ВВЕДЕНИЕ

Дистанционное зондирование Земли – это способ получения информации об объекте без непосредственного физического контакта с ним. На борту летательного аппарата (например, спутника либо самолёта) устанавливается спектрометр, задачей которого является фиксация излучения с поверхности, затем бортовая система осуществляет предобработку полученных данных и передаёт их в центр приёма информации. При этом в зависимости от типа спектрометра рабочий диапазон длин волн, может составлять от долей микрометра до метров.

В зависимости от типа спектрометра различают мультиспектральные и гиперспектральные данные. Основное отличие в том, что гиперспектральные спектрометры фиксируют данные в виде непрерывного диапазона спектра с определённым шагом, в то время как мультиспектральные данные могут иметь такое же количество каналов данных, но распределенных в спектральном диапазоне неравномерно.

Тенденции развития дистанционного зондирования показывают, что акцент в исследованиях смещается в область гиперспектральной съёмки. Однако существует несколько факторов препятствующих их широкому распространению: отсутствие достаточного количества воздушных судов, оборудованных соответствующими спектрометрами; проблемы связанные с обработкой и интерпретацией больших потоков информации, формируемых этими приборами. В связи с этим особенно остро стоит вопрос о создании спутниковой гиперспектральной аппаратуры и технологий обработки получаемой с помощью нее информации на борту летательного аппарата.

Данные, передаваемые с летательного аппарата в центр приёма, представляют собой трёхмерный куб, характеризующийся следующими разрешениями: пространственным (определяет площадь поверхности); спектральным (определяет охватываемый спектральный диапазон); радиометрическим (определяет число уровней сигнала, которые сенсор может зарегистрировать).

Подобная структура, с учётом непрерывного спектрального диапазона, приводит к формированию существенного объёма данных передаваемых на Землю и актуализирует задачу сжатия. Например, данные спектрометра AVIRIS, которые используются для разработки алгоритмов и программного обеспечения для обработки гиперспектральных снимков, имеют следующие характеристики: ширина изображения 677 пикселов, 224 спектральных канала, 12 бит на канал, что в общем случае приводит к 222,1 Кб данных на строку. С учётом характеристик современных радиоканалов связи и того, что съёмка ведётся без остановки, важнейшими требованиями к алгоритмам сжатия данных являются высокий коэффициента сжатия данных и низкие требования к вычислительной мощности алгоритма, что связано с техническими ограничениями летательных аппаратов. Цель данного дипломного проекта – выбрать наиболее подходящий алгоритм и разработать программный комплекс для

сжатия гиперспектральных данных, который должен иметь высокий коэффициент сжатия данных при минимальных потерях качества, а также обладать низкой вычислительной сложностью, простотой в использовании.

В соответствии с поставленной целью были поставлены следующие задачи:

- обзор существующих алгоритмов сжатия;
- разработка структурной схемы системы
- выбор алгоритма подходящего по требованиям;
- реализация программного комплекса;
- реализация модулей поддержки и визуализации.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор аналогов

Гиперспектральное изображение – это трехмерный массив данных (куб данных), который включает в себя пространственную информация (2D) об объекте, дополненную спектральной информацией (1D) по каждой пространственной координате. Иными словами, каждой точке изображения соответствует спектр, полученный в этой точке снимаемого объекта.

До появления техники записи гиперспектральных изображений, для получения информации об объекте (участке местности) использовались мультиспектральные изображения, то есть наборы фотографий, полученные с помощью цветных светофильтров. Изначально такой подход, использующий пространственную картину спектрального распределения, применялся лишь для дистанционного зондирования окружающей среды и распознавания боевых целей. Однако, в настоящее время он широко используется в области биомедицинской оптики. Гиперспектральные визоры вскоре станут одной из основных технологий в медицинской сфере.

Выделяют два подхода к сжатию гиперспектральных данных: с применением общеизвестных методик сжатия и с адаптацией алгоритма под заданные условия.

В рамках первого подхода чаще всего используются алгоритмы сжатия без потерь и почти без потерь (когда потери информации не превышают уровень шума, вносимого используемым спектрометром). Они разделяются на следующие основные классы, главное различие которых сводится к аппаратным ресурсам: на основе предсказания (linear prediction(LP), fast lossless(FL), spectral oriented least squares (SLSQ), correlation-based condition average prediction(CCAP), M-CALIC); поиск по таблице (lookup table (LUT), locally averaged interband scalling lookup tables (LAIS-LUT)); вейвлеты (3D-SPECK, Dual Tree BEZW, 3D-SPIHT).

В алгоритмах сжатия, имеющих в основе алгоритмов предсказания выделяется некоторая окрестность, над которой выполняется математическое действие(предсказание). Результат предсказания вычитается из оригинального значения и формируется ошибка предсказания, которая передаётся в блок энтропийного кодирования, результатом которого является сжатый поток данных. Восстановление осуществляется в обратной последовательности. В качестве алгоритма кодирования могут использоваться, например, коды Голомба-Райса или любой арифметический кодек, допускающий аппаратную реализацию. Основным недостатком многих алгоритмов предсказания является высокая вычислительная нагрузка при небольшом использовании оперативной памяти.

Алгоритмы на основе поиска по таблице обеспечивают ускорение процесса вычисления, основанное на существенности корреляции между спектральными каналами. Размерность таблицы – число спектральных каналов,

умноженное на максимально допустимое значение при данном радиометрическом разрешении. По текущему значению пиксела делается запрос в таблицу, и возвращаемое значение считается предсказанным. Дальнейшая обработка эквивалентна алгоритмам предсказания.

Алгоритмы на основе дискретного вейвлет-преобразования являются наиболее требовательными ко всем вычислительным ресурсам. Данный класс предполагает предварительный перевод спектральной плоскости в частотную область. После этого возможно организовать обработку таким образом, чтобы система кодировала в первую очередь наиболее значимые вейвлет-коэффициенты, постепенно смещаясь в область с наименее значимыми коэффициентами. Такой подход позволяет реализовать как сжатие без потерь (при обработке всех вейвлет-коэффициентов), так и управляемое сжатие с потерями.

Главный недостаток алгоритмов этого класса – вычислительная сложность, связанная с преобразованием в частотную область куба данных и требования к пропускной способности памяти из-за случайных переходов в памяти от низкочастотных к высокочастотным вейвлет-коэффициентам.

Второй подход основан на существенной избыточности получаемых данных и связан с большим спектральным разрешением. Алгоритмы данного класса основываются на следующих упрощениях: 1) условия съемки заведомо известны; при таком подходе появляется возможность на борту летательного аппарата удалить неинформативные каналы (например, учесть влияние атмосферы) либо, наоборот, выделить наиболее информативные, т.е. в любом случае получить мультиспектральные данные;

2) выполнение полного либо частичного анализа полученных данных и передача результата, а не самих данных.

Достоинством алгоритмов, реализующих данный подход является передача только необходимых данных и существенное понижение объёма передаваемых данных. Тем не менее алгоритмы практически не реализуемы на борту летательного аппарата из-за их вычислительной сложности.

1.1.1 Алгоритм сжатия 3D-SPECK

Алгоритм 3D-SPECK (three dimensional Set Partitioning Embedded bloCK) это алгоритм сжатия без потерь (lossless). Трехмерное вейвлет-преобразование использует межполосную корреляцию объёмных блоков изображения. Трёхмерная структура этого алгоритма включает использование межполосных зависимостей и корреляций. Так же алгоритм 3D-SPECK поддерживает расщепление блоков для сортировки важных пикселов, то есть если блок кода содержит важные коэффициенты он разделяется на небольшие подблоки. Также, если гиперспектральное изображение содержит концентрацию энергии в высокочастотных каналах, алгоритм должен хорошо справиться с таким изображением.

3D-SPECK включает два связанных списка: список незначительных наборов и список значимых пикселов. Процесс обработки состоит из четырёх

этапов: этапа инициализации, сортировки, уточнения и квантования. Во время этапа сортировки метод разделения блоков принимается после теста на значимость. На этапе уточнения некоторые коэффициенты передаются и процесс квантование продолжается для следующей уменьшенной битовой плоскости, пока не будет достигнута приемлемая скорость передачи данных. Преобразованный коэффициент трёхмерного дискретного вейвлет-преобразования декоррелирует пространственные и спектральные компоненты гиперспектрального изображения. Это выявляет некоторую избыточность, которая может быть использована во время процесса кодирования. Кодировщик следует за основным алгоритмом сортировки, как в 3D-SPECK. Поэтому межзональную зависимость можно использовать автоматически. Для того чтобы сопоставить трёхмерные коэффициенты с одномерным массивом, используется рекурсивная Z-последовательность или кривая Мортона. Кривая Мортона – это функция которая отображает многомерные данные в одномерные, сохраняя локальность точек данных.

После линейной системы индексирования количество коэффициентов сохраняется в одном массиве длинной I, где I = строки * столбцы * фреймы, и является массивом величин. Массив таблицы состояний, основанный на линейной индексации, также имеет длину I с 4 битами на коэффициент, который является отмеченной частью. Между величиной и меткой существует взаимно однозначное соответствие. В этом алгоритме так же устранена проблема вейвлет-преобразований связанная с избыточным потреблением памяти.

1.1.2 Алгоритм сжатия 3D-SPIHT

Алгоритм сжатия 3D-SPIHT имеет в основе дискретное вейвлет преобразование. Степень сжатия при использовании данного алгоритма сильно зависит от самого изображения (снимаемая территория, наличие шумов).

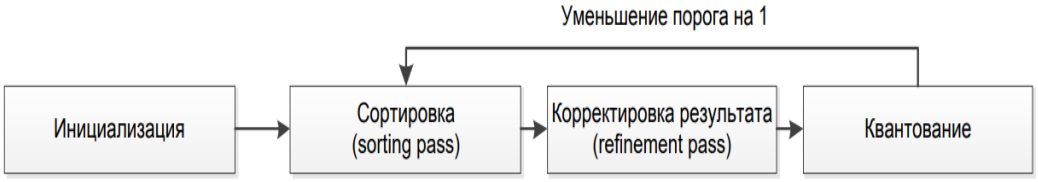

Рисунок 1.1 – Общий вид алгоритма 3D-SPIHT

На этапе инициализации в список несущественных наборов (LIS) помещаются коэффициенты от вейвлет-преобразования подмножеств изображения для текущего уровня декомпозиции. После этого осуществляется анализ элементов в наборе LIS и сравнение результата с пороговым значением n . При прохождении порогового значения осуществляется разбиение анализируемого элемента на 8 эквивалентных подмножеств и процедура сортировки повторя-

ется до тех пор, пока не будет найдено достоверное значение пикселя из оригинального множества. Достоверное значение пикселя переносится в массив LSP (список существенных пикселей), а из множества LIS удаляется. После окончания сортировки осуществляется дополнительная корректировка и квантование данных. Дополнительная корректировка в зависимости от реализации включает обработку текущего пикселя и сравнения его с текущим пороговым значением. В результате этого сравнения принимается решение о формировании соответствующего набора битов в выходной поток. После этого пороговое значение n уменьшается на 1 и осуществляется возврат на этап сортировки. Полученный массив данных после работы алгоритма является сжатым гиперспектральным изображением. Общий вид алгоритма представлен выше (см. рисунок 1.1)

1.1.2 Алгоритм сжатия 3D-BEZW

Алгоритм сжатия 3D-BEZW это алгоритм сжатия изображения с потерями. При низких скоростях передачи, то есть при высоких коэффициентах сжатия, большинство коэффициентов, создаваемых преобразование поддиапазона (например, вейвлет-преобразование), будет равно нулю или очень близка к нулю. Это происходит потому что изображения «реального мира», как правило, содержат в основном низкочастотную информацию (высоко коррелируют). Однако там, где находится высокочастотная информация (например, грани на изображении), это особенно важно для человека с точки зрения восприятия качества картинки, и поэтому должно быть точно представлено в любой высококачественной схеме кодирования. Рассматривая преобразованные коэффициенты как дерево (или деревья) с наименьшими частотными коэффициентами в корневом узле и с дочерними элементами каждого узла дерева являются пространственно связанными коэффициентами в следующем более высокочастотном поддиапазоне, существует высокая вероятность того, что один или несколько поддеревья будут состоять полностью из коэффициентов, которые равны нулю или почти равны нулю, такие поддеревья называются нулевыми. В связи с этим мы используем термины узел и коэффициент взаимозаменяемо, а когда мы обращаемся к детям коэффициента, мы имеем в виду дочерние коэффициенты узла в дереве, где этот коэффициент расположен. Мы используем дочерние элементы, чтобы ссылаться на непосредственно связанные узлы ниже в дереве и потомках, чтобы ссылаться на все узлы, которые находятся ниже определенного узла в дереве, даже если они не связаны напрямую. В схеме сжатия изображений на основе нулевого дерева, такой как EZW и SPIHT, целью является использование статистических свойств деревьев для эффективного кодирования местоположений значимых коэффициентов. Поскольку большинство коэффициентов будут равны нулю или близки к нулю, пространственные местоположения значимых коэффициентов составляют значительную часть общего размера типичного сжатого изображения. Коэф-

фициент (аналогично дереву) считается значимым, если его величина (или величины узла и всех его потомков в случае дерева) превышает определенный порог. Начав с порога, близкого к максимальным значениям коэффициента и итеративно его уменьшая, можно создать сжатое представление изображения, которое постепенно добавляет более мелкие детали. Из-за структуры деревьев весьма вероятно, что если коэффициент в конкретной полосе частот незначителен, то все его потомки (пространственно связанные коэффициенты более высокой полосы частот) также будут значительными.

EZW использует четыре символа для представления: корень нулей, изолированный ноль (коэффициент, который незначителен, но имеющий значительные потомки), значительный положительный коэффициент и значительный отрицательный коэффициент. Таким образом, символы могут быть представлены двумя двоичными битами. Алгоритм сжатия состоит из нескольких итераций через доминирующий проход и подчиненный проход, порог обновляется (уменьшается в два раза) после каждой итерации.

Доминирующий проход кодирует значение коэффициентов, которые еще не были признаны значимыми в предыдущих итерациях, путем сканирования деревьев и испускания одного из четырех символов. Дети коэффициента только сканируются, если коэффициент был признан значимым, или если коэффициент был изолированным нулем. Субординированный проход испускает один бит (самый старший бит каждого коэффициента, который до сих пор не испускался) для каждого коэффициента, который был признан значительным в предыдущих пропусков значимости. Таким образом, подчиненный проход аналогичен кодированию битовой плоскости.

Есть несколько важных особенностей, которые следует отметить. Во-первых, в любой момент можно остановить алгоритм сжатия и получить приближение исходного изображения, чем больше количество полученных бит, тем лучше изображение. Во-вторых, из-за того, что алгоритм сжатия структурирован как ряд решений, один и тот же алгоритм может быть запущен в декодере для восстановления коэффициентов, но с решениями, принимаемыми в соответствии с входящим потоком битов.

В практических реализациях было бы обычно использовать энтропийный код, такой как арифметический код, для дальнейшего улучшения производительности доминирующего прохода. Биты из подчиненного прохода обычно являются случайными, что энтропийное кодирование не дает дополнительного усиления кодирования.

Алгоритм, основанный на преобразовании, сканирует преобразованное изображение несколько раз и кодирует значимые коэффициенты по отношению к нескольким пороговым значениям. Алгоритм эффективен, поскольку он основан на неперекрывающихся асимметричных древовидных структурах, которые интерполируют отношения между вейвлет-коэффициентами в разных масштабах поддиапазонов. Естественно, дерево должно быть спроектировано в соответствии со свойствами трансформированного изображения, так что ветви дерева к более высокочастотным поддиапазонам происходят в одной и

той же пространственной ориентации; другими словами, коэффициенты корней должны быть увеличены из низкочастотных поддиапазонов.

1.2 Обзор спектрометра AVIRIS

AVIRIS (Airborn Visible and InfraRed Imaging Spectrometer) – бортовой спектрометр видимого и инфракрасного диапазонов.

AVIRIS – это проверенный инструмент в области дистанционного зондирования Земли. Это уникальный оптический датчик, который обеспечивает калиброванные изображения спектрального излучения в 224 смежных спектральных каналах (диапазонах) с длиной волны от 400 до 2500 нанометров. AVIRIS установлен на четырёх авиационных платформах: самолет ER-2 NASA, турбовинтовой двигатель Twin Otter International, Proteus Scaled Composites и WB-57 НАСА. ER-2 летит примерно на 20 км над уровнем моря, около 730 км / час. Самолет Twin Otter летает на высоте 4 км над уровнем земли со скоростью 130 км / час. AVIRIS пролетел в Северной Америке, Европе, частях Южной Америки и Аргентины.

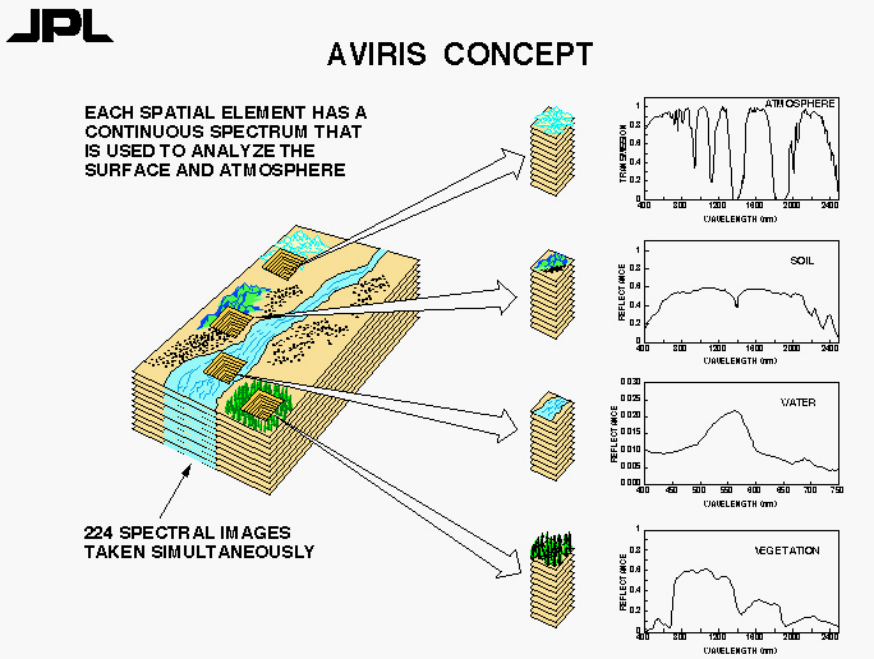

Рисунок 1.2 – Иллюстрация измерительных возможностей прибора AVIRIS[1]

Основной целью проекта AVIRIS является выявление, измерение и мониторинг составляющих земной поверхности и атмосферы на основе сигналов молекулярного поглощения и рассеяния частиц. Исследования с данными AVIRIS в основном сосредоточены на понимании процессов, связанных с глобальной окружающей средой и изменением климата.

Инструмент AVIRIS содержит 224 (см. рисунок 1.2) различных детекторов, каждый из которых имеет диапазон чувствительности к длине волны

(также известный как спектральная пропускная способность), приблизительно 10 нанометров (нм), что позволяет покрывать весь диапазон между 380 нм и 2500 нм. Когда данные из каждого детектора изображаются на графике, он дает полный спектр VIS-NIR-SWIR. Сравнение полученного спектра с показателями известных веществ показывает информацию о составе области, рассматриваемой прибором.

Характеристики спектрометра AVIRIS

– Скорость передачи данных 17 Мбит/с до 1994 года, 20,4 Мбит/с с 1995 по 2004 год, 16 бит с 2005 года;
– 10-битная кодировка данных в 1994 году, 12-битная с 1995 года;
– Скорость сканирования 12 Гц;
– Детекторы, охлаждаемые жидким азотом;
– 10 нанометров номинальная пропускная способность канала, калиброванная с точностью до 1 нанометра;
– 34 градуса общего поля зрения;
– Мгновенное поле зрения 1 миллирадиан, калиброванный с точностью до 0,1 мрад
– Сканирование «Whisk broom»
– 76 Гигабайт на носителе для записи;
– Кремниевые детекторы для видимого диапазона

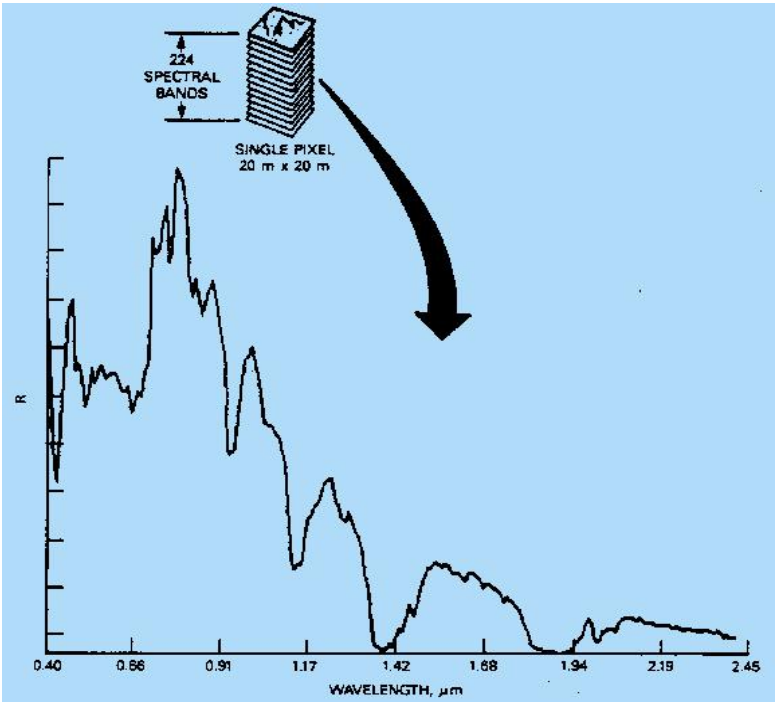

Рисунок 1.3 – Пример спектра одного пиксела от AVIRIS [2]

Выше приведен пример спектра одного пиксела от AVIRIS (см. рисунок 1.3). Ось X представляет собой длину волны канала в микрометрах, также известную как микрон (один микрон = 1000 нанометров). Ось y – это сияние,

обычно выраженное в единицах микроватт на квадратный сантиметр на нанометр на стерадиан.

1.3 Обзор языка программирования C++

Для реализации программного комплекса, из-за поставленных задач был выбран язык программирования C++. C++ – компилируемый строго типизированный язык. Поддерживает разные парадигмы программирования: процедурную, обобщённую, функциональную.

Наибольшее внимание уделено поддержке объектно-ориентированного программирования.

Своими корнями он уходит в язык Си, который был разработан в 1969-1973 годах в компании Bell Labs программистом Деннисом Ритчи (Dennis Ritchie). В начале 1980-х годов датский программист Бьерн Страуструп (Bjarne Stroustrup), который в то время работал в компании Bell Labs, разработал C++ как расширение к языку Си. Фактически вначале C++ просто дополнял язык Си некоторыми возможностями объектно-ориентированного программирования. И поэтому сам Страуструп вначале называл его как "C with classes" ("Си с классами").

C++ является мощным языком, унаследовав от Си богатые возможности по работе с памятью. Поэтому нередко C++ находит свое применение в системном программировании, в частности, при создании операционных систем, драйверов, различных утилит, антивирусов и т.д. К слову сказать, ОС Windows большей частью написана на C++. Но только системным программированием применение данного языка не ограничивается. C++ можно использовать в программах любого уровня, где важны скорость работы и производительность. Нередко он применяется для создания графических приложений, различных прикладных программ.

Также особенно часто его используют для создания игр с богатой насыщенной визуализацией. Кроме того, в последнее время набирает ход мобильное направление, где C++ тоже нашел свое применение. И даже в веб-разработке также можно использовать C++ для создания веб-приложений или каких-то вспомогательных сервисов, которые обслуживают веб-приложения. В общем C++ – язык широкого пользования, на котором можно создавать практически любые виды программ.

C++ является компилируемым языком, а это значит, что компилятор транслирует исходный код на C++ в исполняемый файл, который содержит набор машинных инструкций. Но разные платформы имеют свои особенности, поэтому скомпилированные программы нельзя просто перенести с одной платформы на другую и там уже запустить. Однако на уровне исходного кода программы на C++ по большей степени обладают переносимостью, если не используются какие-то специфичные для текущей ос функции. А наличие компиляторов, библиотек и инструментов разработки почти под все распространенные платформы позволяет компилировать один и тот же исходный код на

C++ в приложения под эти платформы.

1.4 Общее описание алгоритмов сжатия.

Существует несколько классических подходов к сжатию изображений. Среди этих алгоритмов:

- Алгоритм LRE;
- Словарные Алгоритмы;
- Алгоритмы статического кодирования;
- Алгоритм Хаффмана;
- Арифметическое кодирование.

Все алгоритмы серии RLE основаны на очень простой идее: повторяющиеся группы элементов заменяются на пару (количество повторов, повторяющийся элемент). Рассмотрим этот алгоритм на примере последовательности бит. В этой последовательности будут чередовать группы нулей и единиц. Причём в группах зачастую будет более одного элемента. Тогда последовательности 11111 000000 11111111 00 будет соответствовать следующий набор чисел 5 6 8 2. Эти числа обозначают количество повторений (отсчёт начинается с единиц), но эти числа тоже необходимо кодировать. Будем считать, что число повторений лежит в пределах от 0 до 7 (т.е. нам хватит 3 бит для кодирования числа повторов). Тогда рассмотренная выше последовательность кодируется следующей последовательностью чисел 5 6 7 0 1 2. Легко подсчитать, что для кодирования исходной последовательности требуется 21 бит, а в сжатом по методу RLE виде эта последовательность занимает 18 бит. Хоть этот алгоритм и очень прост, но эффективность его сравнительно низка. Более того, в некоторых случаях применение этого алгоритма приводит не к уменьшению, а к увеличению длины последовательности. Для примера рассмотрим следующую последовательность 111 0000 11111111 00. Соответствующая ей RL-последовательность выглядит так: 3 4 7 0 1 2. Длина исходной последовательности – 17 бит, длина сжатой последовательности – 18 бит. Этот алгоритм наиболее эффективен для чёрно-белых изображений. Также он часто используется, как один из промежуточных этапов сжатия более сложных алгоритмов.

Идея, лежащая в основе словарных алгоритмов, заключается в том, что происходит кодирование цепочек элементов исходной последовательности. При этом кодировании используется специальный словарь, который получается на основе исходной последовательности. Существует целое семейство словарных алгоритмов, но мы рассмотрим наиболее распространённый алгоритм LZW, названный в честь его разработчиков Лепеля, Зива и Уэлча. Словарь в этом алгоритме представляет собой таблицу, которая заполняется цепочками кодирования по мере работы алгоритма. При декодировании сжатого кода словарь восстанавливается автоматически, поэтому нет необходимости передавать словарь вместе с сжатым кодом. Словарь инициализируется всеми одноэлементными цепочками, т.е. первые

строки словаря представляют собой алфавит, в котором мы производим кодирование. При сжатии происходит поиск наиболее длинной цепочки уже записанной в словарь. Каждый раз, когда встречается цепочка, ещё не записанная в словарь, она добавляется туда, при этом выводится сжатый код, соответствующий уже записанной в словаре цепочки. В теории на размер словаря не накладывается никаких ограничений, но на практике есть смысл этот размер ограничивать, так как со временем начинаются встречаться цепочки, которые больше в тексте не встречаются. Кроме того, при увеличении размеры таблицы вдвое мы должны выделять лишний бит для хранения сжатых кодов. Для того чтобы не допускать таких ситуаций, вводится специальный код, символизирующий инициализацию таблицы всеми одноэлементными цепочками.

Алгоритмы статического кодирования ставят наиболее частым элементам последовательностей наиболее короткий сжатый код. Т.е. последовательности одинаковой длины кодируются сжатыми кодами различной длины. Причём, чем чаще встречается последовательность, тем короче, соответствующий ей сжатый код.

Алгоритмы арифметического кодирования кодируют цепочки элементов в дробь. При этом учитывается распределение частот элементов. На данный момент алгоритмы арифметического кодирования защищены патентами, поэтому мы рассмотрим только основную идею.

Пусть наш алфавит состоит из N символов от a1 до aN, а частоты их появления от p1до pN соответственно. Разобьем полуинтервал [0;1) на N непересекающихся полуинтервалов. Каждый полуинтервал соответствует элементам ai, при этом длина полуинтервала пропорциональна частоте pi.

Кодирующая дробь строится следующим образом: строится система вложенных интервалов так, чтобы каждый последующий полуинтервал занимал в предыдущем место, соответствующее положению элемента в исходном разбиении. После того, как все интервалы вложены друг в друга можно взять любое число из получившегося полуинтервала. Запись этого числа в двоичном коде и будет представлять собой сжатый код.

Декодирование – расшифровка дроби по известному распределению вероятностей. Очевидно, что для декодирования необходимо хранить таблицу частот.

Арифметическое кодирование чрезвычайно эффективно. Коды, получаемые с его помощью, приближаются к теоретическому пределу. Это позволяет утверждать, что по мере истечения сроков патентов, арифметическое кодирование будет становиться всё более и более популярным.

Не смотря на множество весьма эффективных алгоритмов сжатия без потерь, становится очевидно, что эти алгоритмы не обеспечивают (и не могут обеспечить) достаточной степени сжатия.

Сжатие с потерями (применительно к изображениям) основывается на особенностях человеческого зрения. Рассмотрим основные идеи, лежащие в основе алгоритма сжатия изображений JPEG. JPEG на данный момент один из самых распространенных способов сжатия изображений с потерями. Опишем

основные шаги, лежащие в основе этого алгоритма. Будем считать, что на вход алгоритма сжатия поступает изображение с глубиной цвета 24 бита на пиксел (изображение представлено в цветовой модели RGB).

В цветовой модели YCbCr мы представляем изображение в виде яркостной компоненты (Y) и двух цветоразностных компонент (Cb,Cr). Человеческий глаз более восприимчив к яркости, а не к цвету, поэтому алгоритм JPEG вносит по возможности минимальные изменения в яркостную компоненту (Y), а в цветоразностные компоненты могут вноситься значительные изменения.

После перевода в цветовое пространство YCbCr выполняется дискретизация.

После дискретизации происходит дискретное косинусное преобразование. Изображение разбивается на компоненты 8*8 пикселов, к каждой компоненте применятся ДКП. Это приводит к уплотнению энергии в коде. Преобразования применяются к компонентам независимо.

Человек практически не способен замечать изменения в высокочастотных составляющих, поэтому коэффициенты, отвечающие за высокие частоты можно хранить с меньшей точностью. Для этого используется покомпонентное умножение (и округление) матриц, полученных в результате ДКП, на матрицу квантования. На данном этапе тоже можно регулировать степень сжатия (чем ближе к нулю компоненты матрицы квантования, тем меньше будет диапазон итоговой матрицы).

2. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Изучив теоретические аспекты разрабатываемой системы и выработав список требований необходимых для разработки системы, разбиваем систему на функциональные блоки(модули). Это необходимо для обеспечения гибкой архитектуры. Такой подход позволяет изменять или заменять модули без изменения всей системы в целом.

В разрабатываемом программном комплексе можно выделить следующие функциональные блоки:

- блок визуализации
- блок работы с файловой системой
- блок управления
- блок увеличения битовой глубины
- блок спектрального преобразования
- блок уменьшения битовой глубины
- блок двумерного кодирования промежуточных изображений
- блок кодирования конечного изображения

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.081 С1.

Каждый модуль выполняет свою задачу. Чтобы система работала каждый модуль взаимодействует с другими модулями путем обмена данными, используя различные форматы и протоколы.

В данном программном комплексе будет реализовано сжатие гиперспектральных данных по стандарту CCSDS 112.1-B-1. Соответственно, для большей части проекта использование сторонних библиотек недопустимо, вследствие чего весь основной функционал будет реализован самостоятельно. Из-за высоких требований к скорости обработки проекта и эффективности сжатия было решено использовать язык программирования C++ как основной, т.к. он обеспечивает наименьшие потери производительности при достаточном функционале.

Рассмотрим функциональные блоки программного комплекса.

Блок визуализации содержит функции и данные для визуализации и построения диаграмм для операций над исходным гиперспектральным изображением. Для реализации этой компоненты будет использоваться кроссплатформенный набор инструментов Qt-toolkit. Эта библиотека задумывалась и начиналась как набор инструментов для быстрой разработки графических интерфейсов приложений на языке C++, с целью упростить жизнь программистов, пишущих на C++ кроссплатформенные, переносимые GUI приложения, которые должны работать и в среде Windows и в среде Unix, Linux.

Важным преимуществом Qt является хорошо продуманный, логичный и стройный набор классов, предоставляющий очень высокий уровень абстракции. Благодаря использованию Qt, приходится писать значительно меньше кода, чем это имеет место при использовании, например, библиотеки классов

MFC. Сам же код выглядит стройнее и проще, логичнее и понятнее, чем аналогичный по функциональности код MFC или код, написанный с использованием «родного» для X11 тулкита Xt. Его легче поддерживать и развивать.

Кроме того, даже если в данный конкретный момент для реализации задуманного приложения не нужна кроссплатформенность, никто не может знать, что понадобится завтра. В случае использования Qt для того чтобы адаптировать исходное приложение для работы в другой операционной системе понадобится всего лишь перекомпиляция исходного кода. В случае же использования, например, MFC или «родных» системных API понадобится много тяжёлой работы по портированию, адаптации и отладке, а то и переписыванию с нуля существующего исходного кода для другой ОС или аппаратной платформы.

Блок работы с файловой системой является блоком необходимым для взаимодействия программного комплекса и файловой системы. Для того чтобы подать на вход программы необходимые данные и для сохранения конечного результата необходимо реализовать блок работы с файловой системой. Так как язык программирования, выбранный для реализации проекта содержит более чем исчерпывающий функционал для работы с файловой системой, было решено не использовать сторонних фреймворков или библиотек.

Блок управления предназначен для взаимодействия пользователя, пользовательского интерфейса и остальных блоков программного комплекса и будет реализован с использованием C++ и библиотеки Qt. Это наиболее подходящий для данной задачи язык программирования так как он блестяще справляется с реализацией подобного функционала, при этом расширение возможностей данного блока в будущем не станет проблемой.

Блок увеличения битовой глубины реализует этап повышения битовой частоты, применяемый к изображению перед этапом спектрального преобразования. На этом этапе частота изображения повышается путём добавления дополнительных битов в младшие разряды чисел. Целью этого этапа является повышение битовой глубины входного изображения до максимальной, поддерживаемой данной реализацией этапа преобразования. Эта увеличенная битовая глубина обычно обеспечивает более высокую производительность кодирования при высокоточном сжатии. Данный блок будет реализован с использованием C++ без применения сторонних библиотек из-за высоких требований к производительности для любых операций над самим изображением.

Блок спектрального преобразования реализует основной этап работы над исходным гиперспектральным изображением. За основу данного этапа взято целочисленное вейвлет-преобразование. Преимущества целочисленного вейвлет-преобразования:

– низкая вычислительная сложность;

– обеспечивает заметное улучшение производительности по сравнению с другими видами преобразования;

Используемое преобразование так же известно под именем CDF 5/3 и ис-

пользуется в стандарте JPEG2000, но в отличии от JPEG2000, в котором количество уровней декомпозиции является изменяемым значением, в используемом стандарте их количество фиксировано и равно пяти.

IWT – это обратимое преобразование, которое точно восстанавливает исходное входное изображение при инвертировании (при условии, что преобразованное изображение не страдает от потерь информации).

Блок уменьшения битовой глубины реализует этап понижения битовой частоты, применяемый к изображению после этапа спектрального преобразования. Целью данного этапа является уменьшение глубины бит преобразованного изображения в соответствии с глубиной бит, поддерживаемой двухмерными кодировщиками промежуточных изображений. Сжатие без потерь обычно невозможно, если на данном этапе удаляется любой младший бит.

Блок двумерного кодирования промежуточных изображений - это модуль, получающий на вход изображение пониженной битовой глубины, и создаёт сжатые изображения, используя несколько экземпляров двухмерного кодера. Каждый двухмерный кодер применяет двухмерное дискретное вейвлет-преобразование к каналу изображения с пониженной частотой. Кодированные сегменты расположены в группах, и для каждой группы существует заголовок переменной длины, который используется для описания спектрального преобразования. Группа и связанный с ней заголовок называется коллекцией. Множество коллекций будет передано в блок кодирования конечного изображения

Блок кодирования конечного изображения получает на вход множество коллекций, после чего объединяет их в последовательность которая и будет выходным сжатым изображением.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Ниже будет описана работа разрабатываемого проекта и представлена информация о структуре программного продукта.

В проекте использован объектно-ориентированный подход, поэтому код проекта разделён на классы. Каждый класс выполняет свою уникальную функцию или расширяет функционал, предоставляемый фреймворком или сторонними библиотеками. Для взаимодействия всех классов и интеграции проекта с дополнительными библиотеками используются файлы конфигурации.

3.1 Структура проекта.

Разрабатываемый проект разделен на несколько каталогов:

– data – хранит временные файлы программы и входные данные;
– transform – хранит код алгоритма обработчика;
– imageinfo – хранит код отвечающий за работу файловой системы и описание изображений;
– coder – хранит код отвечающий за финальную обработку преобразованных изображений.

3.2 Модуль описания двумерных изображений.

Модуль отвечает за хранение одномерных изображений в формате массива целых чисел, и информации об этом изображении. Модуль состоит из класса `Image_t`. Класс состоит из полей, описывающих основные составляющие двумерного изображения, такие как высоту в пикселах, ширину в пикселах, количество бит на канал изображения и значение каждого пиксела.

Класс Image имеет следующие поля:

– `Img;`
– `Width;`
– `Height;`
– `BitsPerChannel.`

Поле `Img` представляет собой одномерный массив чисел типа `int`, каждая ячейка которого представляет значение одного пиксела преобразуемого изображения.

Поле `Width` класса `Image_t` – ширина изображения в пикселах, поле `Heigh`, соответственно – высота изображения. Поле `BitsPerChannel` – количество битов на канал изображения.

Класс `Image_t` помимо конструктора обладает следующими методами:

– `Init;`
– `Destroy;`
– `Size;`
– `SizeinBytes.`

Метод Destroy уничтожает объект класса Image_t посредством удаления из памяти массива типа `int` содержащего значения каждого пиксела двухмерного изображения. Метод `Init` напротив, присваивает параметрам `Width`, `Height`, `BitsPerChannel` значения `with`, `height` и `bitsPerChannel` типа `uint32_t`, а также задаёт размер массива `Img` равным возвращаемому значению функции `Size`.

Функция `Size` возвращает результат умножения значения переменной `Width` класса `Image_t` на значение переменной `Height` класса `Image_t` типа `uint32_t`. Функция `SizeInBytes` в свою очередь возвращает произведение значений возвращаемого функцией `Size` и размер в байтах типа данных `int`.

Ниже приведён конструктор класса `Image_t`:

```
Image_t(void) {
     Width = Height = BitsPerChannel = 0;
     Img = NULL;
}
```

Функция инициализации `Init` класса `Image_t` выглядит следующим образом:

```
void Init(const uint32_t width, const uint32_t height, const
uint32_t bitsPerChannel) {
     Width = width;
     Height = height;
     BitsPerChannel = bitsPerChannel;
     Img = new int[Size()];
}
```

Деструктор класса Image_t выглядит следующим образом:

```
     void Destroy(void) {
          if (Img != NULL) {
               delete Img;
               Img = NULL;
          }
     }
```

3.3 Модуль описания трёхмерных изображений.

Гиперспектральное изображение состоит из множества слоёв. Фотография, снятая мультиспектрометром Aviris представляет собой архив двумерных изображений в формате pgm.

Для того чтобы работать с изображением, необходимо при помощи специальной команды распаковать архив, после чего мы получим количество двумерных изображений, равное количеству слоёв гиперспектрального изображения.

Модуль содержит массив объектов Image_t поля описывающие трёхмерные изображения, а также методы для работы с определёнными пикселами и слоями трёхмерного изображения. Так же класс содержит методы для подготовки изображения для чтения и сохранения в файловую систему.

Список полей класса Image3D содержит следующие поля:

- Slices;
- Slices_count;
- maxSlicesCount;
- width;
- height;
- bitsPerChannel;

Методы, которые содержит класс Image3D приведены ниже:

- Init;
- Destroy;
- LoadImage;
- SaveImage;
- Load3D;
- Store3D;
- GetXY;
- GetXZ;
- GetYZ;
- GetZX;
- GetZY;
- SetSlice;
- Width;
- Heigh;
- BitsPerChannel;
- SlicesCount;
- MaxSlicesCount;
- GetValue;
- SetValue.

Метод Init нужен для инициализации объекта класса Image3D. Метод Destroy выполняет функцию уничтожения объекта класса Image3D.

Метод GetValue и SetValue выполняют функции работы с конкретными пикселами изображения: получить i-ый пиксел из массива Img типа int объекта slices_ класса Image_t, изменить i-ый пиксел из массива Img типа int объекта slices_ класса Image_t.

Метод LoadImage принимает на вход имя файла. После, при помощи функции assert он проверяет созданы ли объекты slices_ класса Image_t

и проверяет не равно ли нулю количество слоёв maxSlicesCount_. Функция assert оценивает выражение, которое передается ей в качестве аргумента, через параметр expression. Если аргумент-выражение этого макроса в функциональной форме равно нулю (т.е. выражение ложно), сообщение записывается на стандартное устройство вывода ошибок и вызывается функция abort, работа программы прекращается.

После проводится проверка возможности считывания файла посредством вызова функции LoadPGM (описание которой будет приведено позже). Если чтение файла происходит впервые, устанавливаются значения по умолчанию. Если чтение файла происходит не впервые то выполняется сверка параметров width_, height_ и bitsPerChannel_ класса Image3D со значениями параметров объекта класса Image_t Width, Height, BitsPerChannel.

Метод SaveImage предназначен для генерации параметров и подготовки данных для сохранения обработанного изображения. На вход метод принимает название файла и начальный индекс. После, при помощи функции assert он проверяет созданы ли объекты slices_ класса Image_t и проверяет не равно ли нулю количество слоёв maxSlicesCount_.

После нахождения значения BitsPerChannel у обработанного изображения значения передаются в функцию SavePGM (о которой будет рассказано позже).

Методы Store3D и Load3D предназначены для чтения и сохранения массива grayscale изображений в формате pgm.

Функция Load3D на вход получает название файла и два индекса: начальный индекс для чтения изображений и конечный индекс чтения изображений. После генерации названия файла для чтения оно передаётся в функцию LoadImage, описание которой было приведено выше.

Методы GetZY, GetYZ, GetZX, GetXZ и GetXY предназначены для работы с массивом Img типа int в случае если необходимо получить определённую переменную по двум осям, или если необходимо работать с гиперспектральным изображением в другом представлением.

Метод Width возвращает приватное значение width_ типа uint32_t объекта класса Image3D. Метод Width возвращает значение width_ объекта класса Image3D.

Конструктор класса Image3D приведён ниже:

```
Image3D::Image3D(void) {
     slicesCount_ = maxSlicesCount_ = 0;
     width_ = height_ = 0;
     bitsPerChannel_ = 0;
     slices_ = NULL;
}
```

Методы инициализации класса Image3D выглядит следующим образом:

```
void Image3D::Init(const uint32_t maxSlicesCount, const
uint32_t width, const uint32_t height, const uint32_t
bitsPerChannel) {
     if (slices_ != NULL) {
          Destroy();
     }

     width_ = width;
     height_ = height;
     bitsPerChannel_ = bitsPerChannel;
     maxSlicesCount_ = maxSlicesCount;
     slices_ = new Image_t[maxSlicesCount];
     for (uint32_t i = 0; i < maxSlicesCount; i++) {
          slices_[i].Init(width_, height_, bitsPerChannel_);
     }
}

void Image3D::Init(const uint32_t maxSlicesCount) {
if   (slices_ != NULL) {
          Destroy();
     }
     width_ = height_ = 0;
     maxSlicesCount_ = maxSlicesCount;
     slicesCount_ = 0;
     slices_ = new Image_t[maxSlicesCount];
}
```

Деструктор класса выглядит следующим образом:

```
void Image3D::Destroy(void) {
     if (slices_ != NULL) {
          for (uint32_t i = 0; i < slicesCount_; i++) {
               slices_[i].Destroy();
          }
          delete[] slices_;
          slices_ = NULL;
          maxSlicesCount_ = slicesCount_ = 0;
     }
}
```

3.3 Модуль считывания двумерных изображений

Модуль включает в себя функционал класса `Image_t` и реализует методы `Swap`, `LoadPGM` и `SavePGM` предназначенные непосредственно для считывания и сохранения файлов типа pgm из файловой системы и в файловую систему.

Функция Swap принимает на вход значения типа `uint16_t`, после чего

выполняет побитовый сдвиг вправо на 8 бит. После чего функция возвращает результат операции побитовое или между переменной переданной в функцию и сдвинутой на восемь бит влево и переменной переданной в функцию на восемь бит вправо.

Функция `LoadPGM` принимает на вход название файла для считывания и ссылку на объект класса `Image_t`. После открывает файл из каталога data в режиме чтения `rb` (read binary). После проводится проверка на то что изображение является монохромным. После, из изображения получают информацию по поводу количества пикселов по вертикали и горизонтали, количества битов на канал. Далее при помощи этих значений происходит инициализация объекта класса `Image_t` переданного в функцию. После чего считанные из файла значения записываются в массив `Img` объекта класса `Image_t`.

Функция `SavePGM` принимает на вход название файла filename и объект класса Image_t. После чего создаёт файл с переданным названием с данными находящимися в переданном функции объекте. В конце выполнения функция отправляет ответ вызывающей функции о успешности выполнения операции.

3.4 Модуль спектрального преобразования изображений

Класс SpectralTransform включает в себя методы необходимые для спектрального преобразования изображения. Включает себя несколько полей:

- `Image_t *img;`
- `size;`
- `sizeinbytes;`
- `HighCoefficient*;`
- `LowCoefficient*;`
- `Init;`

В классе реализовано множество видов спектральных преобразований, таких как дискретное вейвлет преобразование, целочисленное вейвлет преобразование, методы для работы с матрицами и прочее. Полный список методов приведён ниже:

- `IntegerWaveletTransform53;`
- `DiscreteWaveletTransform;`
- `InvIntegerWaveletTransform;`
- `InvDiscreteWaveletTransform;`
- `TwoDimDWT;`
- `TwoDimIDWT;`
- `LoadBMP;`
- `StoreBMP;`
- `LoadWLT;`
- `Swt;`

- Iswt;
- Branch_lp_dn;
- Branch_hp_dn;
- Branch_lp_hp_up;
- Dyadic_zpad_2d;
- Dwt_output_dim;
- Zero_remove;
- dispDWT;
- sign;
- StoreWLT.

Так же в классе присутствует структура wavelet_header, которая состоит из следующих полей:

- unsigned short hsize;
- int img_size;
- char steps;
- int input_bytes;
- char h_padding;
- double scale;
- string frequency.

Так же в классе присутствует структура wavelet_header, которая состоит из следующих полей:

- int value;
- unsigned int freq;
- char bit;
- char is_root.

Конструктор класса SpectralTransform выглядит следующим образом:

```
SpectralTransform::SpectralTransform(void) {
     img = NULL;
     size = 0;
     HighCoefficients = NULL;
     LowCoefficients = NULL;
}
```

Функция инициализации класса SpectralTransform получает на вход объект класса Image_t. После чего получает информацию при помощи вызовов функции класса Image_t и записывает полученную информацию в собственные поля. Данный метод представлен ниже:

```
void SpectralTransform::Init(Image_t* src) {
     if (img != NULL) {
          Destroy();
```

```
        }
        sizeinbytes = src->SizeInBytes();
        size = src->Size();
        for(int i = 0; i<src->Size(); i++){
             img[i]=src->Img[i];
        }
    }
```

Функция `SpectralTransform::LoadBMP` принимает на вход следующие значения:

- `char *fname;`
- `int &header_addr;`
- `int &header_size;`
- `int &img_addr;`
- `int &img_size.`

При успешном выполнении функция возвращает единицу.

Функция `SpectralTransform::SWT` принимает на вход следующие значения:

- `string fname;`
- `int i;`
- `vector<double> &input;`
- `vector<double> &output;`
- `int &j.`

При успешном выполнении функция перезаписывает двухмерного вектора `output`.

Функция `SpectralTransform::ISWT` принимает на вход следующие значения:

- `string fname;`
- `int i;`
- `vector<double> &input;`
- `vector<double> &output.`

При успешном выполнении функция перезаписывает двухмерного вектора `output`.

Функция `SpectralTransform::Branch_lp_dn` принимает на вход следующие значения:

- `string fname;`
- `vector<double> &input;`
- `vector<double> &output.`

При успешном выполнении функция перезаписывает двухмерного вектора `output`.

Функция `SpectralTransform::Branch_lp_dn` принимает на вход следующие значения:

- string fname;
- vector<double> &input;
- vector<double> &output.

При успешном выполнении функция перезаписывает значение двухмерного вектора output.

Функция SpectralTransform::Branch_lp_dn принимает на вход следующие значения vector<vector<double>> &input и vector<vector<double>> &output. При успешном выполнении функция перезаписывает значения двухмерного вектора векторов output.

Функция SpectralTransform::Branch_lp_hp_up принимает на вход следующие значения:

- string fname;
- vector<double> &inputh;
- vector<double> &inputl;
- vector<double> &output.

SpectralTransform::Branch_lp_hp_up при успешном выполнении функция перезаписывает значения двухмерного вектора output.

Функция SpectralTransform::Dyadic_zpad_2d принимает на вход следующие значения:

- string fname;
- vector<vector<double>> &inputh;
- vector<vector<double>> &inputl;
- vector<vector<double>> &output.

SpectralTransform::Dyadic_zpad_2d при успешном выполнении перезаписывает значения двухмерного вектора векторов output.

Функция SpectralTransform::Zero_remove принимает на вход следующие значения:

- string fname;
- vector<vector<double>> &inputh;
- vector<vector<double>> &inputl;
- vector<vector<double>> &output.

SpectralTransform::Zero_remove при успешном выполнении перезаписывает значения двухмерного вектора векторов output.

Функция SpectralTransform::DispDWT принимает на вход следующие значения:

- string fname;
- vector<vector<double>> &inputh;
- vector<vector<double>> &inputl;
- vector<vector<double>> &output.

SpectralTransform::DispDWT при успешном выполнении перезаписывает значения которые содержит в данный момент двухмерный вектор векторов output.

Функция SpectralTransform::StoreBMP принимает на вход следующие значения:

- char *fname;
- int &header_addr;
- int &header_size;
- int &img_addr;
- int &img_size.

SpectralTransform::StoreBMP при успешном выполнении возвращает единицу.

Функция SpectralTransform::StoreWLT принимает на вход следующие значения:

- char *fname;
- wavelet_header wlt;
- unsigned char *bmp_header_data;
- int &bmp_header_size;
- int img_size;
- unsigned char *img_data;
- unsigned int *img_size.

SpectralTransform::StoreWLT при успешном выполнении возвращает единицу.

Функция SpectralTransform::LoadWLT принимает на вход следующие значения:

- char *fname;
- wavelet_header wlt;
- unsigned char *bmp_header_data;
- int &bmp_header_size;
- int img_size;
- unsigned char *img_data.

SpectralTransform::LoadWLT при успешном выполнении возвращает единицу.

Функция IntegerWaveletTransform получает на вход массив типа double размера size. После чего происходит массива по алгоритму. Так же этот алгоритм в англоязычной литературе CDF5/3, и используется этот алгоритм в таком стандарте как JPEG2000.

JPEG2000 это графический формат, который вместо дискретного косинусного преобразования, применяемого в стандарте JPEG, использует технологию вейвлет-преобразования, основывающуюся на представлении сигнала в

виде суперпозиции базовых функций – волновых пакетов.

В результате такой компрессии изображение получается более гладким и чётким, а размер файла по сравнению с JPEG при одинаковом качестве является меньшим (см. рисунок 3.1). JPEG 2000 полностью свободен от главного недостатка своего предшественника: благодаря использованию вейвлетов, изображения, сохранённые в этом формате, при высоких степенях сжатия не содержат артефактов в виде «решётки» из блоков размером 8x8 пикселей.

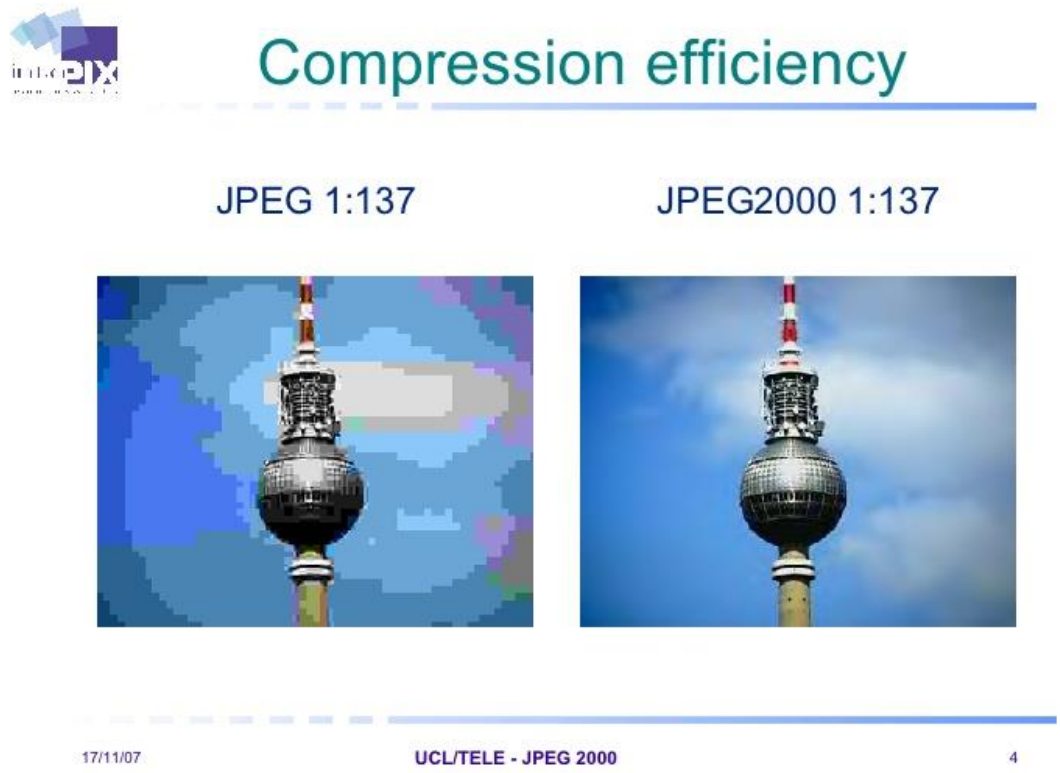

Рис 3.1 Эффективность сжатия JPEG2000[x]

Формат JPEG 2000 так же, как и JPEG, поддерживает так называемое «прогрессивное сжатие», позволяющее по мере загрузки видеть сначала размытое, но затем всё более чёткое изображение.

По этой причине в стандарте, реализуемом в данном проекте был использован данный метод сжатия. Сжатие по стандарту, которое было реализовано в данном программном комплексе, было разработано Международным Консультативным Комитетом по космическим системам передачи данных(CCSDS), образованным в 1982 году.

Но количество трансформаций в стандарте JPEG2000 является изменяемым, в стандарте же который реализован в этом проекте оно зафиксировано на отметке в 5.

Для получения требуемого результата необходимо чтобы функция цело-

численного вейвлет-преобразования возвращала конкатенацию из высокочастотных и низкочастотных коэффициентов.

Для этого необходимо пятикратно пройтись функцией по набору входных данных поданных на неё. Таким образом мы получим пять наборов высокочастотных коэффициентов и один набор низкочастотных (см. рисунок 3.2).

После конечного преобразования значения будут обратно записаны в массив.

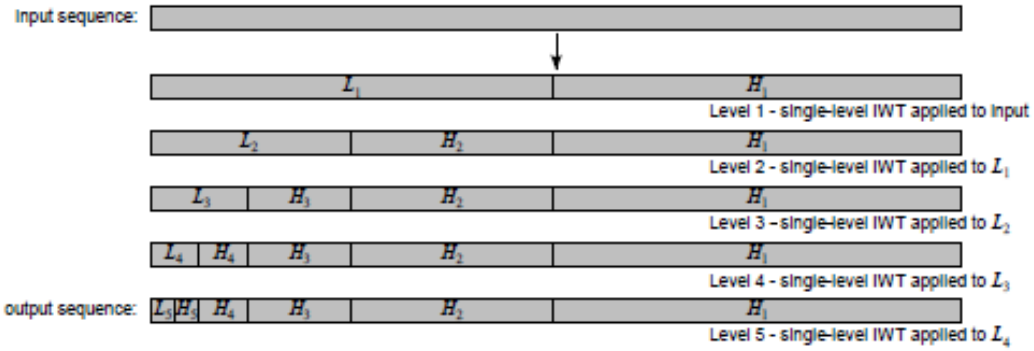

Рис 3.2 Пример целочисленного пятикратного вейвлет-преобразования[x]

Функция Дискретного вейвлет-преобразования `DiscreteWavelet-Transform` принимает на вход следующие аргументы:

- `vector<double> sig;`
- `int J;`
- `string nm;`
- `vector<double> dwt_output;`
- `vector<double> flag;`
- `vector <int> length.`

Где `sig` – вектор входных сигналов, `J` – количество ровней декомпозиции, `length` – длина входного вектора, `dwt_output` – возвращаемый функцией преобразованный вектор.

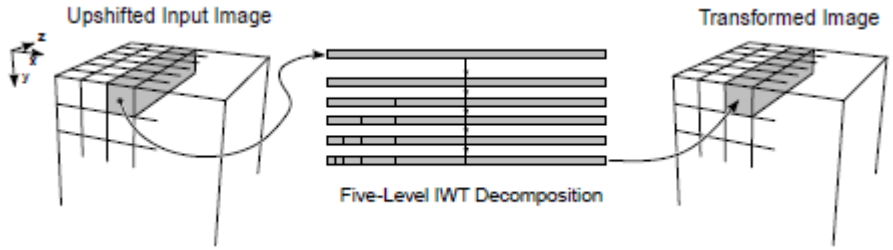

Рис 3.3 Целочисленное вейвлет-преобразование[x]

Дискретное вейвлет-преобразование – преобразование в которых вейвлеты представлены выборками.

Дискретное вейвлет-преобразование (DWT) - реализация вейвлет-преобразования с использованием дискретного набора масштабов и переносов вейвлета, подчиняющихся некоторым определённым правилам. Другими словами, это преобразование раскладывает сигнал на взаимно ортогональный набор вейвлетов, что является основным отличием от непрерывного вейвлет-преобразования (CWT), или его реализации для дискретных временных рядов, иногда называемой непрерывным вейвлет-преобразованием дискретного времени (DT-CWT).

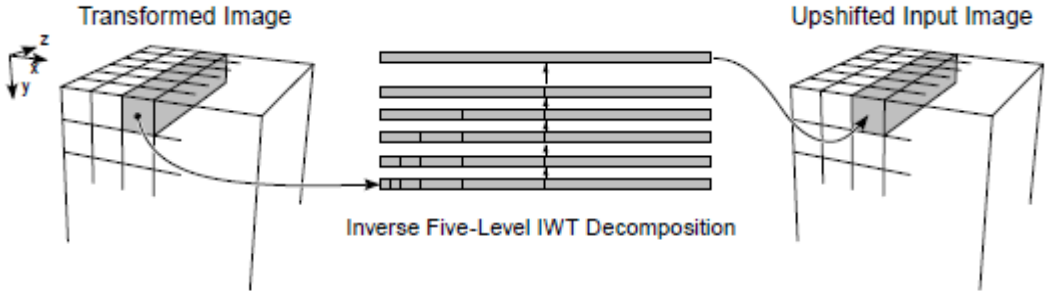

Рис. 3.4 Алгоритм работы обратного целочисленного вейвлет-преобразования[x]

Функция InvIntegerWaveletTransform выполняет восстановительную функцию изображения над которым была проведена операция спектрального преобразования (см. рисунок 3.4).

Функция двумерного дискретного вейвлет-преобразования TwoDimDWT принимает на вход те же параметры что и одномерное дискретное вейвлет-преобразование, а именно:

- `vector<double> sig;`
- `int J;`
- `string nm;`
- `vector<double> dwt_output;`
- `vector<double> flag;`
- `vector <int> length.`

Существует два подхода к реализации дискретного вейвлет-преобразования: симметричное и периодическое расширение

Различие между симметричным и периодическим дискретным вейвлет-преобразованием состоит в том что при симметричном вейвлет преобразовании выходной вектор обладает такой же длинной как и входной вектор, а при периодическом вейвлет-преобразовании имеется избыточность, которая зависит от типа выбранного фильтра.

Как и в случае с целочисленным вейвлет-преобразованием существует

обратное дискретное вейвлет преобразование, которое работает подобным образом.

Для того чтобы получить исходное, неизменённое изображение необходимо передать в функцию вектор обработанного изображения, и при использовании того же фильтра получить исходное изображения, путём обратного применения алгоритма к входному вектору.

3.5 Модули битового сдвига изображения.

Для того чтобы провести обработку изображения при помощи спектрального преобразования, необходимо для провести операцию битового сдвига изображения. Суть этой операции заключается в том чтобы умножить число, которое является пикселом считанным из исходного изображения, на два в степени `U`. Число `U` выбирается произвольным образом в промежутке от нуля до пятнадцати. Так же, после спектральной трансформации изображения необходимо каждый пиксел умножить на два в степени минус `D`.

Эта операция являет собой по сути операцию битовой частоты изображения. Коэффициенты нужно выбирать из расчёта что разность коэффициента U и коэффициента D не должна превышать пяти.

Функция `Upshift` получает на вход объект класса `Image_t` и коэффициент U. После чего, во время прохода по массиву который является полем класса Image_t каждое из значений массива умножается на 2 в степени U. После чего значения в массиве перезаписываются на новые. Функция возвращает преобразованный объект класса Image_t.

Аналогичным образом работает функция `Downshift`, которая так же получает на вход объект класса `Image_t`, только в отличии от функции `Upshift`, вторым аргументом на вход поступает понижающий коэффициент `D`. После чего, во время прохода по массиву который является полем класса `Image_t` каждое из значений массива умножается на 2 в степени минус `D`. После чего значения в массиве перезаписываются на новые. Функция возвращает преобразованный объект класса `Image_t`.

Класс `Upshift` имеет следующие поля:

- `Image_t src*;`
- `int U;`
- `int i.`

Конструктор класса `Upshift` выглядит следующим образом:

```
Upshift::Upshift(void) {
     i = 0;
     U = 0;
     src = NULL;
}
```

Класс `Upshift` имеет следующие поля:

- Image_t src*;
- int D;
- int i.

Конструктор класса Downshift выглядит следующим образом:

```
Downshift::Downshift(void) {
     i = 0;
     D = 0;
     src = NULL;
}
```

3.6 Описание класса WaveletTransform.

Класс WaveletTransform содержит в себе следующие поля:

- vector<vector<double>> sig;
- int J;
- string nm;
- vector<double> output.

Класс WaveletTransform состоит из следующих функций:

- filtcoef;
- getcoeff2d;
- vecsum;
- downsamp;
- bitreverse;
- convfftm;
- convfft;
- freq;
- multisamp;
- reversecoeff;
- bitshift;
- vecmult;
- coeffreverse;
- circshift.

Функция WaveletTransform::filtcoef имеет следующие входные переменные:

- string name;
- vector<double> &lp1;
- vector<double> &hp1;
- vector<double> &lp2;
- vector<double> &hp2.

Данная функция класса необходима для того чтобы высчитать высокие

и низкие частоты для фильтра, используемого определённым вейвлетом.

Функция `WaveletTransform::reversecoeff` имеет следующие входные переменные:

- `string name;`
- `vector<double> &lp1;`
- `vector<double> &hp1;`
- `vector<double> &lp2;`
- `vector<double> &hp2.`

Данная функция класса необходима для того чтобы высчитать высокие и низкие частоты для фильтра, используемого определённым вейвлетом.

Функция `reversecoeff` класса `WaveletTransform` записывает значения фильтров в передаваемые ей вектора `lp1`, `lp2`, `hp1` и `hp2` в зависимости от названия фильтра, переданного в переменной `name`.

Функция `getcoeff2d` класса `WaveletTransform` имеет следующие входные переменные:

- `vector<vector<double>> &dwtoutput;`
- `vector<vector<double>> &cH;`
- `vector<vector<double>> &cV;`
- `vector<vector<double>> &cD;`
- `vector<vector<double>> &flag;`
- `int &N.`

Функция `getcoeff2d` класса `WaveletTransform` имеет тип возвращаемого значения `void`, т.е. не возвращает никаких значений. Функция записывает результат вычислений в следующие вектора:

- `vector<vector<double>> &cH;`
- `vector<vector<double>> &cV;`
- `vector<vector<double>> &cD.`

Функция `WaveletTransform::getcoeff` имеет следующие входные переменные:

- `vector<double> &dwtoutput;`
- `vector<double> &cH;`
- `vector<double> &cV;`
- `vector<double> &cD;`
- `vector<vector<double>> &flag;`
- `int &N.`

Функция `getcoeff` класса `WaveletTransform` имеет тип возвращаемого значения `void`, т.е. не возвращает никаких значений. Функция записывает результат вычислений в следующие вектора:

- `vector<double> &cH;`
- `vector<double> &cV;`

- vector<double> &cD;

Функция WaveletTransform::vecsum имеет следующие входные переменные:

- vector<double> &a;
- vector<double> &b;
- vector<double> &c.

Функция WaveletTransform::vecsum не возвращает никаких переменных, а перезаписывает значения переданных ей векторов, а именно вектора c типа double.

Функция WaveletTransform::vecmult имеет следующие входные переменные:

- vector<double> &a;
- vector<double> &b;
- vector<double> &c.

Функция WaveletTransform::vecmult не возвращает никаких переменных, а перезаписывает значения переданных ей векторов, а именно вектора c типа double.

Функция WaveletTransform::coeffreverse имеет следующие входные переменные:

- vector<double> &a;
- vector<double> &b;
- vector<double> &c.

Функция coeffreverse класса WaveletTransform не возвращает никаких переменных, а перезаписывает значения переданных ей векторов, а именно вектора c типа double.

Функция WaveletTransform::downsamp получает на вход следующие переменные:

- vector<double> &sig;
- int M;
- vector<double> &sig_d;

Функция WaveletTransform::downsamp перезаписывает значение переданной переменной sig_d.

Функция WaveletTransform::multisamp получает на вход следующие переменные:

- vector<vector<double>> &sig;
- int M;
- vector<double> &sig_d;

Функция WaveletTransform:: multisamp перезаписывает значение переданной переменной sig_d.

Функция WaveletTransform::bitreverse имеет одну входную

переменную – vector<complex<double>> &sig. Функция перезаписывает значение этой переменной.

Функция WaveletTransform::bitshift имеет одну входную переменную – vector<complex<double>> &sig. Функция перезаписывает значение этой переменной.

Функция convfftm имеет следующие входные переменные:

- vector<double> &a;
- vector<double> &b;
- vector<double> &c.

Функция не возвращает никаких значений, а перезаписывает значение в переданном ей векторе c типа double.

Функция convfft имеет следующие входные переменные:

- vector<double> &a;
- vector<double> &b;
- vector<double> &c.

Функция не возвращает никаких значений, а перезаписывает значение в переданном ей векторе c типа double.

Функция freq получает на вход два вектора типа double: sig и freq_resp. Функция перезаписывает значение в переданном ей векторе sig.

Функция circshift получает на вход две переменные: вектор типа double sig_cir и переменную типа int L. Функция перезаписывает значение в переданном ей векторе sig_cir.

3.7 Описание класса EncodeMatrix.

Каждый двумерный кодер применяет дискретное вейвлет-преобразование к каналу преобразованного изображения.

Класс EncodeMatrix имеет следующие поля:

- int size;
- float *coeff
- static const char magic[2];
- unsigned int bpp;
- unsigned int width;
- unsigned int height;
- unsigned int realWidth;
- unsigned int realHeight;
- unsigned char nrm(int val);
- unsigned int range(float val).

Конструктора класса EncodeMatrix имеет следующий вид:

```
Downshift::Downshift(Bitmap *input, unsigned int
bpp):bpp(bpp) {
     width = input->getPadWidth();
     height = input->getPadHeight()
     realWidth = input->width();
     realHeight = input->height();
     coeff = input->getPadded(width, height);
}
```

Класс EncodeMatrix имеет следующий список методов:

- CalculateCoeff;
- transform1d;
- untransform1d;
- SizeMatrix;
- ResizeMatrix;
- ClearCoeff;
- ClearMatrix;
- ClearColumn;
- ClearLine;
- ClearColumns;
- ClearLines;
- MatrixLineTransponse;
- MatrixColumnTransponse;
- MatrixTransponse.

Функция EncodeMatrix::CalculateCoef принимает на вход следующие переменные:

- int* param;
- int size;
- string nm.

Функция EncodeMatrix класса CalculateCoef возвращает массив типа int retparam размером size.

Функция EncodeMatrix::SizeMatrix принимает на вход массив типа int и значение size типа int. Функция не возвращает никаких значений.

Функция ResizeMatrix класса EncodeMatrix принимает на вход массив типа int, значение size типа int и значение new_size типа int. Функция ResizeMatrix не возвращает никаких значений.

Функция ClearCoeff класса EncodeMatrix принимает на вход массив типа int и значение size типа int. Функция не возвращает никаких значений.

Функция ClearMatrix класса EncodeMatrix принимает на вход

двухмерный массив типа int, значение size типа. Функция не возвращает никаких значений.

Функция ClearLine класса EncodeMatrix принимает на вход двухмерный массив типа int, значение size типа. Функция не возвращает никаких значений.

Функция ClearColumn класса EncodeMatrix принимает на вход двухмерный массив типа int, значение size типа. Функция не возвращает никаких значений.

Функция MatrixTransponse класса EncodeMatrix принимает на вход указатель на двухмерный массив типа int, и перезаписывает значения в нём.

Функция MatrixLineTransponse класса EncodeMatrix принимает на вход указатель на двухмерный массив типа int, и перезаписывает значения в нём.

Функция MatrixColumnTransponse класса EncodeMatrix принимает на вход указатель на двухмерный массив типа int, и перезаписывает значения в нём.

Функция EncodeMatrix::transform1d принимает на вход следующие переменные:

- float *src;
- unsigned int length;
- unsigned int step;
- float *tmp.

Функция transform1d класса EncodeMatrix не возвращает никаких значений, а перезаписывает значение в массиве типа float *src.

Функция EncodeMatrix::untransform1d принимает на вход следующие переменные:

- float *src;
- unsigned int length;
- unsigned int step;
- float *tmp.

Функция untransform1d класса EncodeMatrix не возвращает никаких значений, а перезаписывает значение в массиве типа float *src.

3.8 Описание класса ProduceSideInfo.

Класс предназначен для записи в файлы информации о сжатии изображения, такой как:

- эффективность сжатия;
- информация о старом и новых размерах;
- размер и название фильтров;

– используемый тип сжатия.

Класс `ProduceSideInfo` имеет следующие поля:

– `float *hcoeff;`
– `float *lcoeff;`
– `char *filename;`
– `char *dir;`
– `unsigned int width;`
– `unsigned int height;`
– `unsigned int realWidth;`
– `unsigned int realHeight;`

Класс `ProduceSideInfo` содержит следующие функции:

– `WriteWLTCfc;`
– `WriteSzs;`
– `CreateFile;`
– `CreateFiles;`
– `WriteSzsByte;`
– `CalcRealWdt;`
– `CalcBitSyze;`
– `CalcRealHgt.`

Функция `ProduceSideInfo::WriteWLTCfc` принимает на вход следующие переменные:

– `char* filename;`
– `float* hcoeff;`
– `float* lcoeff;`
– `int j.`

`ProduceSideInfo::WriteWLTCfc` при успешном выполнении возвращает единицу.

Функция `ProduceSideInfo::WriteSzs` принимает на вход следующие переменные:

– `char* filename;`
– `unsigned int width;`
– `unsigned int height;`
– `unsigned int realWidth;`
– `unsigned int realHeight;`
– `int j.`

`ProduceSideInfo::WriteSzs` при успешном выполнении возвращает единицу.

Функция `ProduceSideInfo::CreateFiles` принимает на вход следующие переменные:

- `char* filename;`
- `int startidx;`
- `int stopidx;`
- `int* sizes;`
- `char ext.`

Во время выполнения функция `CreateFiles` вызывает функцию `CreateFile` класса `ProduceSideInfo`. Функция `CreateFiles` класса `ProduceSideInfo` при успешном выполнении возвращает единицу.

Функция `ProduceSideInfo::CreateFile` принимает на вход следующие переменные:

- `char filename;`
- `int size;`
- `char ext.`

При успешном выполнении функция `CreateFile` класса `ProduceSideInfo` возвращает единицу.

Функция `ProduceSideInfo::CalcRealWdt` принимает на вход следующие переменные:

- `char filename;`
- `char ext.`

Функция `CalcRealWdt` класса `ProduceSideInfo` возвращает переменную `wdth` типа unsigned `int`.

Функция `ProduceSideInfo::CalcRealHgt` принимает на вход следующие переменные:

- `char filename;`
- `char ext.`

Функция `CalcRealHgt` класса `ProduceSideInfo` возвращает переменную `hght` типа unsigned `int`.

Конструктор класса выглядит следующим образом:

```
ProduceSideInfo::ProduceSideInfo (void){
     hcoeff = NULL;
     lcoeff = NULL;
     filename = NULL;
     width = 0;
     height = 0;
     realWidth = 0;
     realHeight = 0;
}
```

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

При разработке системы одними из наиважнейших требований к исходному коду являются его расширяемость и поддерживаемость. Реализация программных модулей с учетом этих требований приводит к простоте расширения функционала в критических местах, обеспечению разделенности и независимости компонентов системы, что улучшает их тестируемость и в целом позволяет добиться реализации более стабильной и простой в понимании кодовой базы.

4.1 Разработка модуля для работы с файловой системой.

Для прочтения из файловой системы и сохранения готовых файлов в неё реализованы следующие функции:

- `LoadPGM;`
- `StorePGM;`
- `Load3D;`
- `Store3D;`
- `LoadImage;`
- `SaveImage;`

Класс `LoadPGM` выглядит следующим образом:

```
bool LoadPGM(const char *fileName, Image_t &image) {
     FILE *fp = fopen(fileName, "rb");
     if (fp == NULL)
          return false;
```

Подробнее рассмотрим данную функцию. На вход данной функции поступает название файла для прочтения и объекта класса `Image_t`. После при помощи функции `fopen` производится открытие файла в бинарном режиме. Если открыть файл не удалось происходит выход из функции.

```
     char header[PGMHeaderSize] = { 0 };
     uint32_t startPosition = 0;
     while (header[startPosition] == 0) {
          startPosition = 0;
          if (fgets(header, PGMHeaderSize, fp) == NULL)
               return 0;
          while (isspace(header[startPosition])) startPosi-
tion++;
     }
```

Далее происходит инициализация массива `header`. После присваиваем переменной `startPosition` значение 0. После чего в цикле инкрементируем значение `startPosition` если значение элемент массива является пробелом.

```
    if (strncmp(&header[startPosition], "P5", 2) != 0)
    {
        fprintf(stderr, "Just gryayscale images sup-
ported\n");
        return false;
    }
```

Далее происходит проверка на то, является ли открытый файл монохромным изображением. Если файл не является монохромным изображением то происходит выход из цикла. После этого проводится проверка на конец файла.

```
    uint32_t maxval = 0, width = 0, height = 0;
    uint32_t i = 0;
    int readsCount = 0;
    int strOffset = 2;
    if ((int)(i = sscanf(&header[startPosition + strOffset],
"%u %u %u", &width, &height, &maxval)) == EOF)
        i = 0;
    while (i < 3)
     {
        if (fgets(header, PGMHeaderSize, fp) == NULL)
            return false;
        if (header[0] == '#')
            continue;
        switch (i)
            {
            case 0:
                if ((readsCount = sscanf(header, "%u %u
%u", &width, &height, &maxval)) != EOF)
                    i += readsCount;
                break;
            case 1:
                if ((readsCount = sscanf(header, "%u
%u", &height, &maxval)) != EOF)
                    i += readsCount;
                break;
            case 2:
                if ((readsCount = sscanf(header, "%u",
&maxval)) != EOF)
                    i += readsCount;
                break;
        }
    }
```

После этих операций происходит считывание файла и запись значений `width`, `height` и `bytePerChannel` в объект класса `Image_t`. Так же происходит запись в цикле каждого пиксела в массив `Img`. После чтения файл закрывается.

```
int bitsPerChannel = int(log(maxval + 1.0) / log(2.0));
image.Init(width, height, bitsPerChannel);
const int bytePerChannel = (image.BitsPerChannel + 7) / 8;
uint8_t *tmp = new uint8_t[width * height * bytePerChannel];
     if (fread(tmp, sizeof(uint8_t), width * height *
bytePerChannel, fp) == 0) return false;
     if (bytePerChannel == 2)
{
          uint16_t *ptr = reinterpret_cast<uint16_t *>(tmp);
          for (i = 0; i < height; i++)
          {
               for (uint32_t j = 0; j < width; j++)
               {
                    image.Img[i * width + j] = Swap(ptr[i *
width + j]);
               }
          }
     } else
          {
          for (i = 0; i < height; i++)
          {
               for (uint32_t j = 0; j < width; j++)
               {
               image.Img[i * width + j] = tmp[i * width +
j];
               }
          }
     }
     delete tmp;
     fclose(fp);
     return true;
}
```

Класс `StorePGM` выполняет функцию сохранения преобразованного файла и выглядит следующим образом:

```
bool SavePGM(const char *fileName, Image_t *image)
{
     assert(image->Img != NULL);
     assert(image->Width > 0);
     assert(image->Height > 0);
```

В первую очередь при помощи функции assert происходит проверка полей объекта класса `Image_t`. Далее происходит попытка создания файла. Если попытка создания файла не оказалась успешной происходит выход из функции.

```
     std::fstream fh(filename,std::fstream::out|
std::fstream::binary);
     if (fh.bad()) {
```

```
            return false;
        }
```

Далее в файл записывается атрибут монохромного файла, который собственно и отвечает за то что система видит файл как монохромный. Далее происходит запись в файл информации о размерах файла, т.е. переменных `width`, `height` и `bytePerChannel`.

```
        fh << "P5\n";
        fh << image->Width << ' ' << image->Height << '\n' <<
((1 << image->BitsPerChannel) - 1) << std::endl;
        const int bytePerChannel = (image->BitsPerChannel + 7)
/ 8;
```

Далее создаётся массив типа `uint8_t` с названием `buffer`, размером равным произведению `width`, `height` и `bytePerChannel`. После в этот массив заносятся значения из массива `Img` объекта класса `Image_t`.

```
        uint8_t *buffer = new uint8_t[image->Width * image-
>Height * bytePerChannel];
        if (bytePerChannel == 2)
        {
            uint16_t *tmp = reinterpret_cast<uint16_t
*>(buffer);
            for (uint32_t i = 0; i < image->Height; i++)
            {
                for (uint32_t j = 0; j < image->Width; j++)
                {
                    tmp[i*image->Width + j] =
Swap((uint16_t)image->Img[i * image->Width + j]);
                }
            }
        }
        else {
            for (uint32_t i = 0; i < image->Height; i++) {
                for (uint32_t j = 0; j < image->Width; j++)
                {
                    buffer[i * image->Width + j] =
(uint8_t)image->Img[i * image->Width + j];
                }
            }
        }
```

После приведения типа к `char` при помощи функции `reinterpret_cast` данные записываются в файл. Функция возвращает значение `true`.

```
        fh.write(reinterpret_cast<const char *>(buffer), image-
```

```
>Width * image->Height * bytePerChannel);
     fh.flush();
     if (fh.bad())
          return false;
     delete buffer;
     fh.close();
     return true;
}
```

Функция `LoadImage` класса `Image3D` выглядит следующим образом:

```
void Image3D::LoadImage(const char *fname) {
     assert(slices_ != NULL && maxSlicesCount_ != 0);
     assert(slicesCount_ < maxSlicesCount_);
```

На этом участке кода при помощи функции `assert` проверяем наличие созданных элементов класса `Image_t` и проверяем инициализировано ли поле `maxSlicesCount`.

```
     if (!LoadPGM(fname, slices_[slicesCount_]))
     {
          fprintf(stderr, "File open error\n");
          return;
     }
```

Далее проверяем считывание файла, если файл считать не удалось, выводим ошибку и выходим из данного метода.

```
     if (width_ == 0 || height_ == 0)
     {
          width_ = slices_[slicesCount_].Width;
          height_ = slices_[slicesCount_].Height;
                    bitsPerChannel_ =
          slices_[slicesCount_].BitsPerChannel;

          printf("Default image parameters (w x h @ bits):
%d x %d @ %d\n", width_, height_, bitsPerChannel_);
     }
     else
     {
          if (slices_[slicesCount_].Width != width_ ||
               slices_[slicesCount_].Height != height_ ||
               slices_[slicesCount_].BitsPerChannel !=
bitsPerChannel_)
          {
               fprintf(stderr, "File %s loading failed (im-
age parameters mismatch with default\n", fname);
               return;
          }
```

Далее инициализируем поля класса Image3D полями класса Image_t. Если поля класса уже были проинициализированы то проверяем правильность инициализации, и в случае если инициализация была произведена неправильно и данные не совпадает, выводим ошибку и выходим из функции. После выполнения всех операций инкрементируем счётчик и выходим из функции.

```
          }
          slicesCount_++;
     }
```

Функция SaveImage класса Image3D выглядит следующим образом:

```
     void Image3D::SaveImage(const char *fname, const uint32_t
sliceIdx)
     {
          assert(slices_ != NULL && maxSlicesCount_ != 0);
          assert(sliceIdx < slicesCount_);
```

На этом участке кода при помощи функции assert проверяем наличие созданных элементов класса Image_t и проверяем инициализировано ли поле maxSlicesCount. Далее используем функцию FindMaxValue для поиска значения BitsPerChannel, после чего переводим его в нужную систему и вызываем функцию SavePGM.

```
          int32_t maxValue = FindMaxValue(&slices_[sliceIdx]);
          if (maxValue < 0)
          {
               fprintf(stderr, "Max value in slice[%d] is not cor-
rect: %d\n", sliceIdx, maxValue);
               return;
          }
          if (maxValue == 0)
          {
               maxValue = 2;
          }

          double log2MaxValue = log((double)maxValue) / log(2.0);
          slices_[sliceIdx].BitsPerChannel                    =
(int)ceil(log2MaxValue);
          SavePGM(fname, &slices_[sliceIdx]);
     }
```

Функция Load3D класса Image3D выглядит следующим образом:

```
     void Image3D::Load3D(const char *fname, const uint32_t
startIdx, const uint32_t stopIdx)
     {
          char tmp[300];
```

```
        for (uint32_t i = startIdx; i < stopIdx; i++)
        {
            //sprintf(tmp, "%s_%04d.pgm", fname, i);
            sprintf(tmp, "%04d.pgm", i);
            printf("%s ", tmp);
            LoadImage(tmp);
        }
```

На данном участке кода генерируем имена файлов для загрузки. Генерируем их по закону. Причина генерации имён файлов заключается в том, что файлов будет около нескольких сотен, соответственно нет смысла вводить название каждого файла отдельно, тем более что названы они схожим образом.

```
        if(slicesCount_ > 0)
        {
            const float bytePerChannel =
(float)((float)bitsPerChannel_ / 8.0);
            const float imageSize = width_ * height_ *
bytePerChannel;
            printf("\tfile size: %d bytes (%.3f kBytes)\n",
int(imageSize), float(imageSize) / 1024.0f);
            printf("\ttotal file size (%d slices): %d bytes
(%.3f kBytes)\n", slicesCount_, int(slicesCount_ * imageSize),
                float(slicesCount_ * imageSize) / 1024.0f);
        }
    }
```

Функция Store3D класса Image3D выглядит следующим образом:

```
    void Image3D::Store3D(const char *fname) {
        char tmp[300];
        for (uint32_t i = 0; i < slicesCount_; i++) {
            sprintf(tmp, "%s.%04d.pgm", fname, i);
            SaveImage(tmp, i);
        }
    }
```

Функции `Init` классов `Image_t` и `Image3D`, предназначенные для считывания значений и записей их в поля классов выглядят следующим образом:

```
    void Image3D::Init(const uint32_t maxSlicesCount)
    {
        if (slices_ != NULL)
        {
            Destroy();
        }

        width_ = height_ = 0;
```

```
        maxSlicesCount_ = maxSlicesCount;
        slicesCount_ = 0;
        slices_ = new Image_t[maxSlicesCount];
    }

    void Image3D::Init(const uint32_t maxSlicesCount, const
uint32_t width, const uint32_t height, const uint32_t bitsPerChan-
nel)
    {
        if (slices_ != NULL)
        {
            Destroy();
        }

        width_ = width;
        height_ = height;
        bitsPerChannel_ = bitsPerChannel;
        maxSlicesCount_ = maxSlicesCount;
        //slicesCount_ = maxSlicesCount;

        slices_ = new Image_t[maxSlicesCount];
        for (uint32_t i = 0; i < maxSlicesCount; i++)
        {
            slices_[i].Init(width_, height_, bitsPerChannel_);
        }
    }
```

Функция Init класса Image_t выглядит следующим образом:

```
        void Init(const uint32_t width, const uint32_t height,
const uint32_t bitsPerChannel)
        {
            Width = width;
            Height = height;
            BitsPerChannel = bitsPerChannel;

            Img = new int[Size()];
        }
```

Функция `SetSlice` предназначенная для смены параметров слоя выглядит следующим образом:

```
    void Image3D::SetSlice(Image_t *src)
        {
        assert(slicesCount_ < maxSlicesCount_);

        if (width_ == 0 || height_ == 0)
        {
            width_ = src->Width;
            height_ = src->Height;
```

```
                bitsPerChannel_ = src->BitsPerChannel;
          }
          else
          {
                // проверяем чтение файла
                if (src->Width != width_ ||
                     src->Height != height_ ||
                     src->BitsPerChannel != bitsPerChannel_)
                {
                     fprintf(stderr, "Image slice parameters
failed\n");
                     return;
                }
          }

          slices_[slicesCount_].Init(width_,  height_,  bitsPer-
Channel_);
          memcpy(slices_[slicesCount_].Img, src->Img, src->Size-
InBytes());
          slicesCount_++;
     }
```

4.2 Разработка модулей битовых сдвигов.

Модули битовых сдвигов состоят из следующих основных функций:

- `DownshiftImage;`
- `UpshiftImage;`
- `Downshift3D;`
- `Upshift3D.`

Функция `DownshiftImage` выглядит следующим образом:

```
void Downshift::DownshiftImage(Image_t *src, int D)
{
     int i = 0;
     double j = 1;
     temp= new Int[src->Size()];
     for(i=0; i<src->Size(); i++)
     {
          temp[i]=pow(src->Img[i], 1/D);
     }
     src->Img = temp;
}
```

В данной функции выполняется стадия обратного битового сдвига преобразованного изображения, что в данном подразумевает под собой извлечения корня из значения каждого значения записанного в массив Img класса Image_t степени D.

Функция `UpshiftImage` выглядит следующим образом:

```
void Upshift::UpshiftImage(Image_t *src, int U){
     int i = 0;
     temp= new Int[src->Size()];
     for(i=0; i<src->Size(); i++)
     {
          temp[i]=pow(src->Img[i], U);
     }
     src->Img = temp;
}
```

В данной функции выполняется стадия битового сдвига, что подразумевает под собой возведение в степень `U` значения каждого пиксела массива `Img` класса `Image_t`.

Функция `Upshift::Upshift3D` выглядит следующим образом:

```
void Upshift::Upshift3D(Image3D *src, int U) {
     assert(slices_ != NULL && maxSlicesCount_ != 0);
     assert(slicesCount_ < maxSlicesCount_);
     int i = 0;
     for (int i = 0; i < src->maxSlicesCount_; i++)
     {
          UpshiftImage(src->slices_[i], U);
     }
}
```

Функция предназначена для передачи параметров и объектов в функцию `UpshiftImage`.

Функция `Downshift::Downshift3D` выглядит следующим образом:

```
void Downshift::Downshift3D (Image3D *src, int D) {
     assert(slices_ != NULL && maxSlicesCount_ != 0);
     assert(slicesCount_ < maxSlicesCount_);
     int i = 0;
     for (int i = 0; i < src->maxSlicesCount_; i++)
     {
          DownshiftImage(src->slices_[i], D);
     }
}
```

Функция предназначена для передачи параметров и объектов в функцию `DownshiftImage`.

4.3 Разработка модулей спектрального преобразования.

Функция спектрального преобразования `fw_encode_image` выглядит следующим образом:

```
void fw_encode_image(int **out,
```

```
                    char *filename,
                    unsigned char *header,
                    wavparams_t *p)
{
    image_t img;
    quantizer_t q;
    vfwl_init_image(&img);
    vfwl_import_pnm(&img, filename);
    p->w = img.w;
    p->h = img.h;
    p->nb_channels = img.nb_channels;
     if (img.nb_channels == 3)
        vfwl_convert_RGB_to_YCbCr(&img);
    vfwl_encode_ch(out, (int*)img.channels, header, p);
    q.nb_bits = 16;
    q.table_size = 65536;
    vfwl_uquantizer_init(&q);
    vfwl_uquantize_slow(*out, p, &q);
    vfwl_uquantizer_free(&q);
}
```

В функцию передаётся двумерный массив типа `int`, название файла, массив типа `char header`, содержащий информацию о изображении. После чего производится инициализация объекта класса `image_t`, и запись в него значений. После проводится проверка на предмет того что изображение находится в цветовом пространстве RGB, после сего если изображение принадлежит к RGB происходит конвертация к цветовому пространству YCbCr.

После изображение и `header` передаются в функцию `fw_encode_ch` в которой производится собственно вейвлет-преобразование.

Функция спектрального преобразования `fw_encode_ch` выглядит следующим образом:

```
     Void fw_encode_ch(int **out,
                    int * channels,
                    unsigned char* header,
                    wavparams_t * p)
     {
         int k;
         size_t img_size = p->w * p->h;
         if (!fw_encode_header(header, p))
          {
             *out = _mm_malloc(sizeof(int*) * img_size * p-
>nb_channels, 16);
             for(k = 0; k < p->nb_channels; k++)
             {
                 fiwt2D_5_3_slow(&((*out)[k*img_size]), &chan-
nels[k*img_size], p);
             }
         }
```

```
}
```

Суть данной функции заключается в передаче параметров функциям fiwt2D_5_3_slow и fw_encode_header. Функция fw_encode _header выглядит следующим образом:

```
Int vfwl_encode_header(unsigned char *header, wavparams_t *p)
{
     if (ceil(log(p->h)/log(2)) != (int)(log(p->h)/log(2)) ||
        ceil(log(p->w)/log(2)) != (int)(log(p->w)/log(2)))
     {
          fprintf(stderr, "%s: The image is not of a size
power of 2\n", __FUNCTION__);
          return 1;
     }
     *(unsigned int *)header = p->nb_channels;
     *(unsigned int *)(header+INT_SIZE) = p->nb_levels;
     *(unsigned int *)(header+INT_SIZE*2) = p->h;
     *(unsigned int *)(header+INT_SIZE*3) = p->w;
     *(unsigned int *)(header+INT_SIZE*4) = p->bpp;
     return 0;
}
```

Данная функция осуществляет кодирование информации о файле для последующей записи в файл.

Функция fiwt2D_5_3_slow осуществляет основную стадию преобразования двумерного изображения.

```
void fiwt2D_5_3_slow(wav_t *mat_out, wav_t *mat_in, wavparams_t
* p) {
    ssize_t i, x, y, k, h, l = 0;
    ssize_t nn, ns2, nnm1;
    size_t width = p->w;
    size_t height = p->h;
    wav_t *in = NULL;
    wav_t *in_i;
    wav_t *in_im1;
    wav_t *in_im2;
    wav_t *out = NULL;
    wav_t *out_l;
    wav_t *out_h;
    nn = width;
    ns2 = width;
```

На данном участке кода происходит объявление переменных. После объявления переменных начинается стадия подсчёта высоких и низких коэффициентов для трансформации столбцов входной матрицы.

```
    for (k = p->nb_levels - 1; k >= 0; k--) {
```

```
        for(y = 0; y < height; y++) {
            /* Can't find a better optimization... */
            out = &mat_out[y*width];
            in  = &mat_in[y*width];
            /* Compute new length / 2 */
            ns2 = nn / 2;
            nnm1 = nn - 1;
            i = 2;
            l = 0;
            h = ns2;
            out[nn - 1] = in[nn - 1] - in[nn - 2];
            /* Primary prediction */
            out[ns2] = in[1] - ((in[0] + in[2] + 1) >> 1);
            i+=2;
            l++;
            for (h = ns2+1; h < nnm1; h++) {
                out[h] = in[i - 1] - ((in[i - 2] + in[i] + 1) /
2);
                out[l] = in[i - 2] + ((out[h - 1] + out[h] + 2)
/ 4);
                i+=2;
                l++;
            }
            out[ns2 - 1] = in[nn - 2] +
                        ((out[nn - 2] + out[nn - 1] + 2) >> 2);
            /* Primary update */
            out[0] = in[0] + out[ns2];
            memcpy(in, out, nn*sizeof(wav_t));
        }
```

После выполнения операций по поиску высоких и низких коэффициентов и записи их в массив in выполняется стадия подсчёта высоких и низких коэффициентов для трансформации строк входной матрицы.

```
        for(x = 0; x < width; x++) {
            /* Compute new length / 2 */
            ns2 = nn / 2;
            nnm1 = nn - 1;
            out = &mat_out[x];
            out_l = &mat_out[x];
            out_h = &mat_out[x+ns2*width];
            in  = &mat_in[x];
            in_i = &mat_in[x+2*width];
            in_im1 = &mat_in[x+width];
            in_im2 = &mat_in[x];
            i = 2;
            l = 0;
            h = ns2;
            out[(nn - 1)*width] = in[(nn - 1)*width] - in[(nn -
2)*width];
```

```
            out[ns2*width] = in[width] - ((in[0] + in[2*width] +
1) >> 1);

            i+=2;
            in_i += width*2;
            in_im1 += width*2;
            in_im2 += width*2;
            out_l+=width;
            out_h+=width;
            l++;
            for (h = ns2+1; h < nnm1; h++) {
                *out_h = *in_im1 - ((*in_im2 + *in_i + 1) / 2);
                *out_l = *in_im2 + ((out[(h - 1)*width] + *out_h
+ 2) / 4);
                i+=2;
                l++;
                in_i += width*2;
                in_im1 += width*2;
                in_im2 += width*2;
                out_l += width;
                out_h += width;
            }
            out = &mat_out[x];
            in  = &mat_in[x];
            out[(ns2 - 1)*width] = in[(nn - 2)*width] +
                                   ((out[(nn - 2)*width] +
                                   out[(nn - 1)*width] + 2) >> 2);
            out[0] = in[0] + out[ns2*width];
        }
        if (k != 0)
            memcpy(mat_in, mat_out, nn*width*sizeof(wav_t));
        nn /= 2;
    }
}
```

После выполнения операций по поиску высоких и низких коэффициентов для строк и столбцов входной матрицы выполняется операция конкатенация всех выходных коэффициентов, после чего все коэффициенты записываются при помощи `memcpy` в массив `mat_in`, переданный в функцию.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ СОЗДАНИЯ И ВНЕДРЕНИЯ ПРОГРАММНОГО КОМПЛЕКСА ДЛЯ СЖАТИЯ ГИПЕСПЕКТРАЛЬНЫХ ДАННЫХ

7.1 Характеристика программного средства

Целью дипломного проекта является разработка приложения для сжатия гиперспектральных изображений, полученных с гиперспектрометра Aviris.

В первую очередь приложение предназначено для внедрения в большинстве случаев в компании, занимающиеся в основном анализом почв, для сельскохозяйственных нужд, компаний занимающихся геодезической разведкой. Так как исходные данные полученные с гиперспектроматра занимают большой объём в памяти и сложны в обработке, такое инструмент будет незаменим для подобных компаний и серьёзно упростит им работу.

Разработка данного приложения осуществляется в IT-компании. Компания является резидентом Парка Высоких Технологий.

Организация-разработчик создает данный программный продукт по индивидуальному заказу.

Экономическая целесообразность инвестиций в разработку и реализацию данного программного продукта определяется на основе анализа таких показателей, как:

- смета затрат и отпускная цена программного продукта;
- прибыль от реализации программного продукта;
- рентабельность инвестиций в разработку программного продукта.

7.2 Расчет сметы затрат на разработку и отпускной цены ПО

Затраты на основную заработную плату команды разработчиков ($З_о$) определяются исходя из состава и численности команды, размеров месячной заработной платы каждого из участников команды, а также общей трудоемкости разработки программного продукта.

Основная заработная плата исполнителей, занятых разработкой программного средства, рассчитывается по формуле [10]:

$$З_о = \sum_{i=1}^{n} T_{чi} \cdot T_ч \cdot Ф_{эi} \cdot К, \quad (7.1)$$

где n – количество исполнителей, занятых разработкой конкретного программного продукта;

$T_{чi}$ – часовая тарифная ставка, руб.;

$Ф_{эi}$ – эффективный фонд рабочего времени i-го исполнителя, дней;

К – коэффициент премирования, 1,5;

$T_ч$ – количество часов работы в день, 8 часов.

В настоящий момент на предприятии тарифная ставка первого разряда установлена 250 рублей.

Расчетная норма рабочего времени на 2018 год для пятидневной рабочей недели составляет 168 часов. Среднемесячная расчетная норма рабочего времени на 2018 год равна 21 день. Коэффициент премирования равен 1,5.

Разработчиками программного средства являются двое исполнителей: программист первой категории и руководитель проекта. Руководитель проекта является ассистентом.

Расчет основной заработной платы на разработку программного продукта с учетом тарифного коэффициента и коэффициента премирования представлен в таблице 7.1

Таблица 7.1 – Расчет основной заработной платы исполнителей

Исполнитель	Разряд	Тарифный коэффициент ($Т_к$)	Часовая тарифная ставка ($Т_{чi}$), руб.	Плановый фонд рабочего времени ($Ф_п$), дн.	Коэффициент премирования	Заработная плата ($З_о$), руб.
1	2	3	4	5	6	7
Руководитель проекта	14	3,25	4,84	40	1,5	2323,20
Программист первой категории	10	2,48	3,69	60	1,5	2657,14
Основная заработная плата без учета премии						3320,23
Основная заработная плата с учетом премии						4980,34

Дополнительная заработная плата ($З_д$) определяется по нормативу в процентах к основной заработной плате:

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

где $З_д$ – дополнительная заработная плата исполнителей на конкретное программное обеспечение, руб.;

$Н_д$ – норматив дополнительной заработной платы, 15%.

Использую формулу 7.2 получим:

$$З_д = \frac{4980{,}34 \cdot 15}{100} = 747{,}05 \text{ руб},$$

К затратам и отчислениям в фонд социальной защиты населения и обязательного страхования от несчастных случаев на производстве относят отчисления в фонд социальной защиты населения ($Н_{сз}$) и отчисления на обязательное страхование от несчастных случаев ($Н_{стр}$). $Н_{сз}$ составляет 34%, а равен $Н_{стр}$ 0,6%. Отчисления на социальные нужды определяются в соответствии с формулой:

$$З_{сз} = \frac{(З_о + З_д) \cdot (Н_{сз} + Н_{стр})}{100}. \quad (7.3)$$

Используя рассчитанные ранее дополнительную заработную плату и основную заработную плату, получим сумму отчислений в фонд социальной защиты населения и обязательного страхования от несчастных случаев в соответствии с формулой 7.3:

$$З_{сз} = \frac{(4980{,}34 + 747{,}05) \cdot (34 + 0{,}6)}{100} = 1981{,}68 \text{ руб.}$$

Далее рассчитаем расходы на оплату машинного времени. Расходы на оплату машинного времени включают оплату машинного времени, необходимого для разработки и отладки программного продукта, и определяются по следующей формуле:

$$Р_м = Ц_м \cdot Т_ч \cdot С_р, \quad (7.4)$$

где $Ц_м$ – цена одного машино-часа, руб.;

$Т_ч$ – количество часов работы в день;

$С_р$ – длительность проекта.

Стоимость машино-часа в компании составляет 2 рубля. Длительность разработки проекта составляет 60 дней.

$$Р_м = 2 \cdot 8 \cdot 60 = 960 \text{ руб.}$$

Прочие затраты можно рассчитать, используя формулу:

$$П_з = \frac{З_о \cdot Н_{пз}}{100}, \quad (7.5)$$

где $Н_{пз}$ – норматив прочих затрат в целом по организации. Для данной организации $Н_{пз}$ равен 15%.

Прочие затраты идут на приобретение лицензий для программного обеспечения, необходимого для сотрудников организации, также включают затраты на оплату услуг связи, Интернета, транспортные расходы, расходы на канцтовары, затраты, связанные с приобретением и подготовкой необходимой научно-технической информации и специальной технической литературы.

Используя формулу 7.5, рассчитаем сумму прочих затрат для разрабатываемого программного продукта:

$$П_З = \frac{4980{,}34 \cdot 15}{100} = 747{,}05 \text{ руб.}$$

Расходы по статье «Накладные расходы» включают в себя расходы, связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных и/или экспериментальных производств, а также с расходами на общехозяйственные нужды ($Р_н$), определяются по следующей формуле:

$$Р_н = \frac{З_о \cdot Н_{рн}}{100}, \quad (7.6)$$

где $Н_{рн}$ – норматив накладных расходов в целом по организации.

В компании, в которой разрабатывается программный продукт, норматив накладных расходов составляет 40%. Рассчитаем расходы по статье «Накладные расходы» по формуле 7.6:

$$Р_н = \frac{4980{,}34 \cdot 40}{100} = 1992{,}14 \text{ руб.}$$

Общая сумма расходов по всем статьям сметы ($С_р$) на разработку программного продукта включает в себя основную заработную плату и дополнительную заработную плату, расходы на оплату машинного времени, расходы по статье «Накладные расходы» и прочие расходы. Общую сумму расходов можно рассчитать по следующей формуле:

$$С_р = З_о + З_д + З_{сд} + Р_м + П_з + Р_н , \quad (7.7)$$

Затраты на сопровождение и адаптацию программного продукта рассчитываются по следующей формуле:

$$Р_с = \frac{С_р \cdot Н_{рса}}{100}, \quad (7.8)$$

где $Н_{рса}$ – норматив расходов на сопровождение и адаптацию программного продукта, 15%.

Использую формулу 7.7, рассчитаем общую сумму расходов:

$$С_р = 4980{,}34 + 747{,}05 + 1981{,}68 + 960 + 747{,}05 + 1992{,}14 = \\ = 11408{,}26 \text{ руб.}$$

Рассчитаем затраты на сопровождение и адаптацию программного продукта по формуле 7.8:

$$Р_с = \frac{11408{,}26 \cdot 15}{100} = 1711{,}24 \text{ руб.}$$

Общая сумма расходов на разработку программного продукта с затратами на сопровождение и адаптацию в качестве полной себестоимости программного продукта ($С_п$) определяется по следующей формуле:

$$С_п = С_р + Р_с, \quad (7.9)$$

Используя формулу 7.9, рассчитаем полную себестоимость разрабатываемого программного продукта:

$$С_п = 11408{,}26 + 1711{,}24 = 13119{,}5 \text{ руб.}$$

Плановая прибыль, которая включается в цену программного обеспечения рассчитывается следующим образом:

$$П_{пс} = \frac{С_п \cdot У_{рп}}{100}, \quad (7.10)$$

где $У_{рп}$ – уровень рентабельности программного продукта. В компании он составляет 20%. Рассчитаем плановую прибыль по формуле 7.10:

$$П_{пс} = \frac{13119{,}5 \cdot 20}{100} = 2623{,}9 \text{ руб.}$$

Согласно законодательству Республики Беларусь, предприятия, не являющиеся резидентами Парка Высоких Технологий не освобождаются от налога на прибыль и налога на добавленную стоимость. Поэтому отпускная цена рассчитывается с учетом НДС:

$$Ц_о = З_п + НДС, \quad (7.11)$$

где НДС - сумма налога на добавленную стоимость, руб;

$З_п$ – затраты с учетом прибыли, руб.

НДС можно рассчитать по следующей формуле:

$$НДС = \frac{З_п \cdot 20}{100}, \quad (7.12)$$

Цена с учетом прибыли определяется по формуле:

$$З_п = С_п + П_{пс}. \quad (7.13)$$

Рассчитаем цену с учетом прибыли по формуле 7.13:

$$З_п = 13119{,}5 + 2623{,}9 = 15743{,}4 \text{ руб.}$$

Таким образом отпускная цена будет равна:

$$Ц_о = 15743{,}4 + \frac{15743{,}4 \cdot 20}{100} = 18892{,}08 \text{ руб.}$$

Таблица 7.2 – Смета затрат и отпускная цена

Наименование статьи	Условное обозначение	Значение, руб.	Методика расчета
1	2	3	4
Основная заработная плата	$З_о$	4980,34	На основании расчетов
Дополнительная заработная плата	$З_д$	747,05	$З_д = \frac{З_о \cdot Н_д}{100}$
Отчисления на социальные нужды	$З_{сз}$	1981,68	$З_{сз} = \frac{(З_о + З_д) \cdot (Н_{сз} + Н_{стр})}{100}$
Машинное время	$Р_м$	980	На основании расчетов
Прочие прямые расходы	$П_з$	747,05	$П_з = \frac{З_о \cdot Н_{пз}}{100}$
Накладные расходы	$Р_н$	1992,14	$Р_н = \frac{З_о \cdot Н_{рн}}{100}$

Продолжение таблицы 7.2

Общая сумма расходов по всем статьям сметы	$С_р$	11408,26	$С_р = З_о + З_д + З_{сд} + Р_м + П_з + Р_н$
Затраты на сопровождение и адаптацию	$Р_с$	1711,24	$Р_с = \frac{С_р \cdot Н_{рса}}{100}$
Полная себестоимость	$С_п$	13119,5	$С_п = С_р + Р_с$
Плановая прибыль	$П_{пс}$	2623,9	$П_{пс} = \frac{С_п \cdot У_{рп}}{100}$
Затраты с учетом прибыли	$З_п$	15743,4	$З_п = С_п + П_{пс}$
Отпускная цена	$Ц_о$	18892,08	$Ц_о = С_п + П_{пс} + НДС$
Договорная цена	$Ц_д$	20000	На основании переговоров

Поскольку в данном случае программный продукт является уникальным, то есть создается под нужды и требования конкретного заказчика, то его договорная цена составила $Ц_д = 20000$ руб. Цена получена в результате переговоров. Все расчеты отпускной цены и статей сметы затрат с учетом заработной платы представлены ниже в таблице 7.2.

7.3 Расчет экономического эффекта у разработчика ПО

Эффект от применения программного продукта, который получит пользователь рассчитать не удалось, так как информация об использовании разработанного продукта является конфиденциальной. В результате использования данного программного продукта заказчик увеличит количество участников конкурса и инвесторов, а также увеличится эффективность деятельности благодаря осуществлению операционного управления процесса проведения грантовых конкурсов.

Экономический эффект организации-разработчика представляет собой

прибыль, полученную от разработки программного продукта под заказ сторонней организации.

Таким образом, рассчитаем экономический эффект у разработчика приложения для распределения грантов для некоммерческих организаций по следующим формулам.

Прибыль от реализации для цены реализации программного продукта заказчику $Ц_д$ и суммы расходов $С_п$ рассчитывается следующим образом:

$$П_р = Ц_д - С_п. \quad (7.14)$$

Используя формулу 7.14, получим:

$$П_р = 20000 - 13119{,}5 = 6880{,}5 \text{ руб.}$$

При определении чистой прибыли предприятия следует учитывать налог на прибыль. Чистая прибыль предприятия с учетом налога на прибыль определяется по формуле:

$$П_ч = П_р - \frac{П_р \cdot Н_п}{100\%}, \quad (7.15)$$

где $Н_п$ – ставка налога на прибыль, 18%.

Рассчитаем чистую прибыль предприятия с учетом налога на прибыль по формуле 7.15:

$$П_ч = 6880{,}5 - \frac{6880{,}5 \cdot 18}{100} = 5642{,}01 \text{ руб.}$$

Для определения фактического уровня рентабельности воспользуемся следующей формулой:

$$У_р = \frac{П_ч}{С_п} \cdot 100\%. \quad (7.16)$$

Используя формулу 7.16, рассчитаем фактический уровень рентабельности:

$$У_р = \frac{5642{,}01}{13119{,}5} \cdot 100\% = 43\%.$$

Сопоставим запланированный уровень рентабельности (20%) с фактическим:

$$\Delta = У_{рен}^{фа} - У_{рен}^{пл}. \quad (7.17)$$

где $У_{рен}^{фа}$ – фактический уровень рентабельности;

$У_{рен}^{пл}$ – запланированный уровень рентабельности.

Из формулы 7.17 получим:

$$\Delta = 43 - 20 = 23\%.$$

Таким образом, фактический уровень рентабельности выше запланированного.

В результате технико-экономического обоснования применения программного продукта были получены следующие значения показателей их эффективности:

– общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) и прогнозируемая отпускная цена составляют соответственно 13119,5 рублей и 18892,08 рублей;

– прибыль компании-разработчика от продажи программного средства составит 6880,5 рублей;

– уровень рентабельности инвестиций для IT-компании составляет 43%, что выше запланированных 20%;

Анализ полученных показателей показывает, что разработка и применение данного программного продукта являются эффективными.