрузить изображения 2. Выбрать одно из них на области просмотра файлов 3. Нажать кнопку «Remove selected»	1. Отображение файлов в области просмотра. Кнопка «Remove selected» заблокирована 2. Кнопка «Remove selected» разблокирована 3. Исчезновение выбранного изображения из области просмотра. Кнопки «Remove selected» и «Next» заблокированы

5.2.4 Тестовый сценарий «Генерация панорамы» описывает единичный процесс генерации, т.е. критический вариант выполнения программы. Тест-кейсы представлены в таблицу 5.4.

Таблицы 5.4 – Описание тест-кейсов сценария «Генерация панорамы»

Идентификатор	Последовательность действий	Результаты действий
panorama.create	1. Загрузить изображения 2. Нажать кнопку «Next»	1. Изображения добавлены на область просмотра файлов 2. Выполнение операции (ожидание). Переход на страницу «Stitching» окна. Отображение панорамного изображения на основной рабочей области страницы.
panorama.save	1. Загрузить изображения 2. Нажать кнопку «Next» 3. Нажать кнопку «Save as..» 4. Выбрать расположение и имя файла и нажать «Save»	1. Изображения добавлены на область просмотра файлов 2. Выполнение операции (ожидание). Переход на страницу «Stitching». Фокус на кнопке «Save as..» 3. Появление диалога сохранения изображения 4. Закрытие диалога. Нет изменений на окне.

5.2.5 Тестовый сценарий «Настройка генерации панорамы» и последующие используют тест-кейс «panorama.create» как действие «Сгенерировать панораму из нескольких изображений», так как оно необходимо для выполнения повторной генерации. Тест-кейсы приведены в таблицу 5.5.

Таблица 5.5 – Описание тест-кейсов сценария «Настройка генерации панорамы»

Идентификатор	Последовательность действий	Результаты действий
matching.visit	1. Сгенерировать панораму из нескольких изображений 2. Нажать кнопку «Back»	1. Переход на страницу «Stitching». Отображение панорамы. 2. Переход на страницу «Matching». Фокус на кнопке «Next».
matching.lists	1. Сгенерировать панораму из нескольких изображений 2. Нажать кнопку «Back» 3. Выбрать другой файл в нижнем списке 4. Выбрать другой файл в верхнем списке 5. Выбрать в верхнем списке файл, выбранный в нижнем списке	1. Переход на страницу «Stitching». Отображение панорамы. 2. Переход на страницу «Matching». Совместное отображение пары изображений и их общих точек на рабочей области окна 3. Изменение изображения на главной рабочей области. Смена в нем левого изображения. 4. Изменение изображения на главной рабочей области. Смена правого изображения. 5. Изменение изображения на главной рабочей области. Изменение содержания нижнего списка (без выбранного файла)
matching.similarity	1. Сгенерировать панораму из нескольких изображений 2. Нажать кнопку «Back» 3. Изменить положение ползунка «Similarity» до минимума 4. Изменить положение ползунка «Similarity» до максимума 5. Установить флаг «Similar?» 6. Нажать кнопку «Next»	1. Переход на страницу «Stitching». Отображение панорамы. 2. Переход на страницу «Matching». 3. Прекращение отображения всех связей между изображениями. Сброс флага «Similar?» 4. Отображение связей изображений. Состояние флага «Similar?» не изменяется 5. Флаг «Similar?» установлен 6. Переход на страницу «Stitching». Отображение панорамы

Продолжение таблицы 5.5

matching.similar	1. Сгенерировать панораму из нескольких изображений 2. Нажать кнопку «Back» 3. Сбросить флаг «Similarity» для всех пар изображений 4. Нажать кнопку «Next»	1. Переход на страницу «Stitching». Отображение панорамы. 2. Переход на страницу «Matching». 3. Флаги «Similar?» сброшены 4. Сообщение о недостатке данных для генерации. Страница «Matching».

5.2.6 Тестовый сценарий «Изменение исходных изображений» описан в таблице 5.6.

Таблица 5.6 – Описание тест-кейсов сценария «Изменение исходных изображений»

Идентификатор	Последовательность действий	Результаты действий
segments.more	1. Сгенерировать панораму из нескольких изображений 2. Нажать на вкладку «Segments» 3. Нажать кнопку «Add images..» 4. Выбрать несколько новых изображений и нажать «Open» 5. Нажать кнопку «Next» 6. Нажать кнопку «Next»	1. Переход на страницу «Stitching». Отображение панорамы. 2. Переход на страницу «Segments» 3. Появление стандартного диалога выбора файлов 4. Добавление новых изображений в область просмотра файлов 5. Переход на страницу «Matching». Наличие имен новых файлов в списках файлов. 6. Переход на страницу «Stitching». Отображение измененной панорамы.

Продолжение таблицы 5.6

segments.less	1. Нажать кнопку «Add images..» 2. Выбрать три изображения JPEG и нажать «Open» 3. Нажать кнопку «Next» 4. Нажать на вкладку «Segments» 5. Выбрать одно из изображений в области просмотра файлов 6. Нажать «Remove selected» 7. Нажать кнопку «Next» 8. Нажать кнопку «Next»	1. Появление стандартного диалога открытия файлов 2. Появление файлов в области просмотра 3. Выполнение операции (ожидание). Переход на страницу «Stitching». Отображение панорамы 4. Переход на страницу «Segments» 5. Кнопка «Remove selected» разблокирована 6. Исчезновение выбранного изображения из области просмотра 7. Переход на страницу «Matching». Отсутствие удаленного изображения в обоих списках файлов 8. Переход на страницу «Stitching». Отображение измененной панорамы

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ПРОГРАММНОГО СРЕДСТВА

Для корректной работы приложения требуется следующая минимальная конфигурация компьютера:

- процессор разрядностью 32 (86) бит;
- ОС Windows 7 или Windows 8;
- не менее 1 Гб оперативной памяти;
- подключенная клавиатура или компьютерная мышь.

Для начала работы с программой необходимо запустить ее посредством файла «ImageStitcher.exe». После этого должно появиться начальное окно (см. рисунок 6.1). На нем имеется ряд управляющим кнопок, но все они заблокированы, кроме одной – «Add images..». При нажатии на кнопку появится стандартный диалог открытия файлов, с установленным фильтром для изображений. Диалог позволяет выбирать несколько изображений из одной директории.

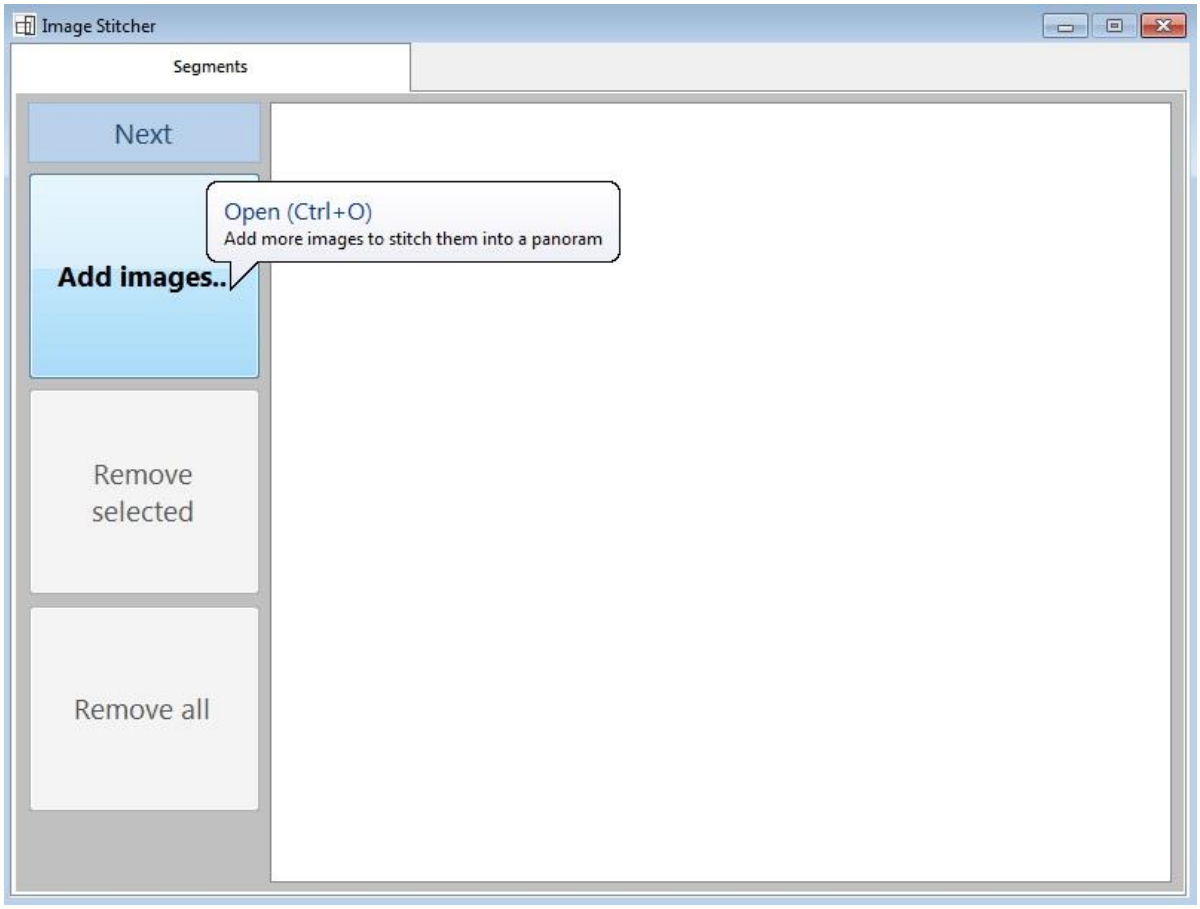

Рисунок 6.1 – Начальный вид окна приложения

Существуют общепринятые рекомендации по подбору изображений для их объединения в панораму. Все они должны быть связаны между собой перекрывающимися областями (когда часть изображения присутствует, в

идентичном или схожем виде, на другом изображении). Если используемые изображения – фотографии, то они должны иметь как можно более близкие значения баланса белого. В идеальном случае – это фотографии удаленной сцены, сделанные из одной точки (так минимизируются искажения).

При загрузке в приложение хотя бы одного изображения, оно появится в области просмотра файлов (в виде значка и имени файла). Станет доступной кнопка «Remove all», очищающая эту область. При клике на файл в рабочей области разблокируется кнопка «Remove selected». Последующая загрузка изображений посредством «Add images..» добавит их к имеющимся на рабочей области. Если среди загружаемых изображений некоторые уже присутствуют на в области просмотра, они будут проигнорированы. Когда общее количество загруженных файлов достигает двух, разблокируется кнопка «Next» в левом верхнем углу окна (см. рисунок 6.2), инициирующая генерацию панорамы.

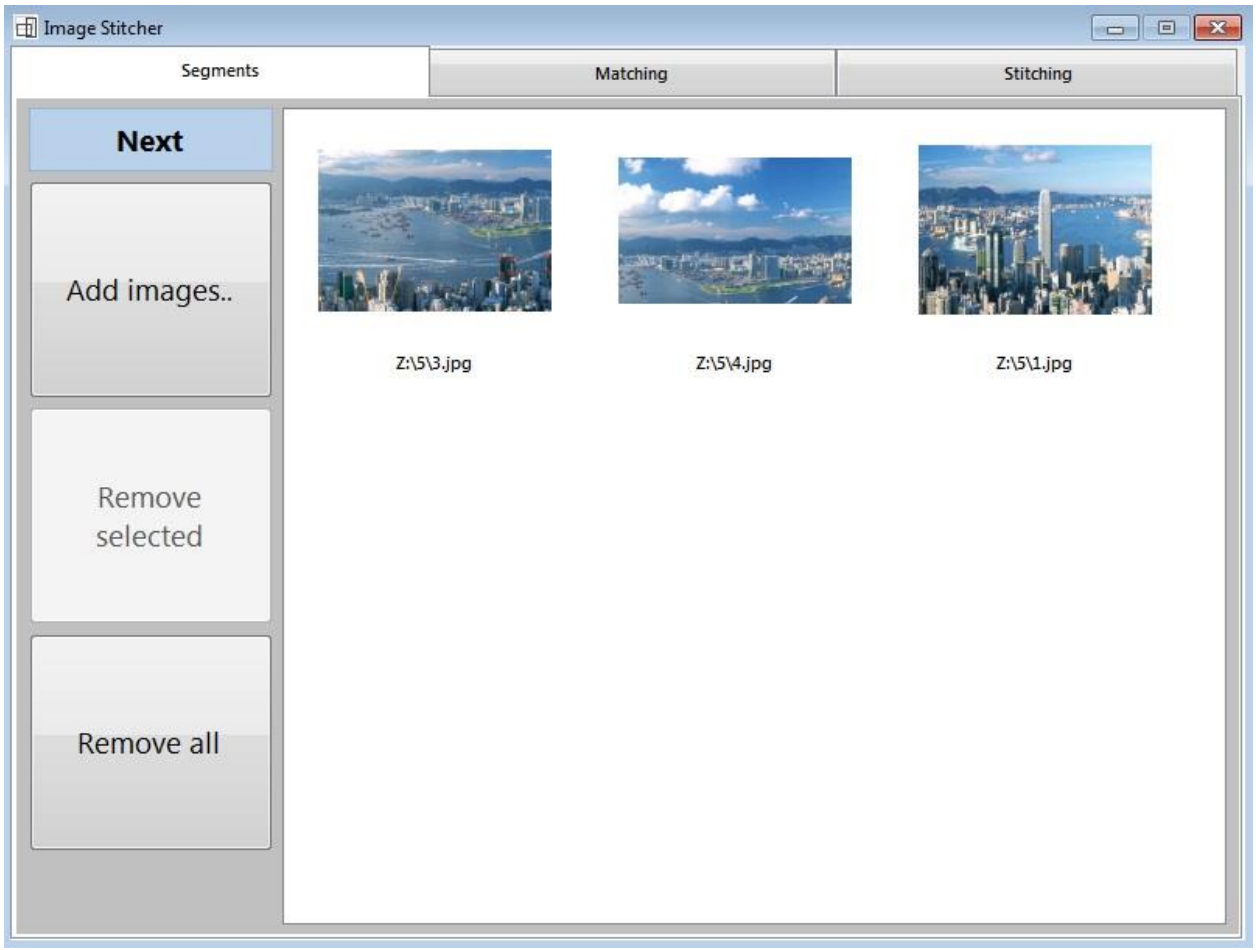

Рисунок 6.2 – Окно приложения после загрузки нескольких файлов

Процесс генерации может занять некоторое время, в зависимости от размера входных изображений. По его окончании рабочая область окна сменится новой страницей «Stitching», с выводом созданной панорамы (см. рисунок 6.3). Элемент отображения панорамы позволяет масштабировать и изменять размеры окна просмотра. Изображение может быть сохранено при

нажатии на «Save as..» (или комбинацией клавиш Ctrl+S). Сохранена будет та часть панорамы, которая просматривается в данный момент. Таким образом обеспечивается возможность обрезки неровных краев.

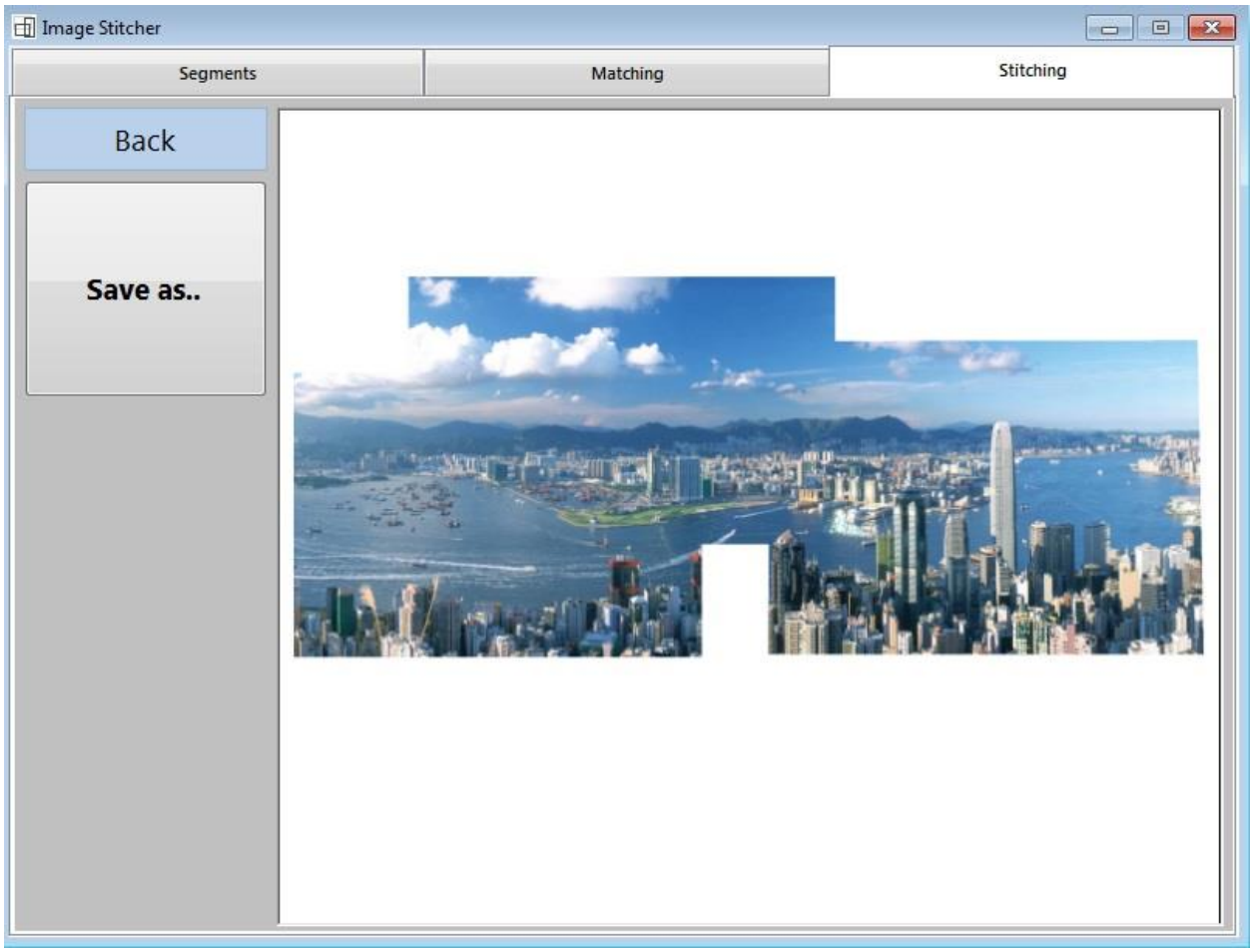

Рисунок 6.3 – Отображение сгенерированной панорамы

На случай, если на изображении будут присутствовать значительные дефекты от склейки сегментов панорамы, предусмотрена возможность повлиять на этот процесс. Со страницы «Stitching» доступны навигационные ссылки на страницы «Segments» и «Matching». При переходе на страницу «Matching» (это может быть сделано также нажатием на кнопку «Back»), будет представлены пары сегментов и отображены связи между ними (см. рисунок 6.4). Есть возможность изменять количество используемых связей путем перемещения ползунка «Similarity» внизу окна. Также доступен флаг «Similar?», напрямую определяющий, будут ли два изображения «склеены» на панораме. В случае сброса флага, изображения будут склеены с другими, наиболее схожими с ними (если их флаги «Similar?» еще установлены). Выбор сегментов для сравнения осуществляется с помощью двух списков, условно – «главного» и «подчиненного». После изменения связей между сегментами панорамы, можно перезапустить генерацию путем нажатия кнопки «Next».

Изменение связей между изображениями может исправить дефекты, заключающиеся в неверном порядке или в неточности их склейки. Однако в некоторых случаях изображения не могут быть объединены в панораму. Если дефективное изображение опознано на результирующей панораме, оно может быть удалено из набора сегментов и, возможно, заменено новым. Для этого имеется возможность вернуться на страницу загрузки файлов (кликом по вкладке «Segments» или нажатием кнопки «Back»). После редактирования набора файлов генерация может быть выполнена повторна путем последовательного нажатия кнопок «Next» (сперва для перехода на страницу «Matching», замет – на «Stitching»). Эти же действия могут осуществляться для полной замены набора файлов и генерации другой панорамы.

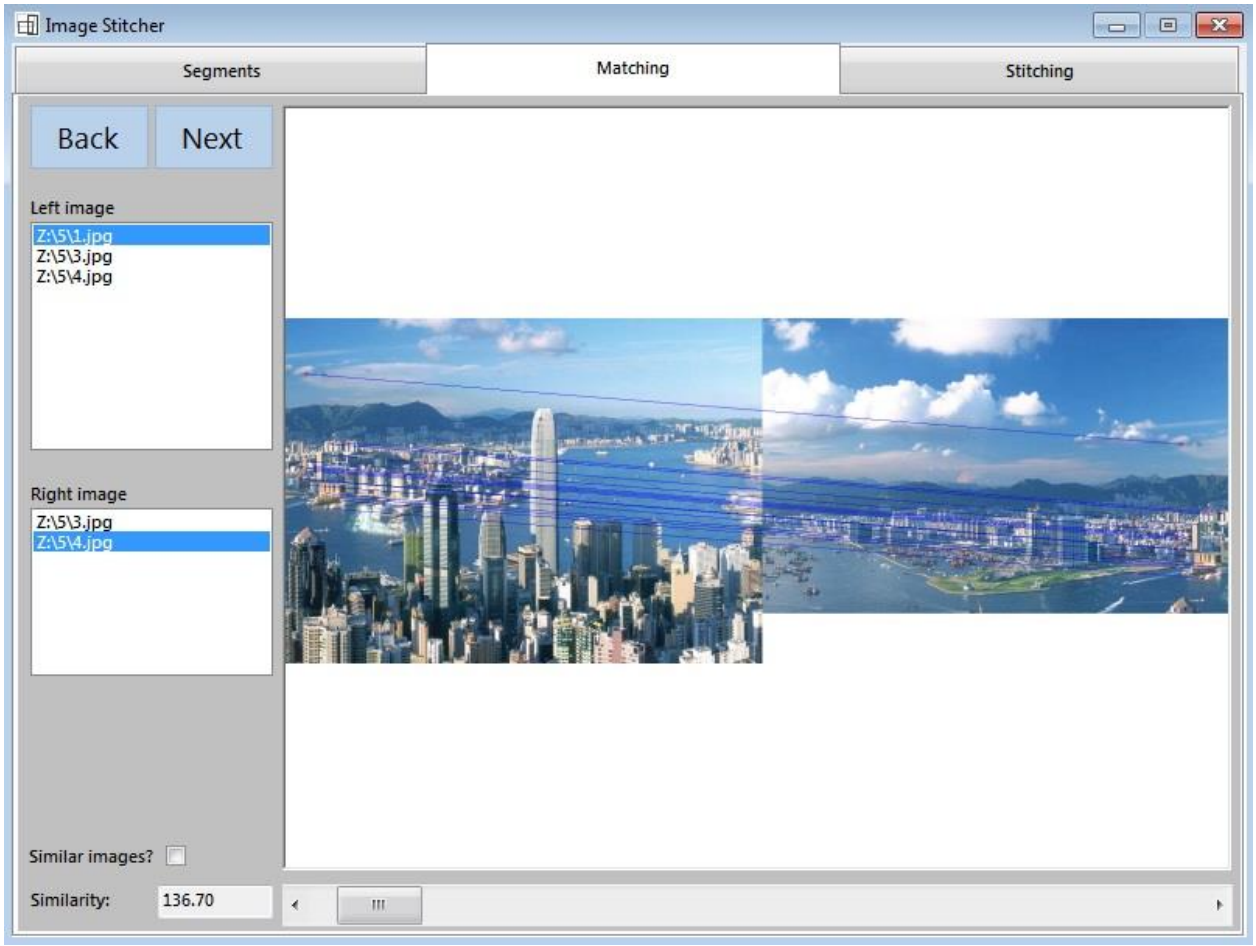

Рисунок 6.4 – Отображение настроек генерации панорамы

Завершение работы с приложением осуществляется путем закрытия его окна (кликом по стандартной кнопке в правом верхнем углу).

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА СИНТЕЗА ПАНОРАМНЫХ ИЗОБРАЖЕНИЙ

7.1 Функции, назначение и потенциальные пользователи ПС

Задача программного средства - генерация цифровых панорамных изображений путем автоматизированной обработки набора некоторых исходных графических изображений. ПС предназначено для неограниченного круга пользователей и свободной продажи на рынке информационных технологий. Ранее существующий набор программных решений весьма ограничен, качественно и количественно, особенно с учетом высокой сложности реализуемых процедур обработки. Подобные процедуры лишь в малой мере поддаются ручной реализации (с помощью программ - графических редакторов) и в общем случае требуют значительных трудозатрат. Автоматизация позволяет существенно снизить данный параметр, оставляя за пользователем лишь задание параметров процедуры или простые манипуляции в специальном графическом интерфейсе.

К основным функциональным требованиям относятся следующие:

- загрузка набора исходных изображений;
- генерация панорамы на основе набора исходных изображений;
- графическое представление панорамы с возможностью управления и изменения параметров обзора;
- возможность редактирования панорамы;
- сохранение и загрузка панорамы из файла на диске.

Особенностью программного средства является гибкость настройки и управления процессом генерации панорамы.

Экономическая целесообразность инвестиций в разработку и использование программного средства выявляется на основе расчета и оценки следующих показателей:

- чистый дисконтированный доход;
- рентабельность инвестиций;
- срок окупаемости инвестиций.

7.2 Расчет затрат на разработку и реализацию ПС

Расчет затрат на разработку ПС производится с использованием следующих статей расходов:

- затраты на основную заработную плату разработчиков;
- затраты на дополнительную заработную плату разработчиков;
- отчисления на социальные нужны;
- прочие расходы.

Расчет величины основной заработной платы разработчиков осуществляется по формуле:

$$З_о = \sum_i^n T_{чi} t_i, \quad (7.1)$$

где n – количество исполнителей, занятых в разработке ПС;

$T_{чi}$ – часовая заработная плата i-го исполнителя, млн. р.;

t_i – трудоемкость работ, производимых i-м исполнителем, ч.

Для инженера-программиста квалификационный разряд – 13 (тарифный коэффициент 3.04). При месячной ставке первого разряда 2 млн. р., месячная ставка каждого исполнителя равна:

$$T_{чi} = 2 * 3{,}04 = 6{,}08 \text{ млн. р.}$$

Время на разработку проекта – 4 месяца, количество разработчиков - 2. Таким образом,

$$З_о = 2 * 6{,}08 * 4 = 48{,}64 \text{ млн. р.}$$

Затраты на дополнительную заработную плату включают выплаты, предусмотренные действующим трудовым законодательством, и определяется по формуле:

$$З_д = \frac{З_о Н_д}{100}, \quad (7.2)$$

где $З_о$ – затраты на основную заработную плату с учетом премии, млн. р.;

$Н_д$ – норматив дополнительной заработной платы (20%).

$$З_д = \frac{48{,}64*20}{100} = 9.728 \text{ млн. р.}$$

Отчисления на социальные нужны (фонд социальной защиты и обязательное страхование) определяются в соответствии с действующими законодательными актами по формуле:

$$З_{соц} = \frac{(З_о+З_д)Н_{соц}}{100}, \quad (7.3)$$

где $Н_{соц}$ – норматив отчислений на социальные нужды (34,6%).

$$З_{соц} = \frac{(48{,}64+9{,}728)*34{,}6}{100} = \frac{58{,}368*34{,}6}{100} = 20{,}196 \text{ млн. р.}$$

Расчет прочих затрат осуществляется в виде расчета процентов от затрат на основную заработную плату команды разработчиков с учетом премии по формуле:

$$З_{пз} = \frac{З_о Н_{пз}}{100}, \quad (7.4)$$

где $Н_{пз}$ – норматив прочих затрат (принят равным 110%).

$$З_{пз} = \frac{48{,}64 * 110}{100} = 53.504 \text{ млн. р.}$$

Результаты расчетов приведены в таблице 7.1.

Таблица 7.1 – Затраты на разработку ПС

Статья затрат	Сумма, млн. р.
Основная заработная плата разработчиков	48,640
Дополнительная заработная плата разработчиков	9,728
Отчисление на социальные нужды	20,196
Прочие расходы	53,504
Общая сумма затрат на разработку	132,068

Следующие за разработкой процессы реализации и сопровождения также требуют определенных расходов. Затраты на реализацию определяются как:

$$З_{реал} = \frac{З_{разр} Н_{реал}}{100}, \quad (7.5)$$

где $З_{разр}$ – сумма затрат на разработку, млн. р.;
$Н_{реал}$ – норматив затрат на реализацию (5%).

$$З_{реал} = \frac{132{,}068 * 5}{100} = 6{,}604 \text{ млн. р.}$$

Затраты на сопровождение ПС определяются по формуле:

$$З_{сопр} = \frac{З_{разр} Н_{сопр}}{100}, \quad (7.6)$$

где $Н_{сопр}$ – норматив затрат на сопровождение (10%).

$$З_{сопр} = \frac{132{,}068 * 10}{100} = 13{,}207 \text{ млн. р.}$$

Итого сумма затрат на разработку, реализацию и сопровождение:

$$З = З_{разр} + З_{реал} + З_{сопр} = 151{,}878 \text{ млн. р.} \quad (7.7)$$

7.3 Оценка эффекта от использования ПС

Экономический эффект для организации-разработчика ПС заключается в получении прибыли от его реализации на рынке информационных технологий. Прибыль, в свою очередь, напрямую зависит от объема продаж, цены реализации и затрат на разработку ПС.

Изучение рынка и статистических данных о продажах программных продуктов схожего функционала позволило рассчитывать на 200-400 покупок лицензий в течение года реализации (при расчетах - 300) при цене не более 1 млн. р. (на основании цен аналогов - PTGui и Photoshop).

Прибыль предприятия от реализации единицы (копии):

$$П_{ед} = Ц - \frac{Ц * НДС}{100\% + НДС} - \frac{З}{N}, \quad (7.8)$$

где Ц – рыночная цена единицы продукта, тыс. р.;
НДС – доля налога на добавленную стоимость (20%);
N – количество реализованных копий за год, шт.

$$П_{ед} = 1000 - \frac{200}{1,2} - \frac{151878}{300} = 327,073 \text{ тыс. р.}$$

Суммарная годовая прибыль:

$$П = П_{ед} * N = 98,122 \text{ млн. р.} \quad (7.9)$$

Чистая прибыль, учитывающая действующий налог на прибыль (18%):

$$П_{ч} = П(1 - 0,18) = 80,460 \text{ млн. р.} \quad (7.10)$$

Значение чистой прибыли является численным выражением годового экономического эффекта инвестирования в проект.

7.4 Расчет показателей эффективности инвестиций в разработку ПС

Рассчитанный годовой экономический эффект ниже требуемого объема инвестиций, следовательно, они окупятся полностью лишь в течение нескольких лет. Расчет показателей эффективности инвестиций требует учета динамики прибылей и расходов на протяжении этого времени, для чего вводятся коэффициенты дисконтирования:

$$\alpha_t = (1 + E_n)^{t_p - t}, \quad (7.11)$$

где t - порядковый номер года реализации продукта;

t_p – номер расчетного года;

E_n- норма дисконта, не меньшая средней ставки по банковским депозитам на момент осуществления расчетов (40%).

Чистый дисконтированный доход рассчитывается по формуле:

$$\text{ЧДД} = \sum_t^n (\text{П}_t \alpha_t - \text{З}_t \alpha_t), \quad (7.12)$$

где n – порядковый номер года реализации продукта;

П_t – чистая прибыль в *t*-м году, млн. р.;

З_t – сумма затрат в *t*-м году, млн. р.

Результаты расчета дисконтированных значений прибылей и затрат приведены в таблице 7.2. Реализация продукта начинается в середине 2015 года, следовательно, прибыль за остаток года вдвое меньше прибыли за один полный год.

По результатам расчета, чистый дисконтированный доход за расчетный период составил 16,196 млн. р. Это положительная сумма, что говорит о целесообразности инвестирования.

Рентабельность затрат на разработку вычисляется по формуле:

$$\text{Р} = \frac{\text{П}_{\text{чср}}}{\text{З}} * 100\%, \quad (7.13)$$

где $\text{П}_{\text{чср}}$ – среднегодовая величина чистой прибыли, млн. р.

Таблица 7.2 – Дисконтированные значения прибылей и затрат

Показатель	Год реализации			
	2015	2016	2017	2018
Чистая прибыль, млн. р.	40,230	80,460	80,460	80,460
Дисконтированная прибыль, млн. р.	40,230	57,471	41,951	29,322
Затраты, млн. р.	151,878	-	-	-
Дисконтированная сумма затрат, млн. р.	151,878	-	-	-
Чистый дисконтированный доход за год, млн. р.	-111,648	57,471	41,051	29,322
Чистый дисконтированный доход нарастающим итогом, млн. р.	-111,648	-54,177	-13,126	16,196
Коэффициент дисконтирования	1,000	0,714	0,510	0,364

$$П_{чср} = \frac{40,230+80,460*3}{4} = 70,402 \text{ млн. р.},$$

$$Р = \frac{70,402}{151,878} * 100\% = 46,36\%.$$

Рассчитанное значение рентабельности существенно превышает средние процентные ставки по банковским депозитным вкладам, что свидетельствует об экономической эффективности проекта.

Срок окупаемости инвестиций – период времени, необходимый для того, чтобы полученная прибыль покрыла всю сумму инвестиций. Иначе говоря, это срок, за который чистый дисконтированный доход принимает положительное значение. Согласно динамике значений дохода (таблица 7.2), этот момент наступает в четвертом году реализации.

Вычисленные значения показателей свидетельствуют об экономической целесообразности разработки и применения программного средства.

8 ЭРГОНОМИЧЕСКАЯ ЭКСПЕРТИЗА ПРОГРАММНОГО СРЕДСТВА СИНТЕЗА ПАНОРАМНЫХ ИЗОБРАЖЕНИЙ

Эргономика – научная дисциплина, изучающая взаимодействие человека и других элементов некоторой системы, а также сферы деятельности по применению теории, принципов, данных и методов этой науки для обеспечения благополучия человека и оптимизации общей производительности системы [15]. Под системой может пониматься практически любая искусственная структура, участником или пользователем которой является человек. В контексте дипломного проектирования система – разрабатываемое программное средство. Эргономическая экспертиза направлена на улучшение общего качества продукта путем оптимизации пользовательского интерфейса.

8.1 Сущность информационной совместимости

Прежде чем говорить об информационной совместимости, следует дать определение информационной модели программы. Это совокупность входных и выходных данных, их вид, структура и способ восприятия – иначе говоря, интерфейс. При разработке программного обеспечения интерфейсы являются своеобразными связующими «мостами» между системами разного рода и назначения. Простота и эффективность интерфейса напрямую влияет на количество проблем, которые возникают при подобных взаимодействиях, поэтому его разработке уделяется большая доля внимания. Важной его спецификой является возможность использования для общения между разными системами. Следовательно, во-первых, качественный интерфейс может быть использован повторно, а во-вторых, эффективность работы с ним возрастает, так как работа с привычной, стандартной информационной моделью избавляет от процедуры освоения и позволяет использовать накопленный опыт при работе с программой.

Информационная совместимость – качественная мера, описывающая способность информационной модели отображать все характеристики описываемого объекта и предоставлять пользователю (оператору) условия для безошибочного восприятия и переработки информации, с учетом его психофизиологических характеристик и возможностей. К последним относятся размещение информационных зон на визуальном поле, особенности внимания, памяти и т.д. Информационная модель должна адекватно отображать управляемый объект, состояние системы управления, обеспечивать оптимальный объем данных.

Сложность при проектировании информационных моделей программного обеспечения состоит в том, что зачастую информация в программах представлена двоичными данными и наложенным на них строгим множеством многоуровневых абстракций. Пользователь непосредственно взаимодействует с физическими средствами ввода и вывода данных –

дисплеем, клавиатурой, мышью и прочими – для восприятия и оперирования абстрактными объектами, такими как файлы, окна, процессы. Широкое распространение персональных компьютеров с графическим пользовательским интерфейсов дало людям массовое представление о взаимодействии с компьютером. И сегодня навыки по обращению с вычислительными машинами постоянно улучшаются, что позволяет использовать более сложные в плане уровня абстракции, но вместе с тем более практичные интерфейсы. Однако правильное ограничение сложности все еще остается проблемой. Поэтому целесообразным является стремление использовать в интерфейсе максимальное количество стандартных элементов. Знание и использование действующих стандартов являются ключом к улучшению информационной совместимости.

Другой аспект совместимости связан с психофизиологическими особенностями человека. К примеру, размеры элементов интерфейса должны учитывать зрительные способности человека, которые, помимо индивидуальных особенностей, зависят от освещенности обозреваемого объекта и контрастности его деталей. Современные аппаратные средства и операционные системы обеспечивают основные требования совместимости – такие, как диапазон яркости, частота обновления экрана. От проектировщика интерфейса зависят менее критические, но тем не менее важные параметры:

- цветовая гамма графического интерфейса;
- частота происхождения событий, на которые может реагировать пользователь, и интенсивность его взаимодействия с программой;
- время отклика программы на действия пользователя;
- размеры элементов интерфейса, в частности, размер шрифта, используемого при выводе текста на экран;
- расположение элементов интерфейса;
- эстетические свойства интерфейса;
- и многие другие.

При проектировании значения большинства из подобных параметров следует брать по аналогии с другими распространенными программами, т.е. эффективно использовать их опыт в данной области. Однако слепое следование общепринятым стандартам не всегда приносит пользу. Так, отличие от других программных средств по части эргономических качеств может быть главным достоинством нового продукта.

8.2 Характеристика трудового процесса пользователя при работе с программным средством. Проектирование информационной архитектуры

Разрабатываемое программное средство предназначено для синтеза панорам, т.е. для обработки и представления графической информации. Следовательно, основное средство взаимодействия системы с пользователем – графический пользовательский интерфейс (GUI). В данном случае он

представлен окном приложения с рабочей областью для отображения информации пользователю. Интерфейс должен быть достаточен и эффективен при обеспечении выполнения программой основных функций:

- загрузка набора исходных изображений;
- генерация панорамы на основе набора исходных изображений;
- графическое представление панорамы с возможностью управления и изменения параметров обзора;
- сохранение и загрузка панорамы из файла на диске.

Первая проблема, с которой приходится столкнуться – размеры рабочей области. Ориентиром при проектировании этой характеристики является правило: зрительные маршруты по экрану должны быть минимизированы. Размещение последовательно воспринимаемой информации не должно вызывать переноса взгляда более чем на 20% от радиуса поля зрения [16]. Количество функций программы относительно невелико, то есть представление вариантов доступных пользователю действий может быть осуществлено в пределах компактной области экрана, что приветствуется вышеописанным принципом минимизации. С другой стороны, результат работы программы – изображение, и для его зрительной оценки необходимо обеспечить достаточный обзор. Это требование может быть реализовано и при небольших размерах рабочей области, если используются инструменты для обзора изображений – такие, как прокрутка и масштабирование. Следовательно, размер окна может быть уменьшен до минимума, определяемого удобством использования, который в итоге определен как 800 на 600 пикселей.

Генерация панорамы – поэтапный процесс. Навигация пользователя по нему линейна, однако ее присутствие дополняет информационную модель. К примеру, частым правилом при разработке навигационных элементов является их дублирование. Так, пользователь может выполнить некоторое действие (перейти на желанную страницу) одним из нескольких доступных способов - наиболее очевидным или простым для него. Примером являются элементы «Назад» и «Далее», которые нередко совмещены с другими элементами навигации. В разрабатываемом ПС их применение является целесообразным. Располагаться эти кнопки должны таким образом, чтобы обеспечивать быструю и безошибочную последовательность переходов от первого этапа до последнего и в обратном порядке. Есть смысл располагать элементы по краям и углам рабочей области, чтобы они не были помехой при отображении основной информации.

Элементы управления и ввода информации в систему непосредственно относятся к функциям программного средства и, следовательно, будут использоваться регулярно. Пользуясь правилом минимизации зрительных расстояний, их следует расположить вблизи от ранее определенных элементов навигации. Это подразумевает визуальное разделение функциональных и навигационных элементов с помощью их форм, цветов, расположения и прочих параметров. Назначение каждого элемента управления должно быть

очевидно, либо поясняться в доступной пользователю сноске. Количество одновременно доступных пользователю элементов следует ограничивать, если не предполагается их одновременное использование [17]. Так, по мере работы программы кнопки на графическом интерфейсе могут блокироваться, в зависимости от этапа процесса функционирования. Это облегчит выбор оператором своего дальнейшего действия.

Цветовая гамма интерфейса также играет заметную роль в определении эргономических параметров ПС. Традиционные окна в современных операционных системах имеют светло-серый цвет, с синим оттенком. Это имеет смысл, так как мягкие тона наименее всего нагружают глаза оператора. Цвета содержимого окна должны контрастировать с цветом фона, чтобы быть заметными пользователю. Высокие контрасты и негармоничная гамма отрицательно влияют на его психофизиологическое состояние, поэтому сочетание необходимо выбирать тщательно. Либо использовать гамму, стандартную для операционной системы – для небольших программ это является оптимальным решением. Таким образом, для ПС выбрана светло-серая гамма, по аналогу с операционными системами семейства Windows.

Размер шрифта, с помощью которого отображается текст в приложении, должен обеспечивать его разборчивость и читаемость. Чрезмерно большой размер сделает текст трудным для восприятия, а слишком малый – вдобавок к этому, потребует зрительных усилий.

Время на обработку результата программой всегда будет больше времени, в течение которого пользователь становится готов к восприятию результата. Причина этого – высокая сложность вычислений при анализе и синтезе изображений. Пользователь должен быть осведомлен о ходе процесса. Для этого используется стандартное решение, перенятое у операционных систем – анимированный значок, схожий с механическими часами. Он заменяет собой стандартный указатель мыши во время выполнения системой длительных операций.

8.3 Оценка эргономической эффективности человеко-машинного взаимодействия с помощью модели GOMS

8.3.1 Описание модели GOMS

GOMS (Goals, Operators, Methods and Selection roles) – модель интерфейса человек-компьютер, раскладывающая все возможные взаимодействия на ряды элементарных повторяющихся операций. Цели задаются пользователем программы. Операторы – атомарные действия, необходимые для достижения цели. Методы – последовательности операторов, достигающие определенную цель. Одной цели может соответствовать несколько методов. Наконец, правила выбора используются для обоснования выбора того или иного метода достижения цели.

Модель позволяет оптимизировать интерфейс на основе одной главной метрики – времени достижения целей. Среднестатистическое значение

времени, требуемого на выполнение операции, определяется единожды на основании реальных данных (см. таблицу 8.1). Время, требуемое определенным методом, обычно линейно зависит от времени выполнения его операций.

Существуют вариации моделей GOMS, различающиеся по эффективности, точности метрик и по сложности своей структуры. CPM-GOMS (Critical Path Method) ориентирована на планирование работы опытного пользователя, способного делать несколько операций единовременно. Это самая сложная модель типа GOMS, требующая глубокого понимания своих правил и концепций. KLM-GOMS (Keystroke-Level Model) – другой вариант модели, схожий с классическим. Он определяет алгоритм из одиннадцати шагов по построению модели GOMS [18]:

1) получить пошаговое описания рассматриваемой задачи;
2) определить цели и ожидаемый результат работы;
3) определить подцели для каждой цели;
4) идентифицировать методы всех целей и подцелей;
5) перевести описания методов в псевдокод;
6) зафиксировать допущения, совершенные на предыдущем шаге.
7) определить умственную или физическую операцию каждого шага;
8) определить время выполнения каждой операции;
9) определить количество выполнений каждой операции;
10) скорректировать значения времени с учетом возрастных и т.п. особенностей основной группы пользователей;
11) проверить результаты.

Полученная система из целей, методов и операций анализируется на наличие лишних операций или возможность их упрощения. Метод KLM-GOMS получил самое широкое распространение ввиду своей простоты. Недостатком его является планирование взаимодействия лишь с опытными пользователями, знающими все доступные методы и операции.

Таблица 8.1 – Временная оценка операций, выполняемых пользователем

Код операции	Описание	Время, сек.
K	Нажать и отпустить клавишу	0,28
T(n)	Ввести n символов	K*n
P	Навести указатель мыши на объект на экране	1,10
B	Нажать и отпустить кнопку на мыши (клик)	0,10
H	Переместить руку с мыши на клавиатуру и наоборот	0,40
M	Умственная подготовка	1,20
W(t)	Ожидание ответа системы	t

8.3.2 Оценка задачи генерации панорамы

Разрабатываемое программное средство обладает простым интерфейсом, при котором объем взаимодействия программы и пользователя

сводится к минимуму. Поэтому в модели GOMS может быть рассмотрена вся последовательность действий при работе с приложением (в простейшем случае, когда результат генерации считается удовлетворительным, и пользователь не видит необходимости в его правке).

При запуске программы пользователь видит главное окно приложения. Он щелкает указателем мыши по кнопке «Add images..», и с помощью стандартного диалогового окна выбирает их (количеством не менее двух шт.) из файловой системы. При нажатии кнопки «Open» диалога он видит эти файлы, представленные в окне приложения в удобной для манипулирования форме. После добавления файлов пользователь нажимает на элемент навигации «Next», и запускается процесс генерации панорамы. Пользователь узнает об этом по смене вида указателя мыши. Когда процесс завершен, пользователь видит результат, оценивает его, щелкает по кнопке «Save as..». В открывшемся диалоге он выбирает нужный формат нового файла и вводит имя для него. После нажатия кнопки «Save» диалога пользователь снова видит результат. Он нажимает на крестообразную иконку в правом верхнем углу окна для закрытия приложения.

Используемые в этом методе операции по работе с диалоговыми окнами являются стандартными для оконных приложений, и возможностей по их оптимизации очень мало. Поэтому для упрощения они будут считаться цельными операциями (D), по 5 секунд каждая.

Данные шаги представлены в виде псевдокода в таблице 8.2.

Таблица 8.2 – Операции, необходимые для генерации панорамы в ПС

Код операции	Описание	Время, сек.
M	Умственная подготовка	1,20
P	Наведение указателя на кнопку «Add images..»	1,10
B	Клик по кнопке	0,10
D	Работа с диалоговым окном выбора файлов	5,00
P	Наведение указателя на кнопку «Next»	1,10
B	Клик по кнопке	0,10
W(5)	Ожидание ответа системы	5,00
M	Умственная подготовка	1,20
P	Наведение указателя на кнопку «Save as..»	1,10
B	Клик по кнопке	0,10
D	Работа с диалоговым окном сохранения файла	5,00
P	Наведение указателя на иконку закрытия окна приложения	1,10
B	Клик по иконке	0,10

В сумме, выполнение метода займет 22,2 секунды, от запуска до закрытия программы.

Анализ последовательности показывает, что повторяемой операцией при работе с ПС является наведение и клик по различным кнопкам на интерфейсе. Существует альтернатива этим действиям. В популярных приложениях, поддерживающими работу с файлами, часто определены комбинации «быстрых клавиш». Так, нажатие и удержание нескольких клавиш в определенном порядке вызывает выполнение некоторого действия, которое закреплено за этим сочетанием. Для операций загрузки и сохранения файлов обычно используются комбинации Ctrl+O и Ctrl+S, соответственно. Для закрытия приложения также можно ввести особую комбинацию, состоящую из одной клавиши - Esc. Так как подобное поведение присуще ограниченному числу приложений и при случайном использовании может привести к потере данных, следует вывести диалог с требованием подтверждения закрытия приложения. Для ускорения работы с ним можно установить фокус по умолчанию на вариант «Закрыть», что позволит совершить операцию одним нажатием клавиши Enter. Аналогичный подход целесообразен и для элемента навигации «Next», на который будет установлен фокус после выбора файлов.

Обновленная последовательность операций для получения панорамы представлена в таблице 8.3.

Таблица 8.3 – Оптимизированная последовательность операций для генерации панорамы в программном средстве синтеза панорамных изображений

Код операции	Описание	Время, сек.
M	Умственная подготовка	1,20
D	Работа с диалоговым окном выбора файлов	5,00
K	Нажатие клавиши Enter	0,28
W(5)	Ожидание ответа системы	5,00
M	Умственная подготовка	1,20
K	Нажатие и удержание клавиши Ctrl	0,28
K	Нажатие клавиши S	0,28
D	Работа с диалоговым окном сохранения файла	5,00
K	Нажатие клавши Esc	0,28
K	Нажатие клавиши Enter	0,28

В сумме, выполнение нового метода занимает 18,8 секунд. Таким образом, изменения в интерфейсе программы обеспечили ускорение процедуры получения панорамы на 3,2 секунды. При этом необходимые от пользователя действия могут выполняться без использования мыши.

Метод GOMS помог выявить резервы оптимизации интерфейса программы. Возможность ускорения взаимодействия позволяет пользователям использовать программное средство наиболее удобным для них способом. О наличии такой возможности пользователь может узнать из

всплывающих подсказок при осмотре интерфейса программы (см. рисунок 8.1).

Интерфейс программного средства спроектирован с использованием ряда принципов эргономической эффективности и оптимизирован по длительности процесса функционирования. Следовательно, уменьшено время воздействия на пользователя негативных факторов от взаимодействия с компьютером, связанных в основном со зрительной нагрузкой. Стремление к простоте и использование стандартных приемов позволило ослабить эффект стресса, возникающего при необходимости освоения нового интерфейса.

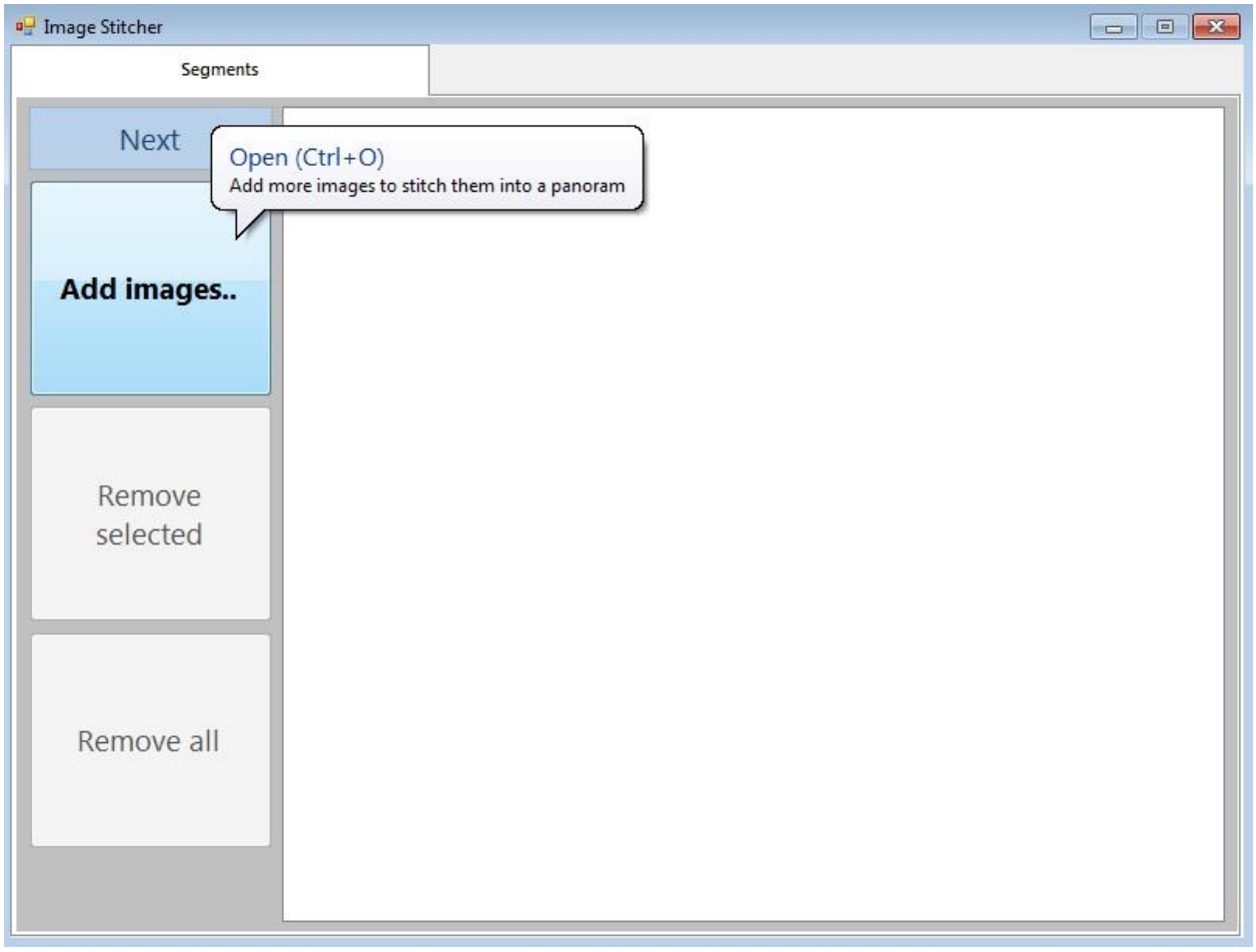

Рисунок 8.1 – Скриншот окна приложения

ЗАКЛЮЧЕНИЕ

В результате выполнения дипломного проекта было разработано программное средство, обеспечивающее автоматизированный синтез панорам на основе множества выбранных изображений.

Была изучены научные области компьютерного зрения и обработки изображений, определены современные методы решения поставленной задачи и примеры их реализации. На основе этих данных сформулированы основные требования к программному средству. В качестве основы для их реализации среди множества изученных методов была выбрана группа наиболее практичных, таких как: перспективное преобразование координат, методы SIFT и RANSAC.

Было осуществлено планирование процесса разработки. На его основе рассчитаны численные параметры, определившие технико-экономическую целесообразность разработки и дальнейшего использования программного средства.

В ходе анализа требований был последовательно обоснован общий вид архитектуры программного средства, а затем детализирован до уровня компонентов системы, с установленными ролями. Определен общий ход выполнения программы, а также оптимальный способ взаимодействия с пользователем – посредством удобного графического интерфейса. При создании интерфейса как показатель качества использовались его эргономические свойства. Дополнительно проработан алгоритм выполнения основной функции программного средства – генерации панорамы.

Была выбрана подходящая среда для разработки программного средства, а также доступный набор инструментов, позволяющий использовать готовые эффективные решения при реализации. Компоненты программного средства были поэтапно реализованы в виде достаточно изолированных модулей и библиотек, с управляемыми связями между ними. Они приспособлены к расширению, что способствует дальнейшему развитию и оптимизации программного средства. Создан простой пользовательский интерфейс, обеспечивающий эффективную работу с программой. Выполнена сборка программного средства и проверено его функционирование с помощью разработанных тестовых сценариев.

Разработанное в результате программное средство отвечает всем основным функциональным требованиям. Пользователь приложения имеет возможность загружать набор исходных изображений и изменять его. Разница в соотношениях сторон, углах наклона и масштабах изображений мало влияет на результат работы программы в результате использования эффективных математических методов. Величина областей перекрытия, разница в цветовой гамме и в искажении определяют качество созданной панорамы, однако их влияние может быть изменено путем настройки процесса генерации с помощью специального пользовательского интерфейса. Результат работы –

панорама - может быть сохранен в независимом файле на компьютере пользователя.

Таким образом, установленные задачи дипломного проектирования можно считать выполненными, а цель – достигнутой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа: http://en.wikipedia.org/wiki/Computer_vision

[2] Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа: http://en.wikipedia.org/wiki/Image_stitching

[3] Szeliski, R. Image Alignment and Stitching / R. Szeliski - MSR-TR-2004-92

[4] Cambridge in Colour [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.cambridgeincolour.com/tutorials/image-projections.htm

[5] Tuytelaars, T. Local Invariant Feature Detectors: A survey / T. Tuytelaars, K. Mikolajczyk - 2006

[6] Habrahabr [Электронный ресурс]. – Электронные данные. – Режим доступа: http://habrahabr.ru/post/106302/

[7] Brown, M. Multi-Image Matching using Multi-Scale oriented Patches / M. Brown, R. Szeliski, S. Winder – Microsoft Research, 2004 MSR-TR-2004-8

[8] Habrahabr [Электронный ресурс]. – Электронные данные. – Режим доступа: http://habrahabr.ru/post/244541/

[9] Szeliski, R. Creating Full View Panoramic Image Mosaics and Environment Maps / R. Szeliski, H.Y. Shum – MSR-TR-97-8

[10] Zhu, M. Efficient Video Panoramic Image Stitching Based on an Improved Selection of Harris Corners and a Multiple-Constraint Corner Matching / M. Zhu, W. Wang, J. Huang – 2013 PLoS ONE 8(12): e81182. doi: 10.1371 / journal.pone.0081182

[11] Belgacem, M. Panoramic Image Stitching : report – University of Geneva, 2014

[12] Hays, James. Computational Photography : tutorial – Brown University courses, 2010

[13] AI Shack [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.aishack.in/tutorials/sift-scale-invariant-feature-transform-introduction/

[14] Kwatra, V. Graphcut Textures: Image and Video Synthesis Using Graph Cuts / V. Kwatra, A. Schodl, I. Essa, G. Turk, A. Bobick - GVU Center / College of Computing Georgia Institute of Technology

[15] Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Эргономика

[16] Neonstudio [Электронный ресурс]. – Электронные данные. – Режим доступа: http://neonstudio.ru/info/ergonomika_informacii/

[17] Сергеев, С. Методы тестирования и оптимизации интерфейсов информационных систем: учебное пособие. – СПб: НИУ ИТМО, 2013. – 117 с.

[18] Wikipedia [Электронный ресурс]. – Электронные данные. – Режим доступа: http://en.wikipedia.org/wiki/Keystroke-level_model

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код библиотеки синтеза панорам и ее расширений

Модули библиотеки Panoramas.
Класс Panoramas.Defaults.Stitcher:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas.Defaults {
  public class Stitcher : IStitcher {
    protected IFactory factory;
    protected IPanoramaImages panorama_segments;
    protected IPanoramaRelations panorama_relations;
    protected IPanoramaTransformations panorama_complete;
    protected IAnalyzer analyzer;
    protected IBuilder builder;
    protected IPresenter presenter;

    public Stitcher(IFactory factory, IImage[] images) {
      if (images.Length < 2)
        throw new ArgumentException("Not enough images");
      this.factory = factory;
      this.analyzer = factory.Analyzer();
      this.builder = factory.Builder();
      this.presenter = factory.Presenter();
      this.panorama_segments = factory.PanoramaImages(images);
      this.panorama_relations = analyzer.Analyze(panorama_segments);
    }

    public IRelationControl MatchBetween(String image_base, String
image_matched) {
      return panorama_relations.MatchBetween(image_base, image_matched);
    }

    public Bitmap StitchAll() {
      this.panorama_complete = builder.Build(panorama_relations);
      return presenter.Present(panorama_complete);
    }
  }
}
```

Интерфейс Panoramas.IImage:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas {
  public interface IImage {
    System.Drawing.Bitmap Bitmap { get; }
    String Name { get; }
```

```
  }
}
```

Интерфейс Panoramas.IImageTransformed:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas {
  public interface IImageTransformed :IImage {
    ITransformation Transformation { get; }
  }
}
```

Интерфейс Panoramas.IStitcher:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas {
  public interface IStitcher {
    void AddImage(String name, System.Drawing.Bitmap bitmap);
    IRelationControl MatchBetween(String image_base, String
image_matched);
    System.Drawing.Bitmap StitchAll();
  }
}
```

Класс Panoramas.Defaults.Stitcher:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Panoramas.Processors;

namespace Panoramas.Defaults {
  public class Stitcher : IStitcher {
    protected IFactory factory;
    protected IPanoramaImages panorama_segments;
    protected IPanoramaRelations panorama relations;
    protected IPanoramaTransformations panorama_complete;
    protected IAnalyzer analyzer;
    protected IBuilder builder;
    protected IPresenter presenter;

    public Stitcher(IFactory factory, IImage[] images) {
      if (images.Length < 2)
        throw new ArgumentException("Not enough images");
      this.factory = factory;
      this.analyzer = factory.Analyzer();
      this.builder = factory.Builder();
      this.presenter = factory.Presenter();
```

```
            this.panorama_segments = factory.PanoramaImages(images);
            this.panorama_relations = analyzer.Analyze(panorama_segments);
          }

          public IRelationControl MatchBetween(String image_base, String
image_matched) {
            return panorama_relations.MatchBetween(image_base, image_matched);
          }

          public Bitmap StitchAll() {
            this.panorama_complete = builder.Build(panorama_relations);
            return presenter.Present(panorama_complete);
          }

          public void AddImage(string name, Bitmap bitmap) {
            throw new NotImplementedException();
          }
        }
      }
```

Интерфейс Panoramas.IPanoramaImages:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas {
  public interface IPanoramaImages {
    IImage[] Images { get; }
  }
}
```

Класс Panoramas.Defaults.PanoramaImages:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas.Defaults {
  public class PanoramaImages : IPanoramaImages {
    public IImage[] Images { get; private set; }

    public PanoramaImages(IImage[] segments) {
      this.Images = segments;
    }

    public PanoramaImages(IPanoramaImages panorama) {
      this.Images = panorama.Images;
    }
  }
}
```

Интерфейс Panoramas.IPanoramaRelations:

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace Panoramas {
  public interface IPanoramaRelations : IPanoramaImages {
    List<IImagesRelation> Relations { get; }
    IImage Core();
    IImagesRelation MatchBetween(IImage base_segment, IImage
related_image);
    IRelationControl MatchBetween(String base_iamge, String
related_image);
    IImage[] NeighboursOf(IImage segment, IImage[] domain = null);
    void ClosestBetween(IImage[] group, IImage[] domain, out IImage
closest, out IImage closest_from_group);
  }
}
```

Класс Panoramas.Defaults.PanoramaRelations:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas.Defaults {
  public class PanoramaRelations: PanoramaImages, IPanoramaRelations {
    public List<IImagesRelation> Relations { get; private set; }

    public PanoramaRelations(IPanoramaImages panorama,
List<IImagesRelation> relations)
      : base(panorama) {
        this.Relations = relations;
    }

    public PanoramaRelations(IPanoramaRelations panorama)
      : base(panorama) {
        this.Relations = panorama.Relations;
    }

    public IImage Core() {
      return Images.OrderBy((s) => distancesFor(s)).First();
    }

    public IImagesRelation MatchBetween(IImage base_segment, IImage
query_segment) {
      if (base_segment == query_segment ||
!Images.Contains(base_segment) || !Images.Contains(query_segment))
        throw new ArgumentException("Request for missing images");
      return Relations.Find((m) => m.BaseSegment == base_segment &&
m.QuerySegment == query_segment);
    }

    public IRelationControl MatchBetween(String base_segment, String
query_segment) {
      var all_files = Images.Select((s) => s.Name);
      if (base_segment == query_segment ||
!all_files.Contains(base_segment) || !all_files.Contains(query_segment))
        throw new ArgumentException("Request for missing images");
      return Relations.Find((m) => m.BaseSegment.Name == base_segment &&
m.QuerySegment.Name == query_segment);
    }
```

```
          public void ClosestBetween(IImage[] group, IImage[] domain, out
IImage closest, out IImage closest_from_group) {
            if (domain.Intersect(group).Count() > 0)
              throw new ArgumentException();
            if (domain == null)
              domain = Images;
            var match = Relations.
              Where((m) => m.Segments.Intersect(group).Count() > 0).
              Where((m) => m.Segments.Intersect(domain).Count() > 0).
              OrderBy((m) => m.Similarity()).
              First();
            closest = match.Segments.Except(group).First();
            closest_from_group = match.Segments.Except(domain).First();
          }

          public IImage[] NeighboursOf(IImage segment, IImage[] domain = null)
{
            if (domain == null)
              domain = Images;
            return domain.Where((s) => closestTo(s) == segment).ToArray();
          }

          protected IImage closestTo(IImage segment) {
            return Relations.
              Where((m) => m.Includes(segment)).
              OrderBy((m) => m.Similarity()).
              First().
              PairOf(segment);
          }

          double distancesFor(IImage segment) {
            return Relations.Sum((m) => m.QuerySegment == segment ?
m.Similarity() : 0);
          }
        }
      }
```

Интерфейс Panoramas.IPanoramaTransformations:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas {
  public interface IPanoramaTransformations : IPanoramaRelations {
    IImageTransformed[] Segments { get;  }
  }
}
```

Класс Panoramas.Defaults.PanoramaTransformations:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas.Defaults {
```

```
        public class PanoramaTransformations: PanoramaRelations,
IPanoramaTransformations {
          public IImageTransformed[] Segments { get; private set; }

          public PanoramaTransformations(IPanoramaRelations panorama,
IImageTransformed[] segments)
            : base(panorama) {
              this.Segments = segments;
          }
        }
      }
```

Интерфейс Panoramas.IImagesRelation:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas {
  public interface IImagesRelation : IRelationControl {
    IImage BaseSegment { get; }
    IImage QuerySegment { get; }
    IImage[] Segments { get; }
    IImagesRelation ReversePair { get; set; }
    ITransformation GenerateTransformation();
    bool Includes(IImage image);
    IImage PairOf(IImage image);
  }
}
```

Интерфейс Panoramas.Itransformation:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas {
  public interface ITransformation {
    void Move(int x_diff, int y_diff);
    void Reset();
    void Distort(ITransformation transformation);
    ITransformation Multiply(ITransformation transformation);
    Bitmap TransformOn(Bitmap image, Bitmap template);
    PointF Transform(PointF point);
  }
}
```

Интерфейс Panoramas.IRelationControl:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace Panoramas {
  public interface IRelationControl {
    Bitmap ToImage();
    double Similarity();
    int LimitPercent { get; set; }
    bool Active { get; set; }
  }
}
```

Интерфейс Panoramas.Procedures.IAnalyzer:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas.Processors {
  public interface IAnalyzer {
    IPanoramaRelations Analyze(IPanoramaImages panorama_segment);
    bool AddImage(IPanoramaRelations panorama_relations, IImage image);
    bool RemoveImage(IPanoramaRelations panorama_relations, IImage image);
  }
}
```

Интерфейс Panoramas.Procedures.IBuilder:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas.Processors {
  public interface IBuilder {
    IPanoramaTransformations Build(IPanoramaRelations panorama_relations);
  }
}
```

Интерфейс Panoramas.Procedures.IPresenter:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas.Processors {
  public interface IPresenter {
    System.Drawing.Bitmap Present(IPanoramaTransformations panorama);
  }
}
```

Интерфейс Panoramas.IFactory:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using Panoramas.Processors;

namespace Panoramas {
  public interface IFactory {
    IImage Image(String name, System.Drawing.Bitmap bitmap);
    ITransformation Transformation();
    IImageTransformed ImageTransformed(IImage image, ITransformation
transformation);

    IPanoramaImages PanoramaImages(IImage[] images);
    IPanoramaRelations PanoramaRelations(IPanoramaImages panorama_segments,
List<IImagesRelation> relations);
    IPanoramaTransformations PanoramaTransformations(IPanoramaRelations
panorama_relations, IImageTransformed[] segments);

    IStitcher Stitcher(IImage[] images);
    IAnalyzer Analyzer();
    IBuilder Builder();
    IPresenter Presenter();
  }
}
```

Интерфейс Panoramas.IPublicFactory:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas {
  public interface IPublicFactory {
    IStitcher Stitcher(String[] files, System.Drawing.Bitmap[] bitmaps =
null);
  }
}
```

Класс Panoramas.Defaults.Factory:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Panoramas.Processors;

namespace Panoramas.Defaults {
  public abstract class DefaultsFactory : IFactory, IPublicFactory {

    public IImageTransformed ImageTransformed(IImage image, ITransformation
transformation) {
      return new Segment(image, transformation);
    }

    public IPanoramaImages PanoramaImages(IImage[] images) {
      return new PanoramaImages(images);
    }

    public IPanoramaRelations PanoramaRelations(IPanoramaImages panorama,
List<IImagesRelation> relations) {
      return new PanoramaRelations(panorama, relations);
    }
```

```
    public IPanoramaTransformations
PanoramaTransformations(IPanoramaRelations panorama, IImageTransformed[]
segments) {
      return new PanoramaTransformations(panorama, segments);
    }

    public IStitcher Stitcher(IImage[] images) {
      return new Stitcher(this, images);
    }

    public IImage Image(string name, System.Drawing.Bitmap bitmap) {
      return new Segment(name, bitmap);
    }

    public IStitcher Stitcher(String[] files, System.Drawing.Bitmap[] bitmaps
= null) {
      if (files.Length < 2 || files.Length != bitmaps.Length)
        throw new ArgumentException("Not enough images");
      var images = new List<IImage>();
      for (int i = 0; i < files.Length; i++)
        images.Add(Image(files[i], bitmaps[i]));
      return Stitcher(images.ToArray());
    }

    public abstract IAnalyzer Analyzer();
    public abstract IBuilder Builder();
    public abstract IPresenter Presenter();
    public abstract ITransformation Transformation();
  }
}
```

Модули расширения PanoramasFeaturesAnalyzer.
Класс Panoramas.FeaturesAnalyzer.Analyzer:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Panoramas.Processors;

namespace Panoramas.FeaturesAnalyzer {
  public class Analyzer : IAnalyzer {
    IFactory factory;

    public Analyzer(IFactory factory) {
      this.factory = factory;
    }

    public IPanoramaRelations Analyze(IPanoramaImages panorama_segment) {
      var relations = generateMatches(panorama_segment.Images);
      return factory.PanoramaRelations(panorama_segment, relations);
    }

    static List<IImagesRelation> generateMatches(IImage[] segments) {
      var featured_segments = segments.Select((s) => s.Bitmap).ToArray();
      var matches = new List<IImagesRelation>();
      var matched_segments = new List<IImage>();
      for (int iBase = 0; iBase < segments.Length; iBase++) {
        var base_segment = segments[iBase];
        for (int iQuery = 0; iQuery < segments.Length; iQuery++) {
```

```
          if (iBase == iQuery)
            continue;
          var matched_segment = segments[iQuery];
          var match = new SegmentsPair(base_segment, matched_segment);
          if (matched_segments.Contains(matched_segment)) {
            var prev_match = matches.Find((m) => m.BaseSegment ==
matched_segment && m.QuerySegment == base_segment);
            match.ReversePair = prev_match;
          }
          matches.Add(match);
        }
        matched_segments.Add(base_segment);
      }
      return matches;
    }


    public bool AddImage(IPanoramaRelations panorama_relations, IImage image)
{
      throw new NotImplementedException();
    }

    public bool RemoveImage(IPanoramaRelations panorama_relations, IImage
image) {
      throw new NotImplementedException();
    }
  }
}
```

Класс Panoramas.FeaturedAnalyzer.KeyPointsPair:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Emgu.CV.Structure;

namespace Panoramas.FeaturesAnalyzer {
  public class KeyPointsPair {
    public MKeyPoint Left { get; private set; }
    public MKeyPoint Right { get; private set; }
    public double Distance { get; private set; }

    public KeyPointsPair(MKeyPoint left, MKeyPoint right, double distance) {
      this.Left = left;
      this.Right = right;
      this.Distance = distance;
    }
  }
}
```

Класс Panoramas.FeatureAnalyzer.SegmentsPair:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas.FeaturesAnalyzer {
```

```
  public class SegmentsPair : IImagesRelation {
    public const int MIN_MATCHES_COUNT = 10;

    int limit;
    double? distance;
    KeyPointsPair[] all_matches;
    bool active;

    public KeyPointsPair[] Matches { get; private set; }
    public IImage BaseSegment { get; private set; }
    public IImage QuerySegment { get; private set; }
    public IImage[] Segments {
      get { return new IImage[] { BaseSegment, QuerySegment }; }
    }
    public bool Active {
      get { return active; }
      set {
        if (Matches.Length < MIN_MATCHES_COUNT && value == true) {
          this.active = false;
          return;
        }
        this.active = value;
        if (ReversePair != null && ReversePair.Active != active) {
          ReversePair.Active = active;
        }
      }
    }
    public IImagesRelation ReversePair { get; set; }

    public SegmentsPair(IImage base_segment, IImage query) {
      this.BaseSegment = base_segment;
      this.QuerySegment = query;
      var matcher = new Flann.Matcher(BaseSegment.Bitmap,
QuerySegment.Bitmap);
      this.all_matches = matcher.Match();
      setOptimalLimit();
    }

    public void SetReversePair(IImagesRelation pair) {
      this.ReversePair = pair;
      pair.ReversePair = this;
    }

    public int LimitPercent {
      get {
        return limit;
      }
      set {
        var percent = value;
        this.limit = percent;
        if (percent < 0 || percent > 100)
          throw new ArgumentException("Invalid parameter value");
        var matches_count = all_matches.Length * percent / 100;
        if (matches_count < MIN_MATCHES_COUNT) {
          active = false;
        }
        if (ReversePair != null && ReversePair.LimitPercent !=
this.LimitPercent) {
          ReversePair.LimitPercent = percent;
        }
        setCountLimit(matches_count);
      }
    }
```

```
    public System.Drawing.Bitmap ToImage() {
      return new PairPresenter(this).Render();
    }

    public double Similarity() {
      if (distance == null || Double.IsNaN((double)distance))
        resetDistance();
      return (double)distance;
    }

    public ITransformation GenerateTransformation() {
      var points_dst = this.Matches.Select((m) => m.Left.Point).ToArray();
      var points_src = this.Matches.Select((m) => m.Right.Point).ToArray();
      return
Panoramas.HomographyTransformer.Transformation.Generate(points_dst,
points_src);
    }

    public bool Includes(IImage segment) {
      return BaseSegment == segment || QuerySegment == segment;
    }

    public IImage PairOf(IImage segment) {
      if (segment == BaseSegment)
        return QuerySegment;
      else if (segment == QuerySegment)
        return BaseSegment;
      else
        return null;
    }

    void setOptimalLimit() {
      int count = MIN_MATCHES_COUNT;
      this.limit = count * 100 / all_matches.Length;
      this.active = true;
      setCountLimit(count);
    }

    void setCountLimit(int count) {
      this.Matches = all_matches.Take(count).ToArray();
      resetDistance();
    }

    void resetDistance() {
      var sum = Matches.Sum((m) => m.Distance);
      this.distance = sum / Matches.Length;
    }
  }
}
```

Класс Panoramas.HomographyTransformer.Transformation:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas.FeaturesAnalyzer {
  public class SegmentsPair : IImagesRelation {
    public const int MIN_MATCHES_COUNT = 10;
```

```
    int limit;
    double? distance;
    KeyPointsPair[] all_matches;
    bool active;

    public KeyPointsPair[] Matches { get; private set; }
    public IImage BaseSegment { get; private set; }
    public IImage QuerySegment { get; private set; }
    public IImage[] Segments {
      get { return new IImage[] { BaseSegment, QuerySegment }; }
    }
    public bool Active {
      get { return active; }
      set {
        if (Matches.Length < MIN_MATCHES_COUNT && value == true) {
          this.active = false;
          return;
        }
        this.active = value;
        if (ReversePair != null && ReversePair.Active != active) {
          ReversePair.Active = active;
        }
      }
    }
    public IImagesRelation ReversePair { get; set; }

    public SegmentsPair(IImage base_segment, IImage query) {
      this.BaseSegment = base_segment;
      this.QuerySegment = query;
      var matcher = new Flann.Matcher(BaseSegment.Bitmap,
QuerySegment.Bitmap);
      this.all_matches = matcher.Match();
      setOptimalLimit();
    }

    public void SetReversePair(IImagesRelation pair) {
      this.ReversePair = pair;
      pair.ReversePair = this;
    }

    public int LimitPercent {
      get {
        return limit;
      }
      set {
        var percent = value;
        this.limit = percent;
        if (percent < 0 || percent > 100)
          throw new ArgumentException("Invalid parameter value");
        var matches_count = all_matches.Length * percent / 100;
        if (matches_count < MIN_MATCHES_COUNT) {
          active = false;
        }
        if (ReversePair != null && ReversePair.LimitPercent !=
this.LimitPercent) {
          ReversePair.LimitPercent = percent;
        }
        setCountLimit(matches_count);
      }
    }

    public System.Drawing.Bitmap ToImage() {
```

```
      return new PairPresenter(this).Render();
    }

    public double Similarity() {
      if (distance == null || Double.IsNaN((double)distance))
        resetDistance();
      return (double)distance;
    }

    public ITransformation GenerateTransformation() {
      var points_dst = this.Matches.Select((m) => m.Left.Point).ToArray();
      var points_src = this.Matches.Select((m) => m.Right.Point).ToArray();
      return
Panoramas.HomographyTransformer.Transformation.Generate(points_dst,
points_src);
    }

    public bool Includes(IImage segment) {
      return BaseSegment == segment || QuerySegment == segment;
    }

    public IImage PairOf(IImage segment) {
      if (segment == BaseSegment)
        return QuerySegment;
      else if (segment == QuerySegment)
        return BaseSegment;
      else
        return null;
    }

    void setOptimalLimit() {
      int count = MIN_MATCHES_COUNT;
      this.limit = count * 100 / all_matches.Length;
      this.active = true;
      setCountLimit(count);
    }

    void setCountLimit(int count) {
      this.Matches = all_matches.Take(count).ToArray();
      resetDistance();
    }

    void resetDistance() {
      var sum = Matches.Sum((m) => m.Distance);
      this.distance = sum / Matches.Length;
    }
  }
}
```

Класс Panoramas.FeaturesAnalyzer.Flann.FeatuedImage:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Emgu.CV.Features2D;
using Emgu.CV.Structure;

namespace Panoramas.FeaturesAnalyzer.Flann {
  public class FeaturedImage {
```

```
    public Bitmap Image { get; protected set; }
    public int Width { get; private set; }
    public int Height { get; private set; }

    public FeaturedImage(Bitmap image) {
      this.Image = image;
      this.Width = Image.Width;
      this.Height = Image.Height;
      var emgu_image = new Emgu.CV.Image<Gray, byte>(Image);
      var detector = new Emgu.CV.Features2D.SIFTDetector();
      _features = detector.DetectFeatures(emgu_image, null);
    }
    public FeaturedImage(String filename) : this(new Bitmap(filename)) { }

    Emgu.CV.Features2D.ImageFeature<float>[] _features;
    public Emgu.CV.Features2D.ImageFeature<float>[] Features {
      get { return _features; }
    }
  }
}
```

Класс Panoramas.FeaturesAnalyzer.Flann.Matcher:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Emgu.CV.Features2D;

namespace Panoramas.FeaturesAnalyzer.Flann {
  public class Matcher {
    const int KNN = 1;
    const int SEARCH_ITERATIONS_COUNT = 24;

    FeaturedImage ImageBase;
    FeaturedImage ImageQuery;

    public Matcher(FeaturedImage image_base, FeaturedImage image_query) {
      this.ImageBase = image_base;
      this.ImageQuery = image_query;
    }
    public Matcher(Bitmap bitmap_base, Bitmap bitmap_query) :
      this(new FeaturedImage(bitmap_base), new FeaturedImage(bitmap_query)) {
}

    public KeyPointsPair[] Match() {
      var index = new
Emgu.CV.Flann.Index(FeaturesToMatrix(ImageBase.Features));
      var query = FeaturesToMatrix(ImageQuery.Features);
      int features_count = ImageQuery.Features.Length;
      var indices = new Emgu.CV.Matrix<int>(features_count, 1);
      var distances = new Emgu.CV.Matrix<float>(features_count, 1);
      index.KnnSearch(query, indices, distances, KNN,
SEARCH_ITERATIONS_COUNT);
      var matches = new KeyPointsPair[features_count];
      for (int iQueryFeature = 0; iQueryFeature < features_count;
iQueryFeature++) {
        var iBaseFeature = indices[iQueryFeature, 0];
        matches[iQueryFeature] = new KeyPointsPair(
          ImageBase.Features[iBaseFeature].KeyPoint,
```

```
          ImageQuery.Features[iQueryFeature].KeyPoint,
          distances[iQueryFeature, 0]);
      }
      return matches.OrderBy((pair) => { return pair.Distance; }).ToArray();
    }

    Emgu.CV.Matrix<float> FeaturesToMatrix(ImageFeature<float>[] features) {
      float[,] matrix = new float[features.Length,
features[0].Descriptor.Length];
      for (int iFeature = 0; iFeature < features.Length; iFeature++) {
        var descriptor = features[iFeature].Descriptor;
        for (int iComponent = 0; iComponent < descriptor.Length;
iComponent++)
          matrix[iFeature, iComponent] = descriptor[iComponent];
      }
      return new Emgu.CV.Matrix<float>(matrix);
    }

    Emgu.CV.Util.VectorOfKeyPoint FeaturesToVector(ImageFeature<float>[]
features) {
      var vector = new Emgu.CV.Util.VectorOfKeyPoint();
      vector.Push(features.Select((feat) => feat.KeyPoint).ToArray());
      return vector;
    }

    System.Drawing.PointF[] FeaturesToPoints(ImageFeature<float>[] features)
{
      return features.Select((f) => f.KeyPoint.Point).ToArray();
    }
  }
}
```

Модули расширения PanoramasTreeBuilder.
Класс Panoramas.TreeBuilder.TreeNode:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas.TreeBuilder {
  public class TreeNode {
    public IImage Image { get; private set; }
    public List<TreeNode> Children { get; private set; }
    public TreeNode Parent { get; private set; }
    public ITransformation Transformation { get; private set; }

    public TreeNode(IImage image, ITransformation transformation, TreeNode
parent = null) {
      this.Image = image;
      this.Transformation = transformation;
      this.Parent = parent;
      this.Children = new List<TreeNode>();
      updateTransformation();
    }

    public TreeNode AddChild(IImage image, ITransformation transformation) {
      var child = new TreeNode(image, transformation, this);
      Children.Add(child);
      return child;
    }
```

```
    public TreeNode FindNode(IImage image) {
      if (Image == image)
        return this;
      else
        foreach (var child in Children) {
          var match = child.FindNode(image);
          if (match != null)
            return match;
        }
      return null;
    }

    public IImage[] CollectSegments(List<IImage> images = null) {
      if (images == null)
        images = new List<IImage>();
      images.Add(Image);
      foreach (var child in Children)
        child.CollectSegments(images);
      return images.ToArray();
    }

    void updateTransformation() {
      if (Parent != null)
        this.Transformation.Distort(Parent.Transformation);
    }
  }
}
```

Класс Panoramas.TreeBuilder.Builder:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Panoramas.Processors;

namespace Panoramas.TreeBuilder {
  public class Builder : IBuilder {
    IFactory factory;
    List<IImage> loose_images;
    List<IImageTransformed> segments;

    public Builder(IFactory factory) {
      this.factory = factory;
      this.segments = new List<IImageTransformed>();
    }

    public IPanoramaTransformations Build(IPanoramaRelations panorama) {
      this.segments.Clear();
      this.loose_images = panorama.Images.ToList();
      var root = addNodeToTree(panorama, panorama.Core());
      while (loose_images.Count > 0) {
        IImage closest_loose_image, closest_tree_image;
        var registered = panorama.Images.Except(loose_images);
        panorama.ClosestBetween(registered.ToArray(), loose_images.ToArray(),
out closest_loose_image, out closest_tree_image);
        var closest_tree_node = root.FindNode(closest_tree_image);
        addNodeToTree(panorama, closest_loose_image, closest_tree_node);
      }
      return factory.PanoramaTransformations(panorama, segments.ToArray());
```

```
    }

    TreeNode addNodeToTree(IPanoramaRelations panorama, IImage image,
TreeNode parent = null) {
      TreeNode node = null;
      if (parent != null) {
        var transformation = panorama.MatchBetween(parent.Image,
image).GenerateTransformation();
        node = parent.AddChild(image, transformation);
      } else {
        node = new TreeNode(image, factory.Transformation());
      }
      loose_images.Remove(image);
      segments.Add(factory.ImageTransformed(node.Image,
node.Transformation));
      var neighbours = panorama.NeighboursOf(image, loose_images.ToArray());
      foreach (var neighbour in neighbours) {
        addNodeToTree(panorama, neighbour, node);
      }
      return node;
    }
  }
}
```

Модули расширения PanoramasBoundsPresenter.
Класс Panoramas.BoundsPresenter.Presenter:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Panoramas;

namespace Panoramas.BoundsPresenter
{
    public class Presenter: Panoramas.Processors.IPresenter
    {
      IFactory factory;

      public Presenter(IFactory factory) {
        this.factory = factory;
      }

      public Bitmap Present(IPanoramaTransformations panorama) {
        PointF offset_vector;
        var template = generateTemplate(panorama, out offset_vector);
        var offset = factory.Transformation();
        offset.Move((int)offset_vector.X, (int)offset_vector.Y);
        foreach (var segment in panorama.Segments)
          template = renderSegment(segment, offset, template);
        return template;
      }

      Bitmap renderSegment(IImageTransformed segment, ITransformation
context, Bitmap template) {
        var transformation = context.Multiply(segment.Transformation);
        return transformation.TransformOn(segment.Bitmap, template);
      }
```

```
      Bitmap generateTemplate(IPanoramaTransformations panorama, out PointF
offset) {
        List<PointF> bounds = new List<PointF>();
        foreach (var image in panorama.Segments) {
          var size = image.Bitmap.Size;
          List<PointF> corners = new List<PointF>();
          corners.Add(new PointF(0, 0));
          corners.Add(new PointF(0, size.Height));
          corners.Add(new PointF(size.Width, 0));
          corners.Add(new PointF(size.Width, size.Height));
          var transformed_corners = corners.Select((c) =>
image.Transformation.Transform(c));
          bounds.AddRange(transformed_corners);
        }
        float min_x = bounds.Min((b) => b.X);
        float max_x = bounds.Max((b) => b.X);
        float min_y = bounds.Min((b) => b.Y);
        float max_y = bounds.Max((b) => b.Y);
        int width = (int)((max_x - min_x) * 1.5);
        int height = (int)((max_y - min_y) * 1.5);
        var template = new Bitmap(width, height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
        offset = new PointF(-min_x, -min_y);
        return template;
      }
    }
}
```

Модули расширения PanoramasFeaturedTree.
Класс Panoramas.FeaturedTree.PrivateFactory:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Panoramas.Processors;

namespace Panoramas.FeaturedTrees {
  public class PrivateFactory : Panoramas.Defaults.DefaultsFactory {
    public override ITransformation Transformation() {
      return new Panoramas.HomographyTransformer.Transformation();
    }

    public override IAnalyzer Analyzer() {
      return new Panoramas.FeaturesAnalyzer.Analyzer(this);
    }

    public override IBuilder Builder() {
      return new Panoramas.TreeBuilder.Builder(this);
    }

    public override IPresenter Presenter() {
      return new Panoramas.BoundsPresenter.Presenter(this);
    }
  }
}
```

Класс Panoramas.FeaturedTree.Factory:

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Panoramas.FeaturedTrees {
  public class Factory {
    public static IPublicFactory Generate() {
      return new PrivateFactory();
    }
  }
}
```

Приложение В
(обязательное)
Исходный код основных классов приложения

Класс StitcherApp.MainForm:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace TransformatorExample {
  public partial class MainForm : BaseForm {
    Panoramas.IPublicFactory panoramas_factory;
    Panoramas.IStitcher stitcher;
    Panoramas.IRelationControl current_match;
    ImageEditor.Editor picturebox_matching, picturebox_merging;
    ImageFilesManager.CollectionManager images_manager;
    ImageFilesManager.ISelectableControl segments_thumbnails;
    ImageFilesManager.ISelectableControl segments_pair_list;
    bool stitched = false;

    TabPage page_matching, page_stitching;

    public MainForm(Panoramas.IPublicFactory factory) {
      InitializeComponent();
      this.panoramas_factory = factory;
      picturebox_matching = new ImageEditor.Editor(this,
this.pictureMatches);
      picturebox_merging = new ImageEditor.Editor(this, this.pictureMerged);
      picturebox_merging.BackgroundColor = Color.Black;
      images_manager = new ImageFilesManager.CollectionManager();
      this.segments_thumbnails =
images_manager.PresentAsListView(imagesContainer);
      this.segments_thumbnails.AddSelectionChangeHandler(new
EventHandler(this.currentFilesSelection_Change));
      this.segments_pair_list =
images_manager.PresentAsPairsList(listSegmentsMatchLeft,
listSegmentsMatchRight);
      this.segments_pair_list.AddSelectionChangeHandler(new
EventHandler(this.currentMatch_Change));
      initToolTips();
      page_matching = tabPageMatching;
      page_stitching = tabPageMerging;
      tabControlMain.TabPages.Remove(tabPageMatching);
      tabControlMain.TabPages.Remove(tabPageMerging);
      updateLoadingPageStatus();
    }

    void initToolTips() {
      setToolTip(this.buttonAddSegments, "Open (Ctrl+O)", "Add more images to
stitch them into a panoram");
      setToolTip(this.buttonRemoveSelectedFiles, "Remove (Del)", "Remove
selected images");
```

```
      setToolTip(this.buttonClearSegment, "Remove images", "Clear images
list");
      setToolTip(this.buttonSavePan, "Save (Ctrl+S)", "Save current panorama
image");
    }

    const int TOOLTIP_DELAY = 1500;
    void setToolTip(Control control, String title, String text = null) {
      ToolTip tooltip = new ToolTip();
      tooltip.InitialDelay = TOOLTIP_DELAY;
      tooltip.IsBalloon = true;
      tooltip.ToolTipTitle = title;
      tooltip.SetToolTip(control, text);
    }

    delegate void a_process();
    void waitForProcessComplete(a_process process) {
      this.Cursor = Cursors.WaitCursor;
      process.Invoke();
      this.Cursor = Cursors.Default;
    }

    //
    // LOADING IMAGES
    //

    private void buttonAddSegments_Click(object sender, EventArgs e) {
      addSegments();
    }

    private void buttonClearSegment_Click(object sender, EventArgs e) {
      images_manager.ClearAll();
      updateLoadingPageStatus();
    }

    private void buttonClearFiles_Click(object sender, EventArgs e) {
      removeSelectedSegments();
    }

    private void imagesContainer_KeyDown(object sender, KeyEventArgs e) {
      if (e.KeyData == Keys.Delete) {
        removeSelectedSegments();
      }
    }

    private void buttonGotoMatching_Click(object sender, EventArgs e) {
      Logger.Logger.LogTime("Matching", () => {
        updateStitcher();
        resetCurrentMatch();
        if (stitched) {
          tabControlMain.SelectedTab = tabPageMatching;
        } else {
          generatePanoram();
          tabControlMain.TabPages.Add(tabPageMatching);
          tabControlMain.TabPages.Add(tabPageMerging);
          tabControlMain.SelectedTab = tabPageMerging;
          stitched = true;
        }
      });
    }

    void addSegments() {
      images_manager.LoadMore();
```

```
      updateLoadingPageStatus();
    }

    void removeSelectedSegments() {
      var selection = segments_thumbnails.SelectedItems();
      images_manager.Remove(selection);
      updateLoadingPageStatus();
    }

    void updateLoadingPageStatus() {
      bool images_present = images_manager.Images.Count > 0;
      buttonClearSegment.Enabled = images_present;
      bool can_stitch = images_manager.Images.Count >= 2;
      scrollLimit.Enabled = can_stitch;
      buttonGotoMatching.Enabled = can_stitch;
      if (!can_stitch) {
        unmarkButton(buttonGotoMatching);
        markButton(buttonAddSegments);
      } else {
        markButton(buttonGotoMatching);
        unmarkButton(buttonAddSegments);
      }
      updateFileSelectionStatus();
    }

    void updateStitcher() {
      waitForProcessComplete(() => {
        this.stitcher = panoramas_factory.Stitcher(
          images_manager.Images.Select((i) => i.FileName).ToArray(),
          images_manager.Images.Select((i) => i.Bitmap).ToArray());
        buttonGotoMerge.Enabled = true;
      });
    }

    void currentFilesSelection_Change(object sender, EventArgs e) {
      updateFileSelectionStatus();
    }

    void updateFileSelectionStatus() {
      bool selection_present = segments_thumbnails.SelectedItems().Length >
0;
      buttonRemoveSelectedFiles.Enabled = selection_present;
    }

    //
    // MATCHING
    //

    private void scrollLimit_Scroll(object sender, ScrollEventArgs e) {
      current_match.LimitPercent = scrollLimit.Value;
      drawCurrentMatch();
    }

    private void buttonResetMathPicture_Click(object sender, EventArgs e) {
      picturebox_matching.ResetState();
    }

    private void checkBoxActiveMatch_CheckedChanged(object sender, EventArgs
e) {
      current_match.Active = checkBoxActiveMatch.Checked;
      drawCurrentMatch();
    }
```

```
    void currentMatch_Change(object sender, EventArgs e) {
      resetCurrentMatch();
    }

    private void buttonBackToFiles_Click(object sender, EventArgs e) {
      tabControlMain.SelectedTab = tabPageSegments;
    }

    private void buttonGotoMerge_Click(object sender, EventArgs e) {
      generatePanoram();
    }

    void resetCurrentMatch() {
      var selection = segments_pair_list.SelectedItems();
      if (stitcher == null || selection.Any((i) => i == null))
        return;
      current_match = stitcher.MatchBetween(selection[0], selection[1]);
      drawCurrentMatch();
    }

    void drawCurrentMatch() {
      picturebox_matching.Image = current_match.ToImage();
      textMatchDistance.Text = current_match.Similarity().ToString("F2");
      scrollLimit.Value = current_match.LimitPercent;
      checkBoxActiveMatch.Checked = current_match.Active;
    }

    void generatePanoram() {
      waitForProcessComplete(() => {
        picturebox_merging.Image = stitcher.StitchAll();
        tabControlMain.SelectedTab = tabPageMerging;
        buttonSavePan.Enabled = true;
        markButton(buttonSavePan);
      });
    }

    //
    // MERGING
    //

    private void buttonSavePan_Click(object sender, EventArgs e) {
      savePanorama();
    }

    private void buttonBackToMatching_Click(object sender, EventArgs e) {
      tabControlMain.SelectedTab = tabPageMatching;
    }

    void savePanorama() {
      var dialog = new ImageFilesManager.Dialog();
      dialog.SaveToFile((filename) => {
        picturebox_merging.Image.Save(filename);
      });
    }

    //
    // HOTKEYS
    //

    protected override bool ProcessCmdKey(ref Message msg, Keys keyData) {
      if (tabControlMain.SelectedTab == tabPageSegments && keyData ==
openFileShortCut)
        addSegments();
```

```
      else if (tabControlMain.SelectedTab == tabPageMerging && keyData ==
saveFileShortCut)
        savePanorama();
      return base.ProcessCmdKey(ref msg, keyData);
    }

    Keys openFileShortCut {
      get { return Keys.Control | Keys.O; }
    }

    Keys saveFileShortCut {
      get { return Keys.Control | Keys.S; }
    }

    //
    // BUTTONS
    //

    void markButton(Button button) {
      button.Font = new Font(button.Font, FontStyle.Bold);
      button.Focus();
    }

    void unmarkButton(Button button) {
      button.Font = new Font(button.Font, FontStyle.Regular);
    }
  }
}
```

Класс StitchingApp.Program:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace TransformatorExample {
  static class Program {
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main() {
      Application.EnableVisualStyles();
      Application.SetCompatibleTextRenderingDefault(false);
      var panoramas_factory = Panoramas.FeaturedTrees.Factory.Generate();
      Application.Run(new MainForm(panoramas_factory));
    }
  }
}
```