**3 Проектирование программного средства**

**3.1** Разработка архитектуры

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** Среда работы 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.

Существуют устоявшиеся модели распределения и взаимодействия компонентов в приложении. Использовать их целесообразно при разработке больших программ, при проектировке которых велик риск создания неэффективной архитектуры. Одна из таких моделей – EBC (Entity-Boundary- Control, [X]). Она определяет три категории элементов:

- 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 будет использована для конкретизации категорий элементов. 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 «ПредставительПанорамы».

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



Рисунок 3.1 – Аналитическая модель программного средства

**3.2** Проектирование пользовательского интерфейса

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 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 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** Разработка основных алгоритмов программы

**3.3.1** Алгоритм работы программы

Программа начинает свое выполнение после 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 приведен на рисунке 3.2.



Рисунок 3.2 – Схема работы программы

**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 характеризуется неопределенностью. 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 подразделе 1.2.3 были упомянуты два метода: аккумулирующая регистрация и регистрация с помощью распределения общей ошибки. Под ошибкой здесь понимается разница между корректным и вычисленным расположением одного сегмента относительно другого. Первый из методов них 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, целесообразной является доработка вышеуказанных методов.

В разрабатываемом ПС используется 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 его пути до корня. Это дает возможность выбирать только те связи между сегментами, что производят наименьшую ошибку. На практике ошибка напрямую зависит от схожести сегментов, т.е. степени их перекрытия и специфики области перекрытия. Чем более схожи изображения, тем большей точностью характеризуется 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. <…>



Рисунок 3.3 – Схема алгоритма генерации панорамы



Рисунок 3.4 – Алгоритм регистрации сегмента в панораме