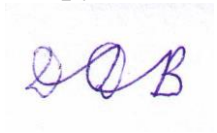
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский Томский государственный университет»

На правах рукописи



Дружинин Денис Вячеславович

АЛГОРИТМИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СЖАТИЯ БЕЗ ПОТЕРЬ ВИДЕОДАННЫХ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

05.13.11 – Математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

Диссертация
на соискание ученой степени
кандидата технических наук

Научный руководитель
доктор технических наук, доцент
Замятин Александр Владимирович

Томск – 2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы исследования. В начале XXI века создание и применение видео различной природы стало доступно широкому кругу пользователей. Один из основных типов видео, распространенных среди пользователей персональных компьютеров (ПК), – видео графического интерфейса пользователя (Graphical User Interface видео, GUI-видео). Такое видео часто создают как необходимое дополнение к руководству пользователя информационно-программных комплексов или к описанию сценариев воспроизведения ошибок при взаимодействии команд разработчиков и инженеров по качеству программного обеспечения (ПО) для повышения эффективности их коммуникации [129, 130].

Ввиду сравнительно значительного объёма *видеоданных* (данных, получаемых в ходе формирования видео) возникают сложности, связанные с необходимостью отводить большой объём дискового пространства для их хранения. Как следствие, при решении задачи обработки GUI-видеоданных появляется условие минимизации их объёма. Дополнительно эта задача осложняется тем, что GUI-видеоданные характеризуются высоким разрешением кадров.

Кроме того, приложения, осуществляющие фиксацию GUI-видео, выполняют лишь вспомогательную функцию, работают одновременно с другими приложениями и сервисами операционной системы, и по этой причине должны исполняться в *фоновом* (низкоприоритетном) режиме. Поэтому необходимо минимизировать уровень использования ими системных ресурсов компьютера, требуемых для эффективного исполнения основных задач пользователя. Как следствие, появляется условие повышения вычислительной эффективности алгоритмов фиксации и сжатия GUI-видеоданных при их максимальной ресурсоэффективности. Под высокой ресурсоэффективностью будем понимать здесь минимальный уровень использования центрального процессора и оперативной памяти.

С середины XX века выполнен большой объём исследований теоретического и практического характера в области сжатия данных такими отечественными и зарубежными учёными как Котельников В.А., D. Huffman, A. Lempel, J. Ziv и многими другими [56, 82, 88, 105, 123, 145]. В области сжатия графических данных, в том числе видеоданных, получены результаты, как предполагающие потери информации, так и обеспечивающие сжатие без потерь. В первой группе исследований следует отметить работы таких учёных как Бабкин В.Ф., Тропченко А.Ю., G. Sullivan, Y. Lu, M.R. El-Sakka, D. Taubman, A.A. Rodriguez [1, 2, 26, 27, 34, 44, 46, 56, 67, 68, 133]. Полученные ими результаты нашли широкое практическое применение, преимущественно предназначены для видео с видеокамеры (*традиционного* видео) и используют особенности этого типа видеоданных, заключающиеся в преобладании непрерывно-тоновых (плавных) цветовых переходов в кадрах. Соответствующие алгоритмы сжатия видеоданных, как правило, предполагают устранение визуально малозаметных элементов изображения и потери информации.

GUI-видеоданные существенно отличаются от традиционных видеоданных, что в значительной мере ограничивает возможности применения к ним традиционных алгоритмов сжатия. Так, в GUI-видеоданных преобладают дискретно-тоновые (резкие) цветовые переходы. Такие видеоданные, как правило, целесообразно сжимать без потерь информации специализированными алгоритмами, так как даже небольшая доля потерь может привести к неприемлемому визуальному ухудшению качества видеоданных.

Во второй группе исследований, посвящённых сжатию графических данных без потерь информации, следует отметить работы таких учёных как B. Carpentieri, J.J. Ding, M. Hernández-Cabronero, M.W. Marcellin, I. Blanes, K. Hirakawa [47, 54, 64, 69, 72, 77, 88, 92, 99, 105, 142]. Однако, лишь в отдельных исследованиях таких учёных как S. Mehrotra, P. Dondi, L. Lombardi, S. Wang, J. Xu, R. Joshi, R.A. Cohen учитываются особенности GUI-видео [96, 64, 93, 94, 96, 99, 100, 110, 137, 141]. Многие существующие алгоритмы, применяемые для сжатия GUI-видеоданных, не учитывают в полной мере отличия GUI-видео от традиционного

видео, что приводит к низкой степени сжатия либо к низкому качеству декодированных видеоданных в случае сжатия с потерями информации. Часть алгоритмов сжатия не обладает достаточной вычислительной эффективностью и ресурсоэффективностью, что так же препятствует их использованию для сжатия GUI-видеоданных в фоновом режиме. Поэтому проблема создания эффективных алгоритмов сжатия GUI-видеоданных всё ещё остаётся малоизученной.

Учитывая всё вышеизложенное, можно сделать вывод об актуальности проблемы создания новых и модификации существующих алгоритмов сжатия без потерь информации, предназначенных для обработки GUI-видеоданных и отличающихся более высокими показателями эффективности, позволяющими производить сжатие GUI-видеоданных без потерь информации не только с высокой степенью сжатия, но и с минимальным использованием вычислительных ресурсов компьютера.

Цель работы и задачи исследования. Целью диссертационной работы является разработка алгоритмического и программного обеспечения фиксации и локального сохранения GUI-видеоданных со сжатием без потерь с высокой степенью сжатия, обладающего высокими показателями ресурсо- и вычислительной эффективности.

Для достижения поставленной цели необходимо последовательное решение следующих задач:

1. Анализ существующего алгоритмического и программного обеспечения сжатия видеоданных, особенностей традиционного видео и GUI-видео, позволяющих определить основные направления исследования в области построения эффективных алгоритмов сжатия GUI-видеоданных.

2. Разработка ресурсо- и вычислительно эффективных алгоритмов сжатия без потерь информации, обеспечивающих высокую степень сжатия GUI-видеоданных. Решение данной задачи предполагает также исследование эффективности предлагаемых алгоритмов.

3. Создание высокопроизводительного ядра кодека (англ. – codec, в данном случае – программное обеспечение преобразования видеоданных),

предназначенного для сжатия и восстановления GUI-видеоданных. Результатом решения этой задачи должны явиться программные средства такого кодека, реализующие предложенные оригинальные алгоритмы.

4. Апробация разработанного кодека с использованием различных видеофрагментов при решении экспериментальных и реальных задач.

Методы исследования и достоверность полученных результатов. В работе использованы методы теории сжатия информации, теории обработки изображений, теории алгоритмов и математической статистики. Экспериментальные исследования выполнены с использованием программной реализации алгоритмов с последующей оценкой полученных результатов и сравнением с экспериментальными данными в специальной литературе. Достоверность полученных результатов подтверждена применением методов математической статистики и успешными результатами практической апробации алгоритмического и программного обеспечения.

Научная новизна. Научную новизну полученных в работе результатов определяют:

1. Алгоритм пространственного группового кодирования, отличающийся более полным учётом горизонтальной и вертикальной корреляции пикселей кадра за счёт выявления большего количества пространственных объектов в нём, и обеспечивающий большую степень сжатия GUI-видеоданные по сравнению с аналогами.

2. Алгоритм сжатия со сниженной пространственной избыточностью, отличающийся единовременным устранением основных типов пространственной избыточности GUI-видеоданных (частое чередование цветов, одноцветные области, градиентные переходы), и позволяющий существенно увеличить степень сжатия таких данных.

3. Алгоритм оценки движения с учётом классификационных признаков, отличающийся способом выбора векторов движения исключительно в наиболее вероятных направлениях перемещения объектов GUI-видеоданных, и

превосходящий аналоги в вычислительной эффективности при сохранении высокой степени сжатия и ресурсоэффективности.

4. Архитектура кодека сжатия GUI-видеоданных, характеризующаяся наличием подсистемы обработки данных на видеокарте и динамическим подключением модулей сжатия, и позволяющая построить программное обеспечение кодека, обеспечивающего высокую степень сжатия данных при высоких показателях ресурсо- и вычислительной эффективности.

Положения, выносимые на защиту:

1. Алгоритм сжатия со сниженной пространственной избыточностью позволяет повысить по сравнению с аналогами степень сжатия дискретно-тоновых изображений при сопоставимых вычислительных затратах.

2. Алгоритм оценки движения с учётом классификационных признаков позволяет существенно повысить вычислительную эффективность по сравнению с аналогами при сохранении высокой степени сжатия.

3. Разработанное алгоритмическое и программное обеспечение кодека преобразования GUI-видеоданных превосходит аналогичные кодеки по степени сжатия и вычислительной эффективности, а также упрощает и ускоряет создание программных систем с его использованием.

Публикации. По теме диссертации опубликовано 16 работ, из них 5 статей, в том числе 3 статьи в ведущих рецензируемых научных журналах, входящих в перечень ВАК. Кодек, созданный с применением алгоритмов, предложенных и описанных автором в диссертационной работе, зарегистрирован в качестве программы для ЭВМ [36].

Апробация результатов исследования. Основные положения и отдельные результаты исследования докладывались и обсуждались на следующих конференциях: VI Международная научно-практическая конференция «Информационные технологии и математическое моделирование» (Анжеро-Судженск, 2007 г.), XLVIII международная научная студенческая конференция «Студент и научно-технический прогресс» (Новосибирск, 2010 г.), IX

Всероссийская научно-практическая конференция с международным участием «Информационные технологии и математическое моделирование» (Анжеро-Судженск, 2010 г.), XV Всероссийская научно-практическая конференция «Научное творчество молодёжи» (Анжеро-Судженск, 2011 г.), X Всероссийская научно-практическая конференция с международным участием «Информационные технологии и математическое моделирование» (Анжеро-Судженск, 2011 г.), XII Всероссийская научно-практическая конференция с международным участием им. А. Ф. Терпугова (Анжеро-Судженск, 2013 г.), 52-я международная научная студенческая конференция МНСК-2014 (Новосибирск, 2014 г.), XVIII Всероссийская научно-практическая конференция «Научное творчество молодежи. Математика. Информатика» (Анжеро-Судженск, 2014 г.), II Всероссийская молодежная научная конференция с международным участием «Математическое и программное обеспечение информационных, технических и экономических систем» (Томск, 2014 г.), Всероссийская конференция «XII Сибирская научная школа-семинар с международным участием Компьютерная безопасность и криптография – SIBECRYPT'14» (Екатеринбург, 2014 г.), Всероссийская научно-практическая конференция «Информационно-телекоммуникационные системы и технологии» (Кемерово, 2014 г.), XV Международная конференция имени А.Ф. Терпугова «Информационные технологии и математическое моделирование» (пос. Катунь, 2016 г.), XVI Международная конференция имени А.Ф. Терпугова «Информационные технологии и математическое моделирование» (Казань, 2017 г.).

Теоретическая и практическая значимость. Теоретическая значимость диссертационной работы заключается в том, что разработанное алгоритмическое обеспечение обладает высокими показателями ресурсо- и вычислительной эффективности, позволяет сжимать GUI-видеоданные без потерь с высокой степенью сжатия, а также позволяет сформировать научно-методический задел для создания программного обеспечения фиксации и локального сохранения GUI-видеоданных.

Практически значимыми являются предложенное в работе семейство алгоритмов, а также ПО кодека сжатия GUI-видеоданных, созданное на их основе. Программные средства кодека функционируют на компьютерах типа IBM PC под управлением Windows XP/Vista/7/8/10. Объём исходного кода разработанного ПО составляет более 8000 строк на языке C++. Кодек сжатия GUI-видеоданных внедрён в Югорском НИИ информационных технологий при создании пользовательского приложения для фиксации GUI-видео, а также в компании-разработчике ПО ООО «Армадэйт» в качестве самостоятельного программного продукта. Результаты внедрения подтверждены соответствующими актами.

Объём и структура работы. Диссертация состоит из введения, четырёх глав, списка использованных источников и четырёх приложений. Текст изложен на 161 странице. Список использованных источников и литературы включает 146 наименований. В работе представлено 28 рисунков и 20 таблиц (6 таблиц в основном тексте и 14 – в приложениях).

В первой главе проводится анализ результатов исследований в области сжатия GUI-видеоданных. Рассмотрены отличия традиционного видео и GUI-видео. Делается вывод о том, что GUI-видео в значительной степени отличается от традиционного видео. Отдельно рассматриваются алгоритмы сжатия ключевых кадров, обрабатываемых независимо от данных в смежных кадров, и промежуточных кадров, сжимаемых с использованием данных в смежных кадрах.

Приводится классификация изображений на монохроматическое, полутоновое, а также цветное с непрерывным тоном и цветное дискретно-тоновое изображение. Подробно рассмотрены цветные дискретно-тоновые изображения, так как кадры GUI-видео в большинстве своём относятся к этому классу изображений. Рассмотрены алгоритмы сжатия изображений, которые могут применяться для сжатия дискретно-тоновых изображений. В частности рассмотрены алгоритмы словарного, статистического сжатия, группового кодирования, алгоритмы классификации блоков изображений.

Рассмотрены существующие алгоритмы сжатия промежуточных кадров в видеопотоке, такие как алгоритм отсечения неизменившихся блоков, алгоритм

оценки движения. Делается вывод о том, что существующие алгоритмы не соответствуют в полной мере требованиям, предъявляемым к алгоритмам сжатия кадров GUI-видео. На основе результатов анализа проблемы сжатия GUI-видеоданных формулируются цель и задачи диссертационной работы.

Во второй главе представлено семейство алгоритмов сжатия. Эти алгоритмы, перечисленные ниже, обеспечивают высокую степень сжатия дискретно-тоновых изображений и обладают высокой вычислительной эффективностью. Представлен сдвиговый алгоритм, относящийся к группе словарных алгоритмов сжатия. Представлена модификация алгоритма группового кодирования, учитывающая одновременно горизонтальную и вертикальную корреляцию пикселей. Представлен гибридный сдвигово-групповой алгоритм, соединяющий в себе черты сдвигового алгоритма и представленной модификации алгоритма группового кодирования. Представлены два алгоритма со сниженной пространственной избыточностью на основе гибридного сдвигово-группового алгоритма, предполагающие выполнение гибридного алгоритма на первом этапе. Один из представленных алгоритмов предполагает выполнение алгоритмов LZW и Хаффмана на втором этапе. Другой алгоритм предполагает выполнение алгоритмов LZO и Хаффмана на втором этапе. Представлены несколько алгоритмов, адаптированных для сжатия промежуточных кадров GUI-видео. Алгоритм отсечения неизменившихся строк и столбцов выявляет строки и столбцы, в которых присутствуют изменившиеся пиксели. Адаптивный алгоритм отсечения неизменившихся областей кадра осуществляет выбор алгоритма с более высокой степенью сжатия для текущей пары кадров. В качестве конкретного алгоритма может быть выбран алгоритм отсечения неизменившихся блоков или алгоритм отсечения неизменившихся строк и столбцов. Предлагается классификация движений в GUI-видео, что сделало возможным поиск заданных классов движений. Предложен оригинальный алгоритм оценки движения с учётом классификационных признаков, использующий информацию о вероятности встречаемости типов движений в GUI-видео. Разработанный алгоритм отсечения неизменившихся блоков целесообразно использовать в качестве предобработки

для выявления минимального охватывающего прямоугольника, в пределах которого затем проводится оценка движения. Предлагается оптимизирующая техника сравнения блоков, заключающаяся в предварительном сравнении диагональных элементов блоков. Сформулированы требования к программному обеспечению кодека для сжатия GUI-видеоданных. Предложена обобщённая архитектура такого кодека.

В третьей главе проанализированы особенности практической реализации алгоритмов сжатия GUI-видеоданных. Предложен способ формирования вспомогательных структур данных, позволяющих повысить вычислительную эффективность кодирования и декодирования гибридного сдвигово-группового алгоритма. Предложен способ формирования служебных данных, записываемых гибридным сдвигово-групповым алгоритмом в результирующий массив, учитывающий частоту встречаемости групп пикселей различных типов. Описаны различные модификации гибридного сдвигово-группового алгоритма. Приводятся результаты сравнительного анализа представленных и существующих алгоритмов сжатия. Делается вывод о том, что алгоритм со сниженной пространственной избыточностью, основанный на гибридном сдвигово-групповом алгоритме и алгоритмах LZW, Хаффмана превосходит другие алгоритмы по совокупности показателей. Представлены результаты практического сравнения алгоритмов отсечения неизменившихся областей кадра. Показано, что на всех тестовых наборах данных адаптивный алгоритм способен выбрать конкретный алгоритм отсечения неизменившихся областей кадра, обеспечивающий более высокую степень сжатия. Приводятся результаты сравнительного анализа разработанного алгоритма оценки движения с учётом классификационных признаков и алгоритма, выявляющего все типы движений. Делается вывод о том, что предложенный алгоритм оценки движения позволяет ускорить выполнение оценки движения до десяти раз при незначительных потерях в количестве распознанных движений.

В четвёртой главе представлено описание программного обеспечения кодека для обработки GUI-видеоданных. Основу этого ПО составляют группы

подсистем кодера и декодера, содержащие реализации предложенных алгоритмов и взаимодействующие с независимыми подсистемами как входящими в состав кодека, так и программными средствами других производителей. Описан объектный и функциональный интерфейсы кодека, обеспечивающие доступ к идентичному набору возможностей кодека. Наличие таких интерфейсов кодека позволяет упростить его использование в приложениях, написанных как на функциональных, так и на объектно-ориентированных языках программирования. Описана технология сжатия данных, реализованная в этом кодеке. Представлена подсистема повышения производительности обработки данных с помощью видеокарты. Делается вывод о том, что реализации алгоритмов сжатия GUI-видеоданных, использующие технологию Nvidia CUDA для доступа к вычислительным ресурсам видеокарты, работают со скоростью, схожей со скоростью реализации, использующей только вычислительные ресурсы центрального процессора (ЦП-реализации). При этом удаётся добиться повышения ресурсоэффективности за счёт высвобождения вычислительных ресурсов ЦП для выполнения других операций. Приводятся результаты сравнительного анализа разработанного кодека и существующих кодеков. Делается вывод о том, что представленный кодек по совокупности характеристик превосходит аналоги.

В приложении А приведена копия свидетельства о государственной регистрации программы для ЭВМ «Butterfly Screen Video Codec». В приложении Б приведены копии актов о внедрении полученных результатов. Приложение В содержит подробные результаты тестирования различных реализаций алгоритмов сжатия. Приложение Г содержит описание функционального интерфейса разработанного кодека.

Благодарности. Автор выражает искреннюю признательность своему научному руководителю – доктору технических наук, доценту Замятину А.В. за неоценимую помощь при подготовке научных статей и диссертации на всех этапах её создания. Особая благодарность кандидату технических наук Лаврову В.А. за ценные замечания и рекомендации, доктору технических наук,

профессору Костюку Ю.Л. за помощь в повышении качества научных статей соискателя.

ГЛАВА 1. АНАЛИЗ СОСТОЯНИЯ ПРОБЛЕМЫ

В этой главе приводится аналитический обзор алгоритмического и программного обеспечения, применяемого при решении задачи сжатия GUI-видеоданных. Проанализированы отличия GUI-видео от традиционного видео, влияние этих отличий на эффективность применения традиционных алгоритмов для сжатия GUI-видеоданных. Рассмотрены существующие алгоритмы сжатия GUI-видеоданных, устраняющие пространственную и временну́ю избыточность, а также перспективные методы повышения эффективности сжатия GUI-видеоданных.

1.1. Сравнительный анализ видов видеоданных

Для изложения сути основных видов видеоданных и проведения их сравнительного анализа введём некоторые обозначения.

Степень сжатия – это отношение размера несжатых данных к размеру соответствующих им сжатых данных [4]. Максимальную степень сжатия данных каким-либо алгоритмом сжатия обозначим C_{max}, минимальную – C_{min}.

Видеоданные разделяют на *традиционные* (полученные с видеокамеры) и видеоданные *графического интерфейса пользователя* (*Graphical User Interface видеоданные, GUI-видеоданные*) – отображение экрана персонального компьютера (ПК).

Кодек (англ. – codec) – программное обеспечение (ПО) преобразования видеоданных. В составе кодека будем выделять кодер и декодер, применяемые для кодирования и декодирования данных, соответственно.

RGB (Red, Green, Blue – красный, зелёный, синий) – цветовая модель, в которой пиксель представлен тремя компонентами (R-, G-, B-компоненты).

С ростом возможностей ПК большое количество пользователей получило возможность создавать, распространять и практически применять GUI-видео. GUI-видео часто создают как необходимое дополнение к руководству

пользователя информационно-программных комплексов или к описанию сценария воспроизведения ошибок при взаимодействии команд разработчиков и инженеров по качеству ПО для повышения эффективности их коммуникации [129, 130]. Кроме того GUI-видео создаётся рядовыми пользователями в режиме демонстрации собственного экрана компьютера с графическим интерфейсом пользователя собеседнику посредством телекоммуникационного ПО (Skype, Google Hangouts и др.) [76, 125].

Ввиду сравнительно значительного объёма *видеоданных* (данных, получаемых в ходе формирования видео), возникают сложности, связанные с необходимостью отводить большое количество дискового пространства для хранения видеоданных, а также использовать каналы связи высокой пропускной способности для передачи таких данных. Как следствие, появляется задача минимизации как суммарного объёма передаваемых или хранимых видеоданных, так и *битрейта* (максимального количества передаваемых между узлами в сети данных за единицу времени).

Необходимость за ограниченное время сжимать изображения возникает не только при обработке GUI-видеоданных. Такая потребность возникает, например, в задачах дистанционного зондирования Земли, где требуется оперативно сжимать значительные объёмы данных, представляющих собой статические изображения. При этом вычисления проводят на бортовом компьютере спутника и пересылают по каналам связи, обладающим сравнительно невысокой пропускной способностью [26, 35, 143, 22, 28, 23, 142, 24, 25].

ПО видеокомпрессии разделяют на две группы – симметричное и ассиметричное [4].

Ассиметричное ПО предъявляет серьёзные требования к декодеру (как правило по времени и памяти), но для них уровень затрат ресурсов при кодировании не является критичным параметром. Примером являются различные мультимедиа-энциклопедии, путеводители, справочники, игры и просто фильмы. При такой постановке задачи появляется возможность применить сложные

алгоритмы компрессии, позволяющие получить большую степень сжатия данных [75, 116].

Симметричное ПО предъявляет одинаково жёсткие требования на время, память и другие ресурсы компьютера как при кодировании, так и при декодировании. Примерами такого рода приложений могут служить видео-почта, видеотелефон, видеоконференции [76, 125].

ПО, осуществляющее фиксацию GUI-видеоданных, разделяют на несколько групп, представленных в таблице 1.1.

Таблица 1.1 – Группы ПО для фиксации GUI-видеоданных

№	**Группа ПО**		**Тип**
1	Для мультимедийного общения (Skype, Google Hangouts и др.) [76, 125]		Симметричное
2.a	Для фиксации и локального сохранения GUI-видео	без оперативного сжатия	Ассиметричное
2.b		с оперативным сжатием	Симметричное

Стоит отметить, что приложения 1-ой группы оперируют GUI-видеоданными в режиме демонстрации экрана компьютера собеседнику, в оперативном режиме отправляя полученные GUI-видеоданные по сети на компьютер собеседника, где выполняется их визуализация, поэтому такие приложения относятся к симметричным. При реализации приложений для мультимедийного общения важно минимизировать как битрейт, так и суммарный объём передаваемых видеоданных [32].

В диссертационной работе решается задача разработки алгоритмического и программного обеспечения для фиксации и локального сохранения GUI-видео.

Приложения группы 2.a относятся к ассиметричным приложениям, так как предполагают сжатие видеоданных в качестве отдельного этапа их обработки, выполняемого уже по завершении фиксации и записи видеоданных на жёсткий диск. При таком подходе проявляется несколько недостатков по сравнению с подходом, предполагающим оперативное сжатие данных:

1. Требуется значительно больше дискового пространства. На практике при записи GUI-видеоданных без оперативного сжатия в минуту расходуется

несколько ГБ дискового пространства, в то время как с оперативным сжатием – порядка нескольких десятков МБ.

2. Требуется дополнительный этап обработки видеоданных, что увеличивает время, необходимое для формирования итогового видеофайла.

3. В случае записи на SSD диск значительно быстрее расходуется его ресурс, напрямую связанный с количеством циклов перезаписи [105].

4. В случае записи на HDD значительно уменьшается производительность записи данных другими приложениями за счёт необходимости постоянного перемещения считывающей головки в разные сектора [105]. По результатам экспериментальных исследований на современном HDD уменьшение скорости записи составляет до 3-х раз, в то время как при оперативном сжатии эта проблема проявляется в гораздо меньшей степени (таблица 14, приложение В).

Поэтому предпочтительным вариантом во второй группе является вариант с оперативным сжатием, именно он и будет рассматриваться в дальнейшем в диссертационной работе.

Компонент приложений для фиксации и локального сохранения GUI-видео, осуществляющий кодирование GUI-видеоданных, должен выполнять лишь вспомогательную функцию и работать в *фоновом* (низкоприоритетном) режиме, не влияя на работоспособность основных приложений пользователя. Поэтому необходимо минимизировать уровень использования этим компонентом системных ресурсов компьютера, требуемых для эффективного исполнения основных задач пользователя. Как следствие, появляется задача повышения вычислительной эффективности алгоритмов фиксации и сжатия GUI-видеоданных при максимальной ресурсоэффективности. Под высокой ресурсоэффективностью будем понимать здесь минимальный уровень использования центрального процессора и оперативной памяти.

Видеоданные представляют собой последовательность кадров. Рассмотрим отличия GUI-видео от традиционного видео на уровне отдельных кадров, а затем особенности межкадровых взаимосвязей в GUI-видеоданных. Это позволит

учесть различные их свойства при анализе и построении алгоритмов сжатия таких видеоданных.

Кадры видео разделяют на ключевые, сжимаемые независимо от других кадров, и промежуточные, сжимаемые с использованием информации о смежных кадрах. Сжатие ключевых кадров происходит за счёт уменьшения пространственной избыточности, а сжатие промежуточных кадров за счёт уменьшения временно́й избыточности данных. Кадрами видео являются изображения, поэтому задача сжатия отдельного (ключевого) кадра сводится к задаче сжатия изображения.

Все изображения разделяют на несколько классов [42].

1. Монохроматическое (двухуровневое) изображение. В этом случае все пиксели могут иметь только два значения.

2. Полутоновое изображение (изображение в «градациях серого»). Каждый пиксель такого изображения может иметь 2^n значений, где n – целое положительное число.

3. Цветное изображение с непрерывным тоном. Этот тип изображений имеет много сходных цветов, причём цвета сменяются плавно. Например, изображения с непрерывным тоном получаются при съёмке на цифровую фотокамеру или при сканировании обычных фотографий.

4. Цветные изображения, в которых присутствуют большие области одного цвета, а соприкасающиеся области могут значительно отличаться по своему цвету. К таким изображениям относятся кадры многих мультфильмов.

5. Цветное дискретно-тоновое (синтетическое) изображение. Этот тип изображений характеризуется резкими цветовыми переходами, количество различных цветов может варьироваться в широком диапазоне. Обычно это изображение получается искусственным путём. Примерами таких изображений могут служить фотографии искусственных объектов, страницы текста, карты, рисунки. Искусственные объекты, тексты, нарисованные линии имеют чёткую форму, хорошо определяемые границы. Они сильно контрастируют на фоне остальной части изображения (фона).

Большинство алгоритмов сжатия предназначено для обработки непрерывно-тоновых изображений. Такие алгоритмы эксплуатируют свойство непрерывно-тоновых изображений, которое и определило название этого класса изображений – наличие плавных цветовых переходов [42]. При этом устраняются малозаметные элементы, что сопровождается потерей информации. Дискретно-тоновые изображения в значительной мере отличаются от непрерывно-тоновых. При сжатии дискретно-тоновых изображений даже небольшой процент потерь может привести к значительному неприемлемому визуальному ухудшению их качества. Например, искажение всего нескольких пикселей буквы делает её неразборчивой, преобразует привычное начертание в практически неразличимое [42, с. 120].

Поэтому применение алгоритмов, предназначенных для сжатия непрерывно-тоновых изображений, при обработке дискретно-тоновых изображений нецелесообразно. Этот же вывод распространяется на кадры GUI-видео, так как они в большинстве своём также относятся к классу дискретно-тоновых изображений. В то же время в кадрах GUI-видео могут присутствовать незначительные, чаще всего статичные, то есть не изменяющиеся от кадра к кадру, элементы непрерывно-тоновой графики.

Рассмотрим отличия GUI-видео от традиционного видео на уровне последовательности кадров. Для изложения сути таких отличий обратимся к особенностям функционирования современных алгоритмов сжатия видеоданных, таких как алгоритм оценки движения. В процессе сжатия кадр, представленный в виде матрицы пикселей, условно разделяется на блоки равного размера. Затем для каждого блока выполняется анализ *векторов движения* – смещений по оси x и оси y относительно предыдущего кадра $\{x; y\}$ (рисунок 1.1), с целью нахождения вектора движения, обеспечивающего максимальную степень сходства соответствующих блоков по заданному критерию. Более подробно алгоритм оценки движения рассмотрен в разделе 1.3.

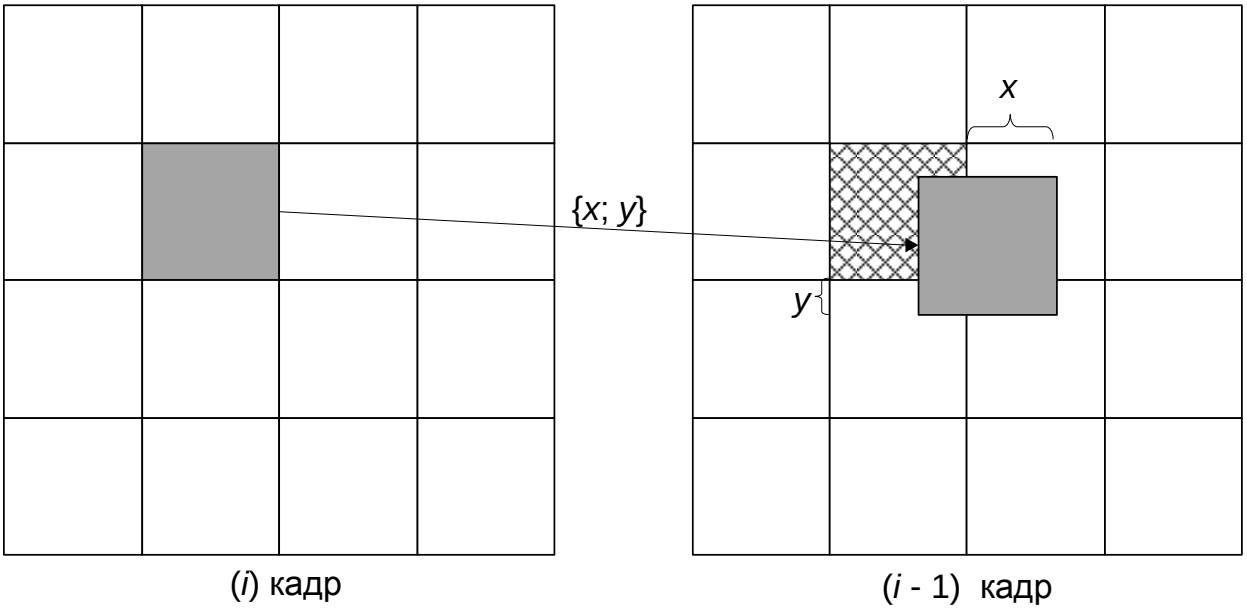


Рисунок 1.1 – Вектор движения блока в видеокадре

1. Для GUI-видео характерны низкая степень совпадения для большинства векторов движения и полное или почти полное совпадение для единственного вектора, в то время как для традиционного видео часто существуют несколько векторов движения, которые обеспечивают высокую степень сходства соответствующих блоков по заданному критерию.

2. В GUI-видео объекты могут перемещаться на значительное расстояние за временной отрезок, разделяющий два соседних кадра, в то время как в традиционном видео обычно происходит плавное движение объектов от кадра к кадру.

3. Для GUI-видео характерно совпадение значительной части блоков при нулевом векторе движения {0; 0}. В традиционном видео такой эффект наблюдается значительно реже за счёт изменения формы объектов, попадающих в кадр, и угла обзора при движении этих объектов.

4. В GUI-видео велика вероятность соседства нескольких блоков, имеющих один и тот же вектор движения (например, при движении окна), в отличие от традиционного видео, где одинаковый вектор движения для соседних блоков встречается существенно реже за счёт изменения формы объекта и угла обзора при движении этого объекта.

Всё это не учитывается в традиционных алгоритмах сжатия видеоданных. Эксперты международного комитета Collaborative Team on Video Coding (JCT-VC) также пришли к выводу о недостаточной вычислительной эффективности существующих алгоритмов сжатия, в частности HEVC (H.265), для обработки GUI-видеоданных [141]. Для устранения этого недостатка комитетом JCT-VC принято решение о создании расширения Screen Content coding (SCC) стандарта HEVC, которое на момент написания диссертационной работы находится в стадии разработки [96, 93, 94, 137, 141]. Стоит также отметить, что в программных документах SCC подчёркивается необходимость поддержки разрабатываемым кодеком режима обработки GUI-видеоданных без потерь информации [141].

Исходя из рассмотренных выше особенностей GUI-видеоданных, исследуемые алгоритмы сжатия должны быть вычислительно и ресурсоэффективными и производить сжатие без потерь информации. При оценке вычислительной эффективности алгоритмов сжатия GUI-видеоданных нужно учитывать, что в ходе формирования таких видеоданных, как правило, достаточно сохранять 10 кадров в секунду, например, именно такое значение используется по умолчанию в широко распространённом телекоммуникационном ПО Adobe Connect в режиме демонстрации собственного экрана компьютера с графическим интерфейсом собеседнику [51]. При этом желательно, чтобы обработка одного кадра занимала не более 1000 / 10 = 100 мс.

1.2. Сжатие ключевых кадров GUI-видео

1.2.1. Алгоритм группового кодирования

Идея алгоритма группового кодирования (RLE, Run Length Encoding) состоит в следующем. На каждом шаге алгоритма происходит подсчёт количества идущих подряд пикселей одного цвета, начиная с текущего пикселя. В

результирующий массив записываются байты пикселя, а также количество подряд идущих пикселей этого цвета [44].

Преимущества алгоритма:

- работает очень быстро, так как в процессе кодирования / декодирования не используются дополнительные структуры данных, сжатие осуществляется за один проход.

Недостатки алгоритма:

- если при построчном обходе пикселей часто чередуются цвета, степень сжатия резко снижается;
- способен выявлять только горизонтальную или вертикальную избыточность (в зависимости от способа обхода пикселей).

Первый недостаток алгоритма проявляется при сжатии GUI-видеоданных, так как частое чередование цветов характерно для кадров GUI-видео ввиду их дискретно-тоновой природы. Второй недостаток проявляется при сжатии любых видеоданных в силу двумерной природы их кадров.

CCITT Group 4. Отличие этой модификации от канонического алгоритма группового кодирования состоит в возможности выявления как горизонтальных, так и вертикальных групп эквивалентных пикселей. На каждом шаге происходит подсчёт количества идущих подряд эквивалентных пикселей вниз и вправо от текущего пикселя. Для кодирования выбирается группа, содержащая максимальное количество эквивалентных пикселей [128].

Эта модификация позволяет отчасти устранить второй недостаток канонического алгоритма группового кодирования, но не в полной мере, так как алгоритм по-прежнему оперирует одномерными объектами –горизонтальными и вертикальными линиями.

1.2.2. Словарные алгоритмы сжатия

Словарные алгоритмы выбирают некоторые последовательности символов и сохраняют их в словаре. Последовательности, встреченные повторно,

кодируются в виде меток, используя словарь. Наибольшую распространённость среди словарных алгоритмов получили алгоритмы семейства LZ (Lempel, Ziv). На данный момент созданы десятки алгоритмов семейства LZ, предназначенных для сжатия текстов, изображений и звука [4, 42].

Рассмотрим универсальный алгоритм семейства LZ, широко применяемый для сжатия GUI-видеоданных – LZO, затем алгоритм LZW, известный сравнительно высокой степенью сжатия и алгоритмы семейства LZ, адаптированные для сжатия изображений. Отдельно остановимся на наиболее распространённом формате словарного сжатия.

LZO (Lempel, Ziv, Oberhumer). Существует более десятка модификаций этого алгоритма, в значительной степени отличающихся по степени сжатия и по вычислительной эффективности кодирования и декодирования. Остановимся на модификации LZO999 [95], так как этот алгоритм активно применяется для сжатия кадров GUI-видео. Например, он используется в свободном ПО для сжатия GUI-видеоданных CamStudio [53].

Идея алгоритма заключается в поиске самого длинного совпадения между строкой буфера, начинающейся с текущего символа S_t и всеми фразами словаря. Эти фразы могут начинаться с любого символа, находящегося в словаре, и выходить за пределы словаря, вторгаясь в область буфера, но не могут выходить за границу окна. Длина совпадения не должна превышать размера буфера. Полученная в результате поиска фраза кодируется с помощью двух параметров: смещения от начала буфера и длины соответствия (совпадения).

C_{min}: 64 / 65.

C_{max}: 100.

Преимущества алгоритма:

- чрезвычайно быстрое декодирование;
- позволяет пользователю выбирать уровень сжатия (целые числа от 1 до 9). При этом при возрастании степени сжатия увеличивается и время, требуемое на кодирование данных.

Недостатки алгоритма:

- в ряде случаев степень сжатия GUI-видеоданных оказывается недостаточной, учитывая высокий уровень избыточности, характерный для таких видеоданных.

LZO может использоваться для быстрого сжатия не только изображений, но и других типов данных. Например, он применяется для организации адаптивной схемы сжатия данных, передаваемых по компьютерной сети [56].

LZW (Lempel, Ziv, Welch). Отличается от прочих алгоритмов семейства LZ предварительным занесением в словарь всех символов алфавита входной последовательности, что позволяет записывать в результирующий поток исключительно индексы фраз словаря. Из-за устранения необходимости регулярной передачи одного символа в явном виде LZW обеспечивает лучшее сжатие, чем многие алгоритмы семейства LZ [4].

Преимущества алгоритма:

- более высокая степень сжатия, чем у многих других алгоритмов семейства LZ.

Недостатки алгоритма:

- более низкая вычислительная эффективность по сравнению с LZO.

Рассмотрим основные словарные алгоритмы сжатия, адаптированные для обработки изображений.

Выбор метода линеаризации. Предпринимались попытки подобрать метод линеаризации изображения, максимизирующий степень сжатия алгоритмом из семейства LZ [105]. При таком подходе основная проблема заключается в том, что даже изображения одного класса существенно отличаются по набору и расположению запечатлённых объектов.

Недостатки алгоритма:

- Не получилось подобрать универсальный способ линеаризации, который бы позволял гарантированно увеличить степень сжатия по сравнению со стандартным обходом изображения по строкам алгоритмом из семейства LZ.

Использование двумерного словаря. Затем были разработаны модификации словарных алгоритмов, использующих двумерный словарь,

например, в [69] представлен алгоритм, основанный на алгоритме LZW [4], в котором в качестве словаря используется некоторая прямоугольная область изображения. В [54] представлен алгоритм сжатия изображений семейства LZ, также использующий прямоугольную область изображения в качестве словаря. Такие алгоритмы способны лучше выявлять пространственную избыточность за счёт использования двумерного словаря. Вычислительная задача при этом схожа с оценкой движения с той разницей, что при оценке движения ищется соответствие для блока фиксированного размера.

Преимущества алгоритма:

- обеспечивает более высокую степень сжатия изображений по сравнению с алгоритмом, использующим одномерный словарь, за счёт лучшего выявления пространственной избыточности.

Недостатки алгоритма:

- имеет более низкую вычислительную эффективность по сравнению с алгоритмом, использующим одномерный словарь, за счёт выполнения поиска в двумерном словаре.

Формат Deflate. Deflate – это формат словарного сжатия, определяющий алгоритм декодирования и не налагающий серьёзных ограничений на реализацию кодера. В принципе, в качестве алгоритма сжатия может применяться любой работающий со скользящим окном алгоритм, лишь бы он исходил из стандартной процедуры обновления словаря для алгоритма LZ77 и использовал задаваемые форматом типы кодов Хаффмана. Этот формат широко применяется при сжатии дискретно-тоновых изображений. Например, расширение «png» информирует о том, что файл сформирован в соответствии с форматом Deflate.

Преимущества формата:

- является универсальным, не ориентирован на конкретный тип данных и прост в практической реализации;
- де-факто является одним из промышленных стандартов на сжатие дискретно-тоновых изображений.

Одной из наиболее известных реализаций, соответствующих стандарту Deflate, является zlib [146]. Zlib используется в ПО для записи GUI-видео Camstudio при перекодировании данных.

1.2.3. Статистические алгоритмы сжатия

Чаще всего статистический алгоритм применяется на финальной стадии для увеличения суммарной степени сжатия. Наиболее известными алгоритмами статистического сжатия являются алгоритм Хаффмана и арифметическое кодирование.

Известно, что алгоритм Хаффмана производит идеальное сжатие (сжимает данные до их энтропии), если вероятности символов точно равны отрицательным степеням двойки [42]. Если же это условие не выполнено, то степень сжатия арифметическим кодированием будет выше, чем степень сжатия алгоритмом Хаффмана. В то же время арифметическое кодирование имеет меньшую вычислительную эффективность, чем алгоритм Хаффмана [122]. Поэтому при выборе статистического алгоритма следует исходить из природы сжимаемых данных, а также из ограничений на время выполнения алгоритма. Поскольку при сжатии GUI-видеоданных важны и вычислительная эффективность, и степень сжатия, рассмотрим оба алгоритма более подробно.

Алгоритм Хаффмана использует только частоту появления одинаковых байтов во входном блоке данных. В соответствие символам входного потока, которые встречаются чаще, ставится цепочка битов меньшей длины. Кодирование происходит в два этапа. На первом этапе строится дерево, в котором листьями являются обычные символы, а остальные вершины дерева представляют собой составные символы. На втором этапе кодирования каждому символу из входного блока данных сопоставляется последовательность бит в соответствии с расположением этого символа в построенном на первом этапе дереве.

C_{min}: 1.

C_{max}: 8.

Преимущества алгоритма:

- сравнительно высокая вычислительная эффективность;
- даже при минимальной степени сжатия не происходит увеличения размера исходных данных (если не учитывать необходимости хранить таблицу перекодировки вместе с закодированными данными).

Недостатки алгоритма:

- низкая C_{max}, поэтому этот алгоритм может применяться только в качестве одного из этапов сжатия GUI-видеоданных.

Арифметическое сжатие. Идея алгоритма заключается в представлении кодируемого текста в виде дроби. После задания начального интервала на каждом из шагов отрезок разбивается на подынтервалы с длинами, равными вероятностям появления символов в потоке. В качестве следующего рабочего интервала берётся диапазон, соответствующий текущему кодируемому символу. Его длина пропорциональна вероятности появления этого символа в потоке.

C_{min}: 1.

C_{max}: > 8 (возможно кодирование менее бита на символ).

Преимущества алгоритма:

- превосходство по степени сжатия над алгоритмом Хаффмана в общем случае (если вероятности символов не равны отрицательным степеням двойки [26]).

Недостатки алгоритма:

- более низкая вычислительная эффективность по сравнению с алгоритмом Хаффмана.

Таким образом, часть алгоритмов не подходит для сжатия GUI-видеоданных, так как осуществляют сжатие с потерями информации. Другая часть алгоритмов сжатия обладает недостаточной вычислительной эффективностью для сжатия ключевых кадров GUI-видео в фоновом режиме. Существуют алгоритмы, которые осуществляют сжатие без потерь информации и способны оперативно обрабатывать ключевые кадры. Но такие алгоритмы

обеспечивают недостаточный уровень сжатия, учитывая высокий уровень избыточности, характерный для GUI-видеоданных.

1.2.4. Алгоритмы классификации блоков изображения

Для увеличения степени сжатия изображения, а также для улучшения качества декодированного изображения на первом этапе сжатия применяется техника классификации блоков изображений. Информация о классе блока используется на дальнейших этапах сжатия, чтобы применить к конкретному блоку оптимальный алгоритм сжатия, обеспечивающий необходимую степень сжатия, вычислительную эффективность и допустимый уровень потерь информации.

Существуют различные классификации блоков изображения. В [118, 119] блоки изображения разделяют на непрерывно-тоновые и дискретно-тоновые. В [67, 68] блоки изображения разделяют на однотонные (сюда же относятся блоки с минимальными перепадами цвета пикселей), непрерывно-тоновые и структурные. В [65, 90] выделяют более трёх классов блоков.

Для сжатия изображений, являющихся кадрами GUI-видео, в [99] при классификации блоков изображения выделяют дискретно-тоновые и непрерывно-тоновые блоки. Делается логичный вывод о том, что дискретно-тоновые блоки (например, содержащие текст) должны сжиматься без потерь информации, а непрерывно-тоновые блоки (например, часть фонового рисунка) можно сжимать с потерями информации. Также предлагается проводить классификацию блоков изображения в качестве первого этапа сжатия ключевых кадров GUI-видео. Однако в этом исследовании не описан алгоритм такой классификации.

Таким образом, существующие алгоритмы классификации блоков изображения не адаптированы для обработки GUI-видеоданных.

1.3. Сжатие промежуточных кадров GUI-видео

При сжатии промежуточных кадров происходит уменьшение избыточности информации во временно́м измерении, основанное на том, что смежные видеокадры с высокой вероятностью отличаются незначительно.

1.3.1. Алгоритм отсечения неизменившихся блоков кадра

Простейший способ учитывать подобие соседних кадров – это вычислять разницу между значениями блока текущего кадра и соответствующего блока предыдущего кадра в соответствии с некоторой метрикой. Если разница оказалась ниже заданного предела, из блока текущего кадра попиксельно вычитается соответствующий блок предыдущего кадра. При этом происходит повышение статистической избыточности в блоке, что может быть использовано на последующих стадиях сжатия. Если разница оказалась выше заданного предела, блок сжимается теми же алгоритмами, что и блоки ключевых кадров. Описанный алгоритм *отсечения неизменившихся блоков кадра* имеет трудоёмкость $O(n)$.

В силу дискретно-тоновой природы GUI-видеоданных, случай, когда блок изменился незначительно, встречается реже, чем для традиционных видеоданных. Поэтому для сжатия промежуточных кадров GUI-видео целесообразно применять алгоритм, который определяет, какие блоки изменились по сравнению с предыдущим кадром, и записывает в результирующий массив номера изменившихся блоков. Принцип работы этого алгоритма проиллюстрирован на рисунке 1.2. В дальнейшем при упоминании об алгоритме отсечения неизменившихся блоков, речь будет идти именно о нём. Такой, адаптированный для обработки дискретно-тоновых данных алгоритм обладает трудоёмкостью $O(n)$ и осуществляет сжатие без потерь информации.

Для повышения степени сжатия данных необходимо подобрать размер блока. При увеличении размера блока уменьшаются накладные расходы,

связанные с хранением номеров изменившихся блоков, но уменьшается и точность определения изменившейся области (и наоборот).

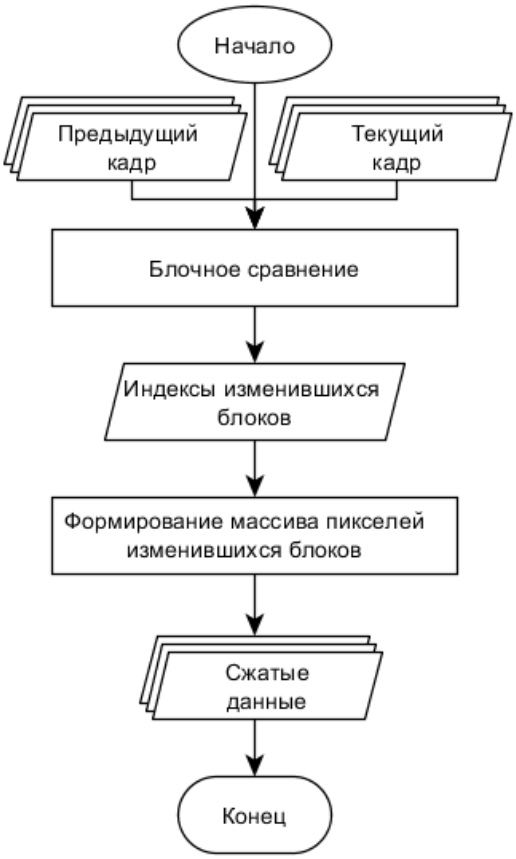


Рисунок 1.2 – Схема алгоритма отсечения неизменившихся блоков в кадре

Преимущества алгоритма:

- высокая вычислительная эффективность.

Недостатки алгоритма:

- низкая степень сжатия в случае перемещения объектов относительно предыдущего кадра.

1.3.2. Алгоритм оценки движения

Более гибким, но и более трудоёмким способом учёта корреляции соседних кадров является алгоритм *оценки движения* – алгоритм поиска векторов, на которые сместились блоки текущего кадра по отношению к предыдущему (рисунок 1.3).

Для восстановления кадра, закодированного алгоритмом оценки движения, используется алгоритм *компенсации движения*. В ходе оценки движения для каждого блока в изображении выполняется поиск блока, близкого по некоторой метрике (например, по сумме квадратов разности пикселей), в предыдущем кадре в некоторой окрестности текущего положения блока. Если минимальное расстояние по выбранной метрике с блоками в предыдущем кадре больше выбранного порога, блок сжимается независимо. Если для текущего блока найдено соответствие при векторе движения $\{x_1; y_1\}$, то этот вектор движения принимается в качестве начального для блоков, соседствующих с текущим. Таким образом происходит предсказание вероятного вектора движения [5]. Вместе с каждым блоком теперь сохраняются координаты смещения максимально похожего блока в предыдущем кадре относительно положения рассматриваемого блока в текущем кадре либо признак того, что данные сжаты независимо.

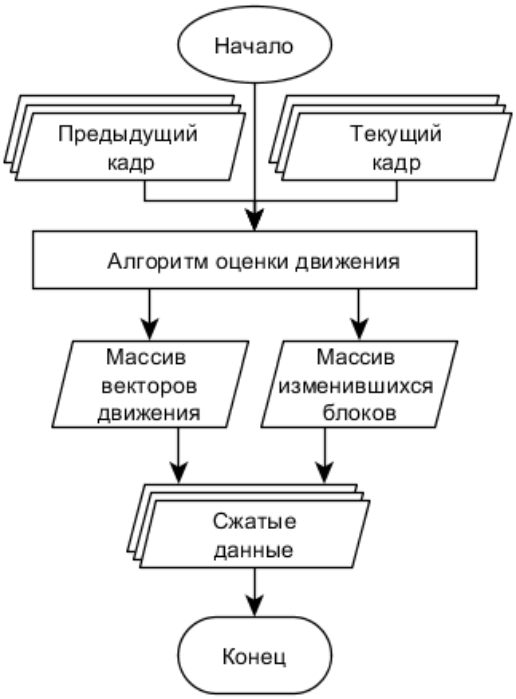


Рисунок 1.3 – Схема алгоритма оценки движения

В связи с вышеизложенным, алгоритм оценки движения позволяет существенно повысить степень сжатия видеоданных в случае, когда видеоданные содержат движущиеся объекты. Разумно использовать информацию об особенностях GUI-видео, чтобы адаптировать алгоритм оценки движения для

более эффективного выявления объектов, перемещённых на некоторое расстояние относительно предыдущего кадра.

В случае традиционных видеоданных существенная часть изменений между смежными кадрами может быть вызвана изменением положения камеры относительно сцены (удаление, приближение, а также движение и поворот в параллельной плоскости). Поэтому при обработке таких видеоданных имеет смысл проводить не только оценку движения отдельных блоков, но и *глобальную* оценку движения [111], в ходе которой могут применяться более сложные методы из области фотограмметрии [40, 77, 87]. В случае GUI-видеоданных указанные изменения между смежными кадрами являются маловероятными, поэтому глобальная оценка движения, как правило, не выполняется.

При обработке традиционных видеоданных алгоритмом оценки движения в случае нахождения вектора движения, обеспечивающего частичное совпадение блоков, для отличающихся пикселей текущего кадра вычисляется разность их цветов с цветами соответствующих пикселей предыдущего кадра [4, с. 346]. Для обеспечения эффективного сжатия на дальнейших этапах кодирования используется тот факт, что эти разности чаще всего близки к нулю (в силу непрерывно-тоновой природы таких видеоданных). Например, когда плавно происходят следующие изменения на экране: меняется освещение объекта, объект поворачивается в каком-то направлении, объект приближается или отдаляется – то блоки текущего и предыдущего кадров, содержащие этот объект, не будут идентичны, но соответствующие пиксели этих блоков будут иметь близкие значения. Стоит отметить, что для учёта изменения уровня освещённости запечатлённых объектов при обработке традиционных видеоданных может выполняться не только оценка движения, но и оценка оптического потока [111].

В случае GUI-видеоданных упомянутое предположение относительно разностей цветов отличающихся пикселей неверно, так как для GUI-видео нетипичны перечисленные выше изменения между кадрами. Так в отличающихся частях сравниваемых блоков разности цветов соответствующих пикселей могут принимать самые разные значения (в силу дискретно-тоновой природы таких

видеоданных). Поэтому при обработке GUI-видеоданных имеет смысл сжимать блоки, для которых не найдено точного соответствия, независимо от других кадров.

В [99] представлен алгоритм оценки движения, адаптированный для обработки GUI-видеоданных. Отличия этого алгоритма от аналогов связаны с отличиями GUI-видео от традиционного видео, проанализированными в разделе 1.1 (отличия 2 и 3). Рассмотрим особенности этого алгоритма оценки движения более подробно.

1. В алгоритме используется более обширная область поиска, вплоть до поиска по всему кадру, чем при сжатии традиционных видеоданных. Так, ограничение интервала поиска при сжатии традиционных видеоданных вводится в стандартах сжатия Mpeg-4, H.264 и других [41, 46, 111, 144]. В случае поиска по всему кадру трудоёмкость алгоритма $O(n^2)$, где n – количество пикселей в изображении.

2. В алгоритме предлагается сначала провести блочное сравнение текущего и предыдущего кадров на равенство. Затем оценка движения проводится только для блоков текущего кадра, изменившихся относительно предыдущего кадра.

3. В алгоритме не используется градиентный метод поиска оптимального вектора движения, активно применяемый при сжатии традиционных видеоданных [2, 89, 132, 136]. В случае GUI-видеоданных в силу их дискретно-тоновой природы, выбор из нескольких ближайших векторов движения в приоритетном порядке в качестве текущего вектора $v1$, который обеспечивает минимальное расхождение (если оно не равно 0), зачастую неэффективен. Это обусловлено тем обстоятельством, что вероятность того, что вектор движения, обеспечивающий наилучшее соответствие блоков, находится в непосредственной близости от вектора $v1$ не выше, чем при проверке любого другого вектора движения.

Преимущества алгоритма:

- обеспечивает высокую степень сжатия данных по сравнению с алгоритмом отсечения неизменившихся блоков.

Недостатки алгоритма:

- не является вычислительно эффективным, что препятствует его использованию для оперативного сжатия GUI-видеоданных.

Алгоритм компенсации движения, представленный в [99], имеет линейную трудоёмкость и является вычислительно эффективным, поэтому не нуждается в оптимизации.

1.4. Перспективные методы повышения эффективности сжатия GUI-видеоданных

Существуют не только алгоритмические, но и технологические способы повышения эффективности обработки данных. Рассмотрим перспективные методы повышения эффективности сжатия GUI-видеоданных предполагающие использование таких технологий.

1.4.1. Использование вычислительных ресурсов видеокарты

Производительность современных дискретных видеокарт, которыми оснащена значительная часть ПК, измеряется сотнями гигафлопов, в то время как производительность центрального процессора (ЦП) – десятками гигафлопов. ЦП не участвует в выполнении программ на видеокарте, поэтому его ресурсы высвобождаются для выполнения других задач, что открывает возможности для сжатия GUI-видеоданных в фоновом режиме. Направление информатики, посвящённое способам использования вычислительных ресурсов видеокарты для решения задач, не связанных напрямую с визуализацией, называется *GPGPU* (General-Purpose computation on Graphics Processing Units – универсальные вычисления на видеокарте). GPGPU позволяет перенести большой объём вычислений с ЦП на видеокарту.

Рассмотрим технологии, применяемые в GPGPU. Исторически первой технологией, позволяющей отойти от модели жёстко зафиксированного *графического конвейера* (аппаратно-программного комплекса визуализации

трёхмерной графики), были *шейдеры* (программы, предназначенные для исполнения процессорами видеокарты) [31]. *Пиксельные шейдеры* больше других (вершинных и геометрических) подходят для осуществления общих вычислений на видеокарте, так как позволяют получить программный доступ к отдельным пикселям. Но пиксельные шейдеры проектировались для визуализации изображений, что значительно сужает их область применения для общих вычислений [31, 120, 121]. Затем были разработаны более гибкие технологии, позволяющие перенести на видеокарту обширный спектр задач. В их числе Nvidia CUDA (Compute Unified Device Architecture), ATI CTM (Close To Metal) и стандарт OpenCL (Open Computing Language).

Архитектура видеокарты такова, что подавляющая часть транзисторов сосредоточена на ALU (arithmetic logic unit – арифметико-логическое устройство), и лишь незначительная часть транзисторов обеспечивает кэширование и выполнение инструкций потока управления. Таким образом, архитектура видеокарты рассчитана на интенсивные арифметические и логические вычисления, но плохо приспособлена для алгоритмических задач с большим количеством условных переходов.

Вот неполный список сфер, в которых находят своё применение вычисления на видеокарте: вычислительная гидрогазодинамика, автоматизация проектирования электроники, финансы, физика для игр, графика, вывод изображений, визуализация в медицине, численные методы, бионауки, обработка сигналов, обработка видео- и аудио-данных [50, 59, 79, 91, 123]. Использование вычислительных ресурсов видеокарты позволяет существенно повысить вычислительную эффективность многих реализаций алгоритмов обработки изображений [59, 79¸ 83, 117].

Активно исследуется применение видеокарты для сжатия данных. В [91] описывается метод сжатия больших изображений, полученных лазерным сканированием трёхмерных объектов. Декодирование осуществляется с помощью вершинных шейдеров. Авторы работ [123] и [110] достигли значительных успехов в декодировании видеоданных высокого разрешения с помощью видеокарты. Им

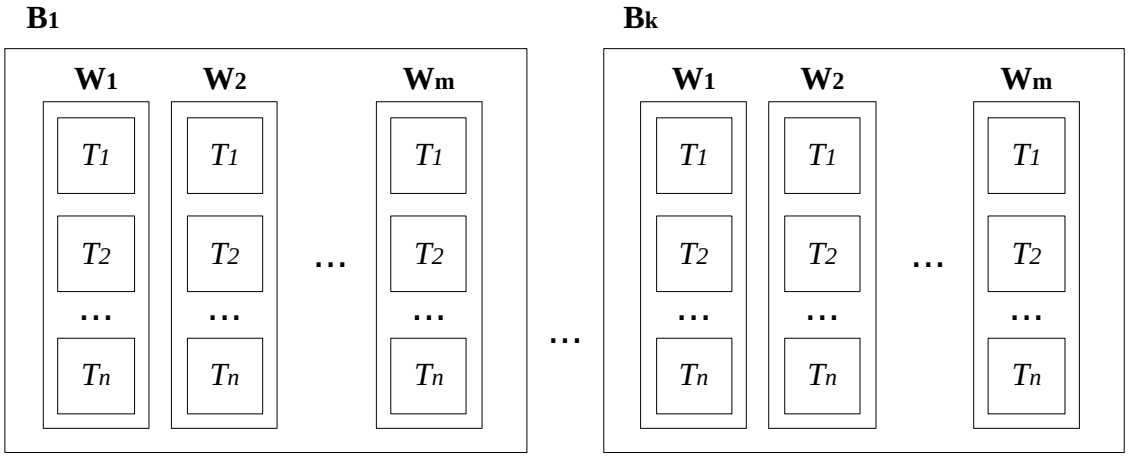
удалось организовать конвейер, где первые стадии обработки выполняются ЦП, а последующие – пиксельными и вершинными шейдерами. Авторы [135] реализуют значительную часть декодирования кодека Дирака [63] с помощью технологии Nvidia CUDA, добиваясь при этом многократного ускорения в работе декодера по сравнению с ЦП-реализацией. В [73] представлен H.264 декодер, реализованный с помощью технологии CUDA, позволяющий в несколько раз ускорить декодирование видеоданных по сравнению с ЦП-реализацией.

Существует ряд работ, посвящённых реализации алгоритма оценки движения с использованием ресурсов видеокарты. В [126] описывается реализация алгоритма оценки движения с помощью пиксельных шейдеров, приводятся результаты тестирования, демонстрирующие её ускорение относительно ЦП-реализации. В [136] описано сканирование трёхмерных объектов, которое включает в себя оценку движения, реализованную с помощью вершинных и пиксельных шейдеров. Авторы [62] применили алгоритм оценки движения, реализованный с помощью вершинных и пиксельных шейдеров, для цифровой рентгенографии кровеносных сосудов. Авторы [107] используют алгоритм оценки движения, реализованный с помощью пиксельных шейдеров, как этап при сжатии видеоданных, получаемых одновременно с нескольких камер с разных позиций. Но ни в одной из этих работ не ставилась задача выполнения оценки движения в оперативном режиме, что является необходимым условием применения алгоритма оценки движения для сжатия GUI-видеоданных (раздел 1.1).

Таким образом, возможности современных видеокарт успешно применяются для создания более быстрых реализаций алгоритмов по сравнению с ЦП-реализациями, для уменьшения нагрузки на ЦП при решении разнообразных задач. Но работ, посвящённых обработке GUI-видеоданных с помощью видеокарты, найти не удалось. Видится логичным проанализировать возможность и целесообразность использования вычислительных ресурсов видеокарты при сжатии GUI-видеоданных.

На момент написания данной работы Nvidia CUDA, пожалуй, является наиболее распространённой технологией GPGPU. Рассмотрим эту технологию более подробно.

Технология Nvidia CUDA позволяет выполнять программы, написанные на си-подобном языке на процессоре видеокарты без привлечения центрального процессора. Эта технология позволяет эффективно организовать параллельную обработку данных для решения задач, связанных с математическими вычислениями. Потоки объединены в блоки, а каждый блок содержит равное количество потоков (рисунок 1.4).



B – блок, **W** – warp, *T* – поток

Рисунок 1.4 – Организация потоков в технологии Nvidia CUDA

При вызове функции, выполняемой непосредственно процессором видеокарты, эта функция выполняется тем количеством потоков, которое указано при её вызове. При этом часть работы, выполняемая некоторым потоком, определяется по номеру блока и номеру потока в этом блоке. Блок содержит несколько *warp'ов. Warp* – группа потоков, которые выполняются одновременно и для которых гарантировано, что они будут выполнять одну и ту же инструкцию в один и тот же такт времени.

Модель памяти технологии CUDA включает несколько типов памяти. В том числе это разделяемая внутри блока память и константная память. Разделяемая внутри блока память является внутрипроцессорной, поэтому скорость доступа к ней намного выше, чем к глобальной памяти. Каждому блоку потоков выделяется

определённое количество такой памяти. Константная память оптимизирована для доступа на чтение и доступна для потоков всех блоков. Этот тип памяти кэшируемый, что в среднем ускоряет чтение данных, находящихся в константной памяти. Наличие этих двух типов памяти позволяет ускорить выполнение алгоритмов, требующих большого количества обращений к памяти [109]. Программист сам определяет количество потоков, которые будут выполнять указанную функцию. При этом любой поток может обращаться на чтение и запись к любому байту обрабатываемых данных (отсутствует жёсткое разделение на входные и результирующие данные). Программные модули, написанные с использованием технологии Nvidia CUDA, легко встраиваются в обычное приложение, написанное на C или C++.

1.4.2. Технологии определения изменившихся частей кадра в GUI-видео

Информация об изменениях, произошедших на экране, очень полезна при сжатии GUI-видеоданных. Зная, какие области экрана не изменялись относительно предыдущего кадра, а какие получены копированием областей предыдущего кадра, можно не сравнивать текущий и предыдущий кадры, а сжимать только изменившиеся области текущего кадра, которые не были получены копированием областей предыдущего кадра. Использование таких технологий может выступать в качестве альтернативы алгоритмам устранения временно́й избыточности GUI-видеоданных либо дополнять их.

Одной из основных технологий, позволяющих определять изменившиеся относительно предыдущего кадра области текущего кадра в GUI-видео, является *Mirror Video Driver* (*зеркальный видеодрайвер*). Операционная система (ОС) дублирует все графические операции на зеркальный видеодрайвер. Эта технология поддерживается в ОС Windows, начиная с версии Windows 2000. Таким образом, приложение, использующее зеркальный драйвер, имеет возможность определять любые изменения экрана [29]. Эта технология применяется во многих приложениях, предназначенных для удалённого

управления компьютером: UltraVNC [134], TightVNC [131], Radmin [114] и RemotePC [115].

Однако существуют сложности в использовании зеркального видеодрайвера в Windows Vista и Windows 7. Основная проблема заключается в том, что зеркальный видеодрайвер основан на модели драйвера устройства отображения Windows XP, поэтому при использовании зеркального видеодрайвера в Windows Vista две модели драйвера устройства отображения, соответствующие Windows XP и Windows Vista, должны функционировать одновременно [52]. Как следствие, при использовании зеркального видеодрайвера Windows Vista выходит из режима Windows Aero, являющегося одним из наиболее заметных нововведений, которые появились в Windows Vista/7 [139]. Так происходит потому, что ядром технологии Windows Aero является Desktop Window Manager, который активно использует модель драйвера устройства отображения Windows Vista. Для пользователей ПО, обеспечивающего фиксацию GUI-видео, может оказаться важным, чтобы графический пользовательский интерфейс созданного ими приложения отображался корректно при записи GUI-видео. Таким образом, зеркальный видеодрайвер может полноценно функционировать только на некоторых версиях ОС Windows, в то время как в ОС семейства Linux нет поддержки этой технологии, а на Windows Vista/7 такое функционирование затруднительно.

Другой распространённой технологией, применяемой для определения изменившихся частей кадра в GUI-видео, являются перехватчики событий (hooks). *Перехватчик события* – это механизм, с помощью которого приложение может перехватывать события, такие как посылка сообщений, активность мыши, нажатие клавиш мыши [81]. Эта технология, также как и зеркальный видеодрайвер, поддерживается в ОС Windows. При сжатии GUI-видеоданных имеет смысл использовать перехватчики событий для перехвата событий, связанных с выводом на экран данных различными приложениями. Эта технология применяется в ПО TightVNC, предназначенном для удалённого управления компьютером.

Но перехватчики событий специфичны для каждой ОС, поэтому для создания ПО, использующего перехватчики событий, для одной ОС на основе реализации для другой ОС могут потребоваться значительные усилия. Использование перехватчиков событий также не гарантирует, что удастся получить точную копию картинки, отображаемой на экране. Например, известно, что для стандартных заголовков окон и стандартных меню не удаётся получить точную информацию о том, как они выглядят с помощью перехватчиков событий [52]. Понятно, что список отображаемых элементов, о которых не удаётся получить точной информации, будет варьироваться в зависимости от версии Windows.

Таким образом, в обеих рассмотренных технологиях, позволяющих определять изменившиеся части кадра в GUI-видео, присутствует ряд серьёзных ограничений. Поэтому применение таких технологий не может в полной мере заменить алгоритмы сжатия, устраняющие временну́ю избыточность в GUI-видеоданных.

1.5. Выводы о направлениях работы

На основании вышеизложенных результатов анализа состояния проблемы сжатия GUI-видеоданных можно сделать следующие выводы:

1. Фиксация и обработка GUI-видео находят применение у широкого круга пользователей ПК.

2. Выявлены отличия GUI-видеоданных от традиционных видеоданных как на уровне отдельных кадров, так и на уровне последовательности кадров. Показано наличие сложностей, сопутствующих фиксации и обработке GUI-видеоданных. В их числе сравнительно значительный объём таких видеоданных, необходимость осуществлять их сжатие оперативно в фоновом режиме.

3. Сформулированы требования к алгоритмам сжатия GUI-видеоданных. Такие алгоритмы должны обеспечивать высокую степень сжатия без потерь информации и быть вычислительно и ресурсоэффективными.

4. Проведён анализ существующих алгоритмов, применяемых для сжатия ключевых кадров GUI-видео и устраняющих пространственную избыточность. Показано, что статистические, словарные алгоритмы и алгоритм группового кодирования в ряде случаев сжимают GUI-видеоданные с низкой степенью сжатия.

5. Проведён анализ существующих алгоритмов, применяемых для сжатия промежуточных кадров GUI-видео и устраняющих временну́ю избыточность. Показано, что алгоритм отсечения неизменившихся блоков в ряде случаев сжимает GUI-видеоданные с низкой степенью сжатия, а алгоритм оценки движения, адаптированный для обработки GUI-видеоданных, не является вычислительно эффективным.

6. Проведён анализ технологий повышения эффективности сжатия GUI-видеоданных. Использование ресурсов видеокарты для обработки GUI-видеоданных отмечено в качестве наиболее перспективной из таких технологий. Показано, что использование технологий, позволяющих определять изменившиеся части кадра в GUI-видео, сопряжено с существенными ограничениями и поэтому не является полноценной альтернативой алгоритмам сжатия, устраняющим временну́ю избыточность.

С учётом вышеизложенных выводов, целью диссертационной работы является разработка алгоритмического и программного обеспечения фиксации и локального сохранения GUI-видеоданных со сжатием без потерь с высокой степенью сжатия, обладающего высокими показателями ресурсо- и вычислительной эффективности.

Для достижения поставленной цели необходимо последовательное решение следующих задач:

1. Анализ существующего алгоритмического и программного обеспечения сжатия видеоданных, особенностей традиционного видео и GUI-видео, позволяющих определить основные направления исследования в области построения эффективных алгоритмов сжатия GUI-видеоданных.

2. Разработка вычислительно и ресурсоэффективных алгоритмов сжатия без потерь информации, обеспечивающих высокую степень сжатия GUI-видеоданных. Решение данной задачи предполагает также исследование эффективности предлагаемых алгоритмов.

3. Создание высокопроизводительного ядра кодека, предназначенного для сжатия и восстановления GUI-видеоданных. Результатом решения этой задачи должны явиться программные средства такого кодека, реализующие предложенные алгоритмы.

4. Апробация разработанного кодека с использованием различных видеофрагментов при решении экспериментальных и реальных задач.

Цель исследования может быть представлена в форме задачи условной оптимизации с неявными ограничениями в следующем виде:

$$\max_{x} E_c(x) \qquad (1.1)$$

при ограничениях (условиях):

$$\begin{cases} C_{comp}(x) \geq C_{min} \\ M_{avr}(x) \leq M_{max} \end{cases},$$

где x – алгоритм сжатия без потерь информации, E_c – вычислительная эффективность, C_{comp} – степень сжатия данных, M_{avr} – средний уровень использования ОП, C_{min} – минимальное практически допустимое значение критерия C_{comp} и M_{max} – максимальное практически допустимое значение критерия M_{avr}. Максимизация вычислительной эффективности алгоритма, его реализации и ПО сжатия GUI-видеоданных выражается в минимизации трудоёмкости, времени выполнения t и степени загруженности ЦП C_{cpu}, соответственно.

ГЛАВА 2. АЛГОРИТМИЧЕСКОЕ ОБЕСПЕЧЕНИЕ СЖАТИЯ GUI-ВИДЕОДАННЫХ

Одной из задач, требующих решения для достижения целей диссертационной работы, является задача разработки алгоритмического обеспечения сжатия GUI-видеоданных. Данная глава посвящена описанию решения задачи сжатия ключевых и промежуточных кадров GUI-видео, включает результаты исследования предложенных алгоритмов и результаты сравнительного анализа с аналогами.

2.1. Сжатие ключевых кадров GUI-видео

Для изложения сути алгоритмов сжатия ключевых кадров GUI-видео необходимо ввести следующие определения. Здесь и далее под *остаточным изображением* будем понимать массив пикселей исходного изображения, которые не были заменены на служебные данные в ходе кодирования алгоритмом сжатия. В случае, когда алгоритм сжатия заменяет все пиксели исходного изображения служебными данными, остаточное изображение имеет нулевой размер. Определим функцию $D(I_x, I_y)$, задающую *расстояние* между элементами в векторе **B**:

$$D(I_x, I_y) = |x - y|,$$

где I_x, I_y – элементы вектора **B** с индексами x, y соответственно. В качестве элементов вектора **B** могут выступать пиксели или отдельные байты. В дальнейшем при упоминании расстояния речь будет идти именно о значении, возвращаемом функцией $D(I_x, I_y)$.

Для достижения высокой степени сжатия ключевых кадров GUI-видео необходимо устранить типы избыточности, связанные:

1. с частым чередованием цветов;
2. с наличием групп одноцветных пикселей.

Для устранения первого типа избыточности разработан *сдвиговый алгоритм* (п. 2.1.1), а для устранения второго типа избыточности разработан *алгоритм пространственного группового кодирования* (п. 2.1.2), учитывающий горизонтальную и вертикальную корреляцию пикселей кадра. Эти два алгоритма логически объединены в *гибридном сдвигово-групповом алгоритме* (п. 2.1.3) и рассмотрены в данном разделе.

На финальной стадии сжатия требуется устранить избыточность, присутствующую в результирующих наборах данных гибридного сдвигово-группового алгоритма. Это в первую очередь градиентные переходы в остаточном изображении, которые не могут быть устранены гибридным алгоритмом. Алгоритм, предполагающий обработку данных гибридным сдвигово-групповым алгоритмом и финальное сжатие его результирующих наборов данных известными алгоритмами, назовём *алгоритмом со сниженной пространственной избыточностью* (*АСПИ*) (п. 2.1.4).

2.1.1. Сдвиговый алгоритм

Сдвиговый алгоритм обрабатывает изображение, представленное в виде вектора. Вектор **B** формируется путём последовательной записи в него строк исходной матрицы пикселей $\mathbf{P}_{\text{исх}}$. На рисунке 2.1 схематично описаны действия, выполняемые на каждом шаге сдвигового алгоритма.

Идея сдвигового алгоритма заключается в следующем. Если до текущего пикселя P_{cur} в векторе **B** встречался пиксель эквивалентного цвета P_{prev}, то байты, кодирующие цвет пикселя P_{cur}, заменяются адресной ссылкой на пиксель P_{prev} (*базовый пиксель*) при условии:

$$\exists\, P_{prev}\,\big(P_{prev}= P_{cur}\big)\cap \\ (pos\big(P_{prev}\big)<pos(P_{cur}))\cap(D(P_{prev}, P_{cur})\leq S_{max}), \tag{2.1}$$

где функция *pos* возвращает позицию пикселя в векторе **B**, S_{max} – максимальная величина адресной ссылки, функция $D(P_{prev}, P_{cur})$ может быть задана одним из двух способов: 1) функция $D(P_{prev}, P_{cur})$ возвращает количество пикселей между

P_{prev} и P_{cur} в остаточном изображении, представленном в виде вектора; 2) функция $D(P_{prev}, P_{cur})$ возвращает количество байтов между P_{prev} и P_{cur} в результирующем массиве **R**. Если условие 2.1 не выполнено, в **R** записываются байты, кодирующие цвет пикселя. Впервые алгоритм представлен в [8].

Рисунок 2.1 – Принцип работы сдвигового алгоритма

Алгоритм относится к группе словарных алгоритмов. Сдвиговый алгоритм, как и прочие алгоритмы сжатия, представленные в этой главе, может применяться к исходным данным, представленным в различных цветовых моделях (RGB, YUV, CMYK и т. д.) и с различным количеством битов, отводимых для хранения пикселя. Необходимое условие сжатия можно представить как

$$s < p,$$

где s – количество битов, отводимых для хранения адресной ссылки, p – количество битов, отводимых для хранения пикселя.

Минимальная и максимальная степени сжатия данных разнятся в зависимости от реализации. Максимальная степень сжатия подчиняется следующей закономерности:

$$C_{max} \leq p / s$$

Трудоёмкость алгоритма $O(n)$.

Как указано в разделе 1.1, для GUI-видеоданных свойственны резкие (дискретно-тоновые) цветовые переходы между соседними пикселями. Одним из следствий этого факта является ограниченное количество цветов в одном кадре,

что повышает вероятность замены пикселя на адресную ссылку сдвиговым алгоритмом и уменьшает требуемый объём памяти.

Преимущества алгоритма:

- способен устранить избыточность, связанную с частым чередованием цветов, свойственную именно GUI-видеоданным;
- может быть с высокой эффективностью скомбинирован с алгоритмами группового кодирования за счёт имеющихся в словаре адресных ссылок на отдельные пиксели.

Вместе с тем алгоритм обладает существенными недостатками:

- не способен эффективно кодировать одноцветные группы пикселей;
- C_{max} невелика;
- невысокая вычислительная эффективность из-за необходимости определения позиции базового пикселя на каждом шаге алгоритма.

Таким образом, для обеспечения эффективности сдвигового алгоритма и раскрытия его потенциала он должен использоваться в комбинации с алгоритмом группового кодирования.

2.1.2. Алгоритм пространственного группового кодирования

Канонический алгоритм группового кодирования обладает ограниченной способностью к выявлению пространственной избыточности (п. 1.2.1). Чтобы устранить этот недостаток, разработан алгоритм пространственного группового кодирования.

Суть алгоритма пространственного группового кодирования заключается в следующем. На каждом шаге алгоритма происходит выявление пространственных объектов, начинающихся с текущего пикселя и состоящих из эквивалентных пикселей. Для кодирования выбирается объект, содержащий максимальное количество таких пикселей. В результирующий массив записывается тип этого объекта, количество пикселей в нём и байты, кодирующие цвет пикселя. Предлагается включить в набор выявляемых пространственных объектов

горизонтальную, вертикальную линии и прямоугольник (рисунок 2.2). На каждом шаге подсчёт эквивалентных пикселей производится по горизонтали вправо от текущего пикселя, по вертикали вниз от текущего пикселя, а также в прямоугольнике, левым верхним углом которого является текущий пиксель. Выявление прямоугольных областей одноцветных пикселей оправдано ввиду наличия в GUI-видеоданных сравнительно большого количества таких областей.

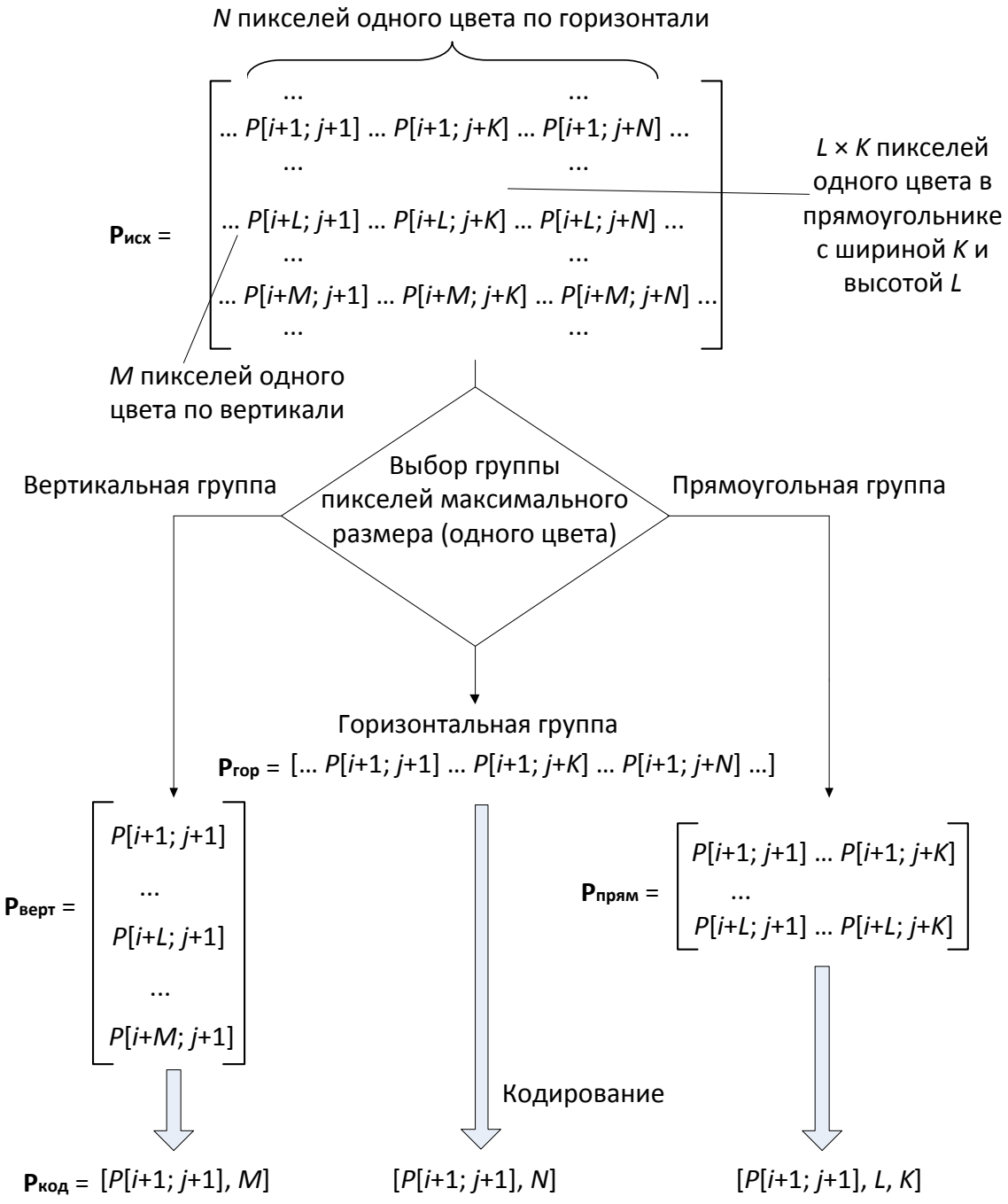


Рисунок 2.2 – Принцип работы алгоритма пространственного группового кодирования

Выявление прямоугольных групп способно значительно увеличить степень сжатия GUI-видеоданных, так как одна такая группа размером $L \times K$ заменяет $min(L, K)$ групп других типов (горизонтальных или вертикальных).

Рассмотрим минимальную и максимальную степени сжатия данных алгоритма.

$$C_{min} = p / (p + l_{min}),$$

$$C_{max} = n_{max} \times p / (p + l_{max}),$$

где n_{max} – максимальное количество пикселей в группе, p – количество битов, отводимых для хранения пикселя, l_{max} – количество битов, отводимых под служебные данные в случае кодирования группы пикселей максимального размера, l_{min} – количество битов, отводимых под служебные данные в случае кодирования одиночного пикселя. Трудоёмкость данного алгоритма $O(n)$.

Преимущество алгоритма:

- повышенная способность выявлять пространственную избыточность по сравнению с каноническим алгоритмом.

Но при этом сохраняется главный недостаток канонического алгоритма группового кодирования:

- эффективность сжатия резко падает при частом чередовании цветов при прямом обходе пикселей.

Поэтому имеет смысл использовать алгоритм пространственного группового кодирования в составе гибридного сдвигово-группового алгоритма, в котором этот недостаток устранён за счёт применения сдвигового алгоритма. Рассмотрим гибридный сдвигово-групповой алгоритм подробнее.

2.1.3. Гибридный сдвигово-групповой алгоритм

На рисунке 2.3 схематично описаны действия, выполняемые на каждом шаге гибридного сдвигово-группового алгоритма. На этом рисунке используются те же условные обозначения, что и на рисунке 2.2.

Этот алгоритм построен на основе алгоритма пространственного группового кодирования и сдвигового алгоритма. На каждом шаге гибридного сдвигово-группового алгоритма кодирование выполняется две стадии:

Рисунок 2.3 – Принцип работы гибридного сдвигово-группового алгоритма

1. Поиск возможностей замены байтов пикселя на адресную ссылку.

2. Поиск возможностей группового кодирования группы одноцветных пикселей, включая текущий пиксель.

Гибридный сдвигово-групповой алгоритм, как и сдвиговый алгоритм, обрабатывает изображение, представленное в виде вектора. Гибридному сдвигово-групповому алгоритму посвящены работы [8, 9, 10, 17, 18, 19, 20].

Максимальная и минимальная степени сжатия варьируются в зависимости от выбранной реализации (раздел 3.1). Трудоёмкость алгоритма $O(n)$.

Гибридный сдвигово-групповой алгоритм обладает всеми преимуществами алгоритма пространственного группового кодирования. При использовании гибридного алгоритма достигается дополнительное сжатие за счёт замены байтов чередующихся пикселей на адресные ссылки, что устраняет один из недостатков алгоритма группового кодирования. При этом гибридный сдвигово-групповой алгоритм практически не наследует недостатки сдвигового алгоритма. Так, недостатки (1) и (2) устраняются за счёт сжатия алгоритмом пространственного группового кодирования, а недостаток (3) присутствует в значительно меньшей степени, так как обращение к дополнительным структурам данных происходит не для каждого пикселя, а для каждой группы.

Чтобы повысить эффективность финального сжатия, необходимо определить, какой тип избыточности характерен для каждого из результирующих наборов данных гибридного сдвигово-группового алгоритма. При обработке

изображения этим алгоритмом формируются три набора данных, описанных ниже.

1. Массив флагов.

2. Массив сдвигов и количеств.

3. Массив пикселей остаточного изображения.

В массив флагов записываются служебные данные, имеющие битовую природу. Избыточность в этом наборе данных минимальна, так как каждый байт содержит несколько элементов, и существенная часть байтов содержит дробное количество элементов за счёт записи части битов элемента, занимающего несколько битов.

В массиве сдвигов и количеств содержатся различные служебные данные (кроме флагов), используемые гибридным сдвигово-групповым алгоритмом. Все данные, попадающие в этот массив, имеют однобайтовую природу и характеризуются бóльшей избыточностью по сравнению с массивом флагов. Действительно, для кадра GUI-видео оказывается, что различные значения таких параметров, как количество подряд идущих пикселей одного цвета, а также расстояние до ближайшего встреченного пикселя эквивалентного цвета, не являются равновероятными. Поэтому такие данные имеют статистическую избыточность.

Как правило, статистическая избыточность в остаточном изображении гибридного сдвигово-группового алгоритма невысока (например, алгоритм Хаффмана, как правило, способен сжать остаточное изображение всего на 1–3 %). Вместе с тем в этих данных присутствует некоторая пространственная избыточность. Часто в остаточном изображении можно увидеть группы пикселей близких цветов, идущих подряд по вертикали или горизонтали (градиентные переходы). Такой тип пространственной избыточности характерен для остаточного изображения, так как гибридный сдвигово-групповой алгоритм не может его устранить.

2.1.4. Варианты алгоритма со сниженной пространственной избыточностью

Для устранения избыточности результирующих наборов данных гибридного сдвигово-группового алгоритма предложены два АСПИ, использующих гибридный сдвигово-групповой алгоритм на первом этапе (рисунок 2.4).

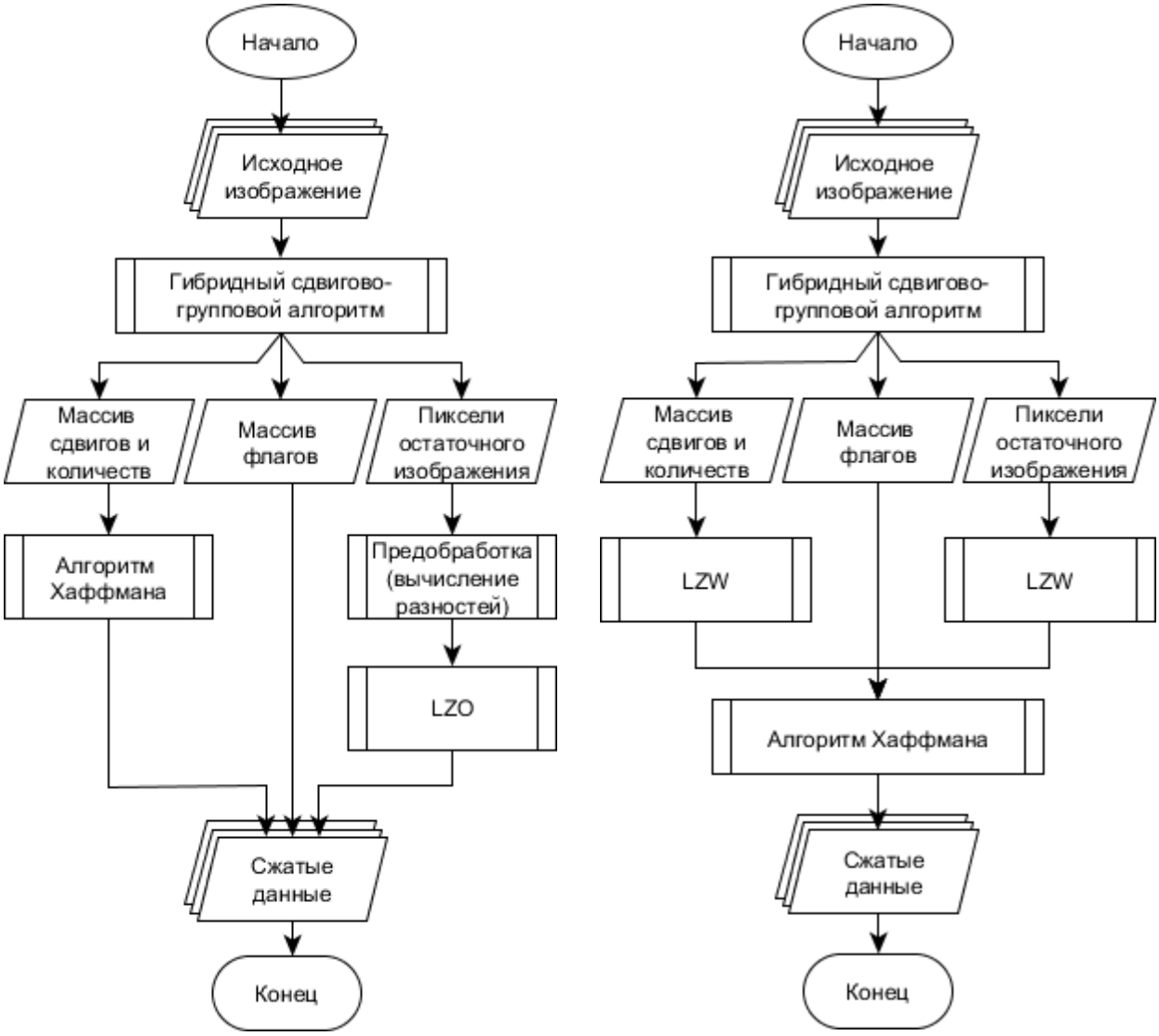


Рисунок 2.4 – Схема АСПИ, использующего: а) LZO и алгоритм Хаффмана; б) LZW и алгоритм Хаффмана

Первый из этих алгоритмов на втором этапе сжатия использует LZO и алгоритм Хаффмана. Это связано с тем, что LZO ввиду высокой вычислительной эффективности широко применяется для оперативного сжатия GUI-видеоданных, а алгоритм Хаффмана обладает сравнительно высокой вычислительной эффективностью среди статистических алгоритмов сжатия (раздел 1.2).

Второй АСПИ на втором этапе использует алгоритм LZW, так как LZW позволяет достичь более высокой степени сжатия по сравнению со многими прочими алгоритмами семейства LZ (п. 1.2.2).

Таким образом, первый АСПИ нацелен на достижение высокой вычислительной эффективности, в то время как второй – на достижение более высокой степени сжатия.

Рассмотрим АСПИ, осуществляющий на втором этапе сжатие алгоритмами LZO и Хаффмана. Результирующие наборы данных обрабатываются следующим образом.

1. Массив флагов записывается в результирующий поток без дополнительного сжатия.

2. Массив сдвигов и количеств сжимается алгоритмом Хаффмана, так как кодирование алгоритма Хаффмана обладает достаточно высокой вычислительной эффективностью для оперативного сжатия GUI-видеоданных.

3. Для массива пикселей **C** проводится предобработка, в ходе которой **C** преобразуется в массив **C_coded**. Новое значение *C_coded*[*i*] для каждой компоненты пикселя *C*[*i*], начиная со второго, вычисляется по формуле 2.2.

$$C\_coded[i] = (C[i] - C[i-1]) \ \% \ 2^k, \qquad (2.2)$$

где «%» – операция взятия остатка от деления, *k* – количество битов, отводимых для хранения одной компоненты цвета. k зависит от формата входных данных (RGB888, RGB565 и т. д.).

После этой предобработки вместо различных последовательностей пикселей близкого цвета появляются значения, близкие к нулю. И только первый пиксель в группе пикселей близкого цвета после такой предобработки получает значение, далёкое от нуля. Затем применяется алгоритм LZO, который обеспечивает дополнительное сжатие при минимальных временных затратах.

Перейдём к рассмотрению АСПИ, осуществляющего на втором этапе сжатие алгоритмами LZW и Хаффмана. В отличие от предыдущего АСПИ, массив сдвигов и количеств и остаточное изображение здесь обрабатывается алгоритмом LZW, так как он способен обеспечить сравнительно высокую степень

сжатия по сравнению с прочими словарными алгоритмами сжатия (п. 1.2.2). Затем все результирующие наборы данных гибридного сдвигово-группового алгоритма кодируются алгоритмом Хаффмана по отдельности, так как каждый из этих наборов данных обладает специфическим типом избыточности. Было принято решение отказаться от предварительного вычисления разностей в этом АСПИ, так как при использовании такой предобработки степень сжатия ухудшается, что установлено в ходе экспериментальных исследований (п. 3.2.1). Потенциально, LZW в АСПИ может быть заменён другим словарным алгоритмом сжатия с требуемым соотношением степени сжатия и вычислительной эффективности.

Оба АСПИ, рассмотренные в разделе, впервые представлены в [10] и [15].

2.2. Сжатие промежуточных кадров GUI-видео

Для достижения высокой степени сжатия промежуточных кадров GUI-видео необходимо устранить временну́ю избыточность, учитывая особенности GUI-видеоданных. Для этого разработано несколько алгоритмов сжатия (пп. 2.2.1-2.2.3). В п. 2.2.1 представлен *алгоритм отсечения неизменившихся строк и столбцов кадра*, повышающий степень сжатия данных в отдельных, часто встречающихся в GUI-видеоданных случаях, по сравнению с аналогами. В п. 2.2.2 представлен *адаптивный алгоритм*, сочетающий преимущества алгоритмов отсечения неизменившихся блоков (п. 1.3.1) и строк, столбцов. В п. 2.2.3 представлен *алгоритм оценки движения с учётом классификационных признаков*, обладающий более высокой вычислительной эффективностью по сравнению с аналогами.

Предлагается использовать адаптивный алгоритм отсечения неизменившихся областей на первой стадии сжатия промежуточных кадров GUI-видео, так как он обладает высокой вычислительной эффективностью и позволяет уменьшить объём входных данных для алгоритма оценки движения, который обеспечивает высокую степень сжатия, но имеет меньшую вычислительную

эффективность. Алгоритмы сжатия промежуточных кадров, рассмотренные в разделе, также представлены в [11, 13, 16].

2.2.1. Алгоритм отсечения неизменившихся строк и столбцов в кадре

В отличие от алгоритма отсечения неизменившихся блоков (п. 1.3.1), кадр не делится на блоки, а рассматривается как матрица пикселей (рисунок 2.5).

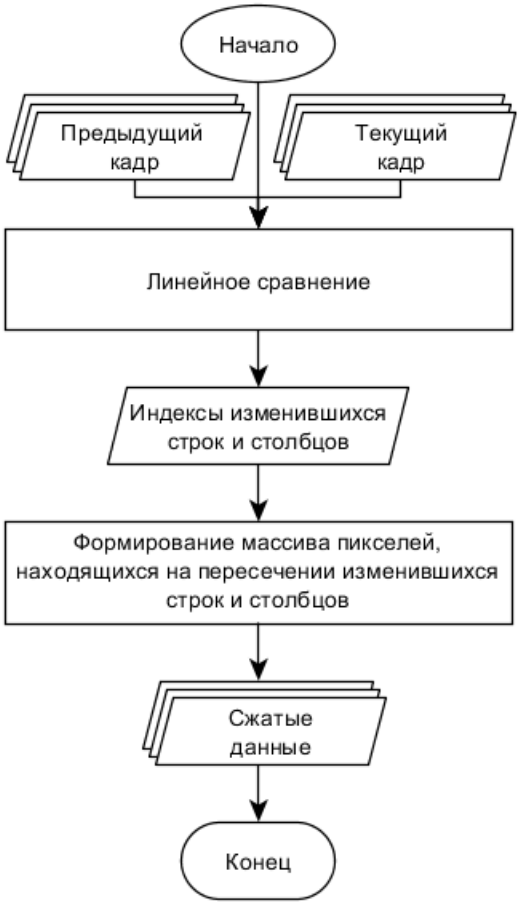


Рисунок 2.5 – Схема алгоритма отсечения неизменившихся строк и столбцов в кадре

Промежуточный кадр сравнивается с предыдущим кадром с целью выявления строк и столбцов, в которых есть изменившиеся пиксели. В результирующий массив записываются номера таких строк и столбцов, а также пиксели, находящиеся на их пересечении. Такие пиксели образуют изображение меньшего размера, которое затем может быть сжато теми же алгоритмами, что и ключевой кадр. Остальная часть промежуточного кадра при декодировании восстанавливается по предыдущему кадру.

Это алгоритм сжатия без потерь информации, обладающий трудоёмкостью $O(n)$.

Проведём сравнительный анализ алгоритма отсечения неизменившихся строк и столбцов в кадре и алгоритма оценки движения, адаптированного для обработки GUI-видеоданных (раздел 1.3). Если действия пользователя приводят к перемещению объектов по экрану, то алгоритм оценки движения даёт ощутимый эффект по степени сжатия. Примерами таких действий пользователя служат скроллинг, перемещение окна. Если же на экране появляются новые объекты, то алгоритм оценки движения и алгоритм линейного сравнения кадров продемонстрируют близкую степень сжатия. Таким образом, алгоритм оценки движения в некоторых случаях даёт выигрыш по степени сжатия, но обладает более низкой вычислительной эффективностью.

Проведём аналитическое сравнение алгоритма отсечения неизменившихся блоков и алгоритма отсечения неизменившихся строк и столбцов в кадре. Эти алгоритмы обладают трудоёмкостью $O(n)$. Алгоритм отсечения неизменившихся строк и столбцов в кадре обеспечивает более высокую степень сжатия при точечных изменениях на экране (например, ввод пользователем текста), когда независимое кодирование целого блока является избыточным, а также в случае, когда форма изменившейся области близка к прямоугольной. Примером такого изменения может служить сворачивание или открытие свёрнутого окна. Минимальная степень сжатия данных у алгоритма отсечения неизменившихся строк и столбцов в кадре достигается при изменении одной из диагоналей изображения, потому что при этом выявляемая изменившаяся область охватывает всё изображение. Примером такого изменения является перемещение окна по направлению, близкому к диагональному.

Рассмотрим более подробно некоторые из типичных изменений между кадрами GUI-видео. В случае, когда пользователь сворачивает или открывает свёрнутое окно, алгоритм отсечения неизменившихся строк и столбцов выявит и передаст на следующий этап сжатия в точности ту область, которая была изменена. Алгоритм отсечения неизменившихся блоков в общем случае будет

считать изменённой область, охватывающую реально изменившуюся область, так как блок пикселей считается изменённым, если изменился хотя бы 1 пиксель в блоке. Поэтому по периметру реально изменившейся области будет захвачена «буферная зона», толщиной максимум $T_{max} = (n - 1)$ пикселей из неизменившейся области, где $n \times n$ – размер блока, а T_{max} – максимальная величина буферной зоны.

Пусть $T_{ср} = T_{max} / 2 = (n - 1) / 2$, где $T_{ср}$ – это величина буферной зоны в среднем. Тогда среднее количество неизменившихся пикселей $N_{ср}$ из буферной зоны вокруг изменившейся области, которые алгоритм отсечения неизменившихся блоков будет считать изменившимися, можно оценить по следующей формуле:

$$N_{ср} \approx P \times T_{ср},$$

где P – это периметр реально изменившейся области, измеряемый в пикселях. Таким образом, при сворачивании или открытии свёрнутого окна алгоритм отсечения неизменившихся строк и столбцов обеспечивает значительно более высокую степень сжатия GUI-видеоданных.

Рассмотрим случай, в котором достигается минимальная степень сжатия данных у алгоритма отсечения неизменившихся строк и столбцов, а именно случай перемещения окна на некоторое расстояние по направлению, близкому к диагональному. Для определённости предположим, что окно было перемещено сверху вниз и слева направо (рисунок 2.6). Окну до движения соответствует прямоугольник с вершинами (x_{11}, y_{11}), (x_{12}, y_{12}) (x_{13}, y_{13}) (x_{14}, y_{14}). Окну после движения соответствует прямоугольник с вершинами (x_{21}, y_{21}), (x_{22}, y_{22}), (x_{23}, y_{23}), (x_{24}, y_{24}). Неизменившиеся области, которые попадают в охватывающий прямоугольник, на рисунке 2.6 заштрихованы. В этом случае алгоритм отсечения неизменившихся строк и столбцов будет считать изменившейся областью прямоугольник, координаты верхнего левого угла которого совпадают с координатами левого верхнего угла окна на ключевом кадре (до движения), а координаты правого нижнего угла совпадают с координатами правого нижнего угла окна на промежуточном кадре (после движения). Назовём такую область *охватывающим прямоугольником*. На рисунке 2.6 это прямоугольник с

вершинами (x_{11}, y_{11}), (x_{22}, y_{21}), (x_{23}, y_{23}), (x_{14}, y_{24}). Количество пикселей N в неизменившейся области, которая попадает в охватывающий прямоугольник, можно посчитать по следующей формуле:

$$N = (x_{22} - x_{12}) \times (y_{12} - y_{22}) \times 2$$

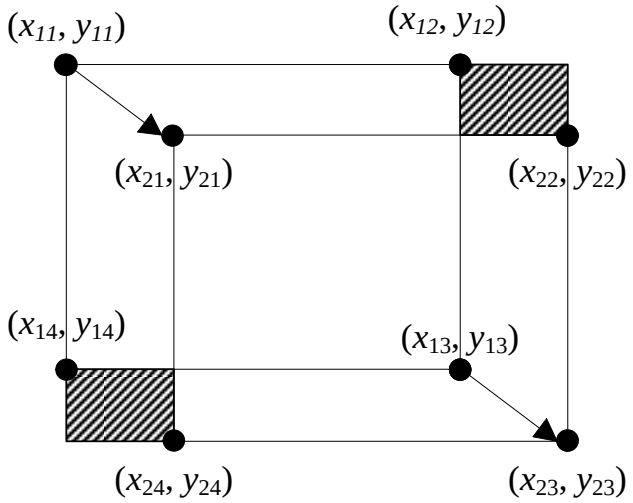


Рисунок 2.6 – Перемещение окна по направлению, близкому к диагональному

Размер буферной зоны, захватываемой алгоритмом отсечения неизменившихся блоков, в большинстве случаев значительно меньше той неизменившейся области, которая попадает в охватывающий прямоугольник.

Таким образом, ни один из этих алгоритмах не является лучшим по степени сжатия при всех рассмотренных типичных изменениях между кадрами GUI-видео.

2.2.2. Адаптивный алгоритм отсечения неизменившихся областей в кадре

Поскольку алгоритмы отсечения неизменившихся блоков и строк, столбцов в кадре обладают трудоёмкостью $O(n)$ и не предполагают вычислительно сложных операций, возможно осуществление и линейного и блочного сравнения в ходе сжатия кадра GUI-видео, что позволило разработать адаптивный алгоритм, сочетающий преимущества алгоритмов отсечения неизменившихся блоков и строк, столбцов в кадре (рисунок 2.7).

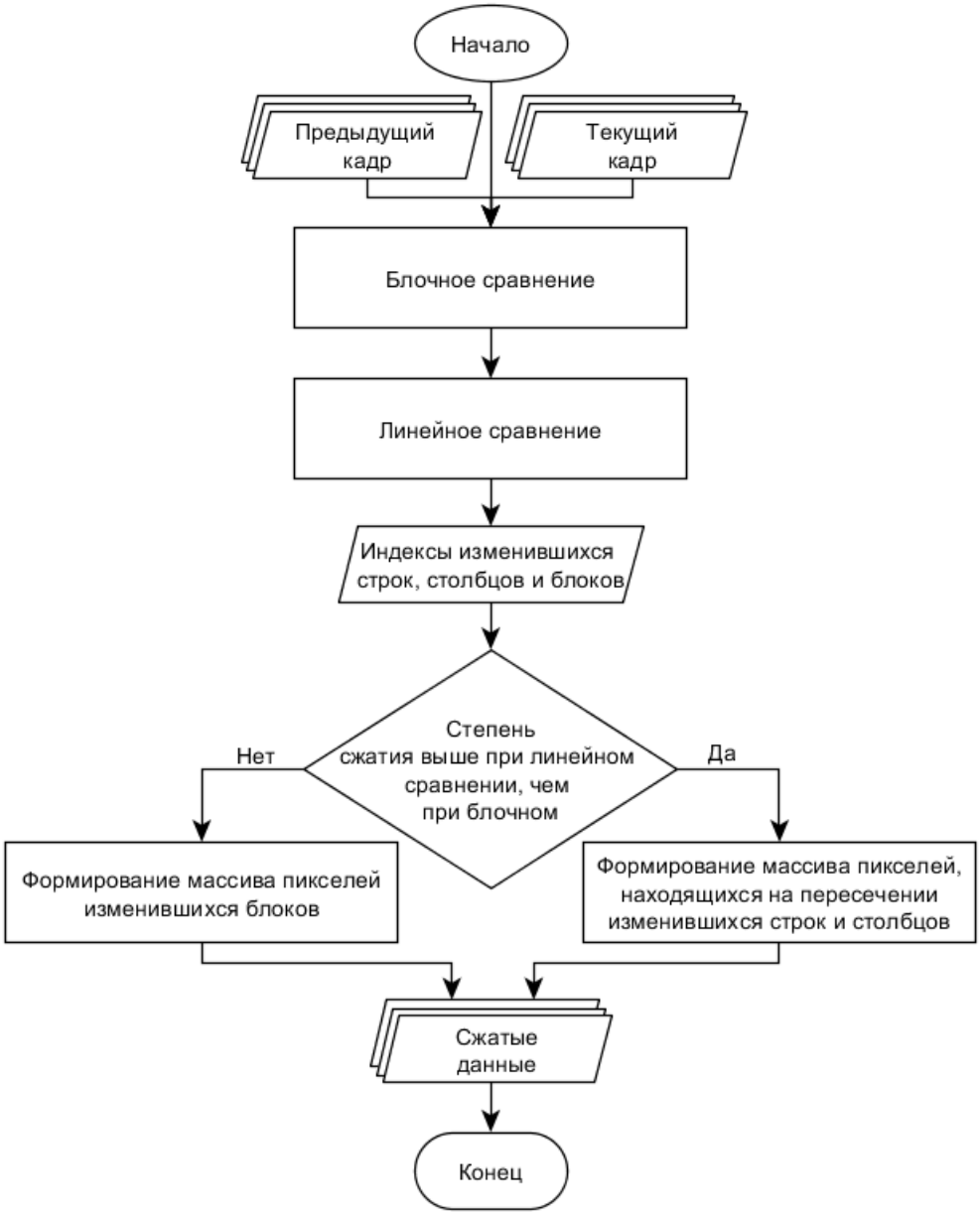


Рисунок 2.7 – Схема адаптивного алгоритма отсечения неизменившихся областей в кадре

Этот алгоритм предполагает выполнение линейного и блочного сравнения кадров на первом этапе и кодирование в соответствии с типом сравнения, обеспечивающим более высокую степень сжатия, на втором этапе. Таким образом, адаптивный алгоритм на основе выполняемого на первом этапе анализа входных данных, осуществляет выбор одного из указанных выше алгоритмов отсечения неизменившихся блоков и строк, столбцов в кадре для обработки данных на втором этапе.

Адаптивный алгоритм отсечения неизменившихся областей в кадре обладает трудоёмкостью $O(n)$ и производит сжатие без потерь информации. При этом неравенства 2.3 верны при любых входных данных.

$$\begin{cases} C_{адапт} \geq C_{лин} \\ C_{адапт} \geq C_{блоч} \end{cases}, \quad (2.3)$$

где $C_{адапт}$ – степень сжатия данных адаптивного алгоритма отсечения неизменившихся областей в кадре, $C_{блоч}$, $C_{лин}$, – степени сжатия данных алгоритмов отсечения неизменившихся блоков и строк, столбцов соответственно. Таким образом, адаптивный алгоритм в общем случае позволяет повысить степень сжатия видеоданных. Впервые адаптивный алгоритм представлен в [6].

2.2.3. Алгоритм оценки движения с учётом классификационных признаков

Как отмечено в разделе 1.3, существует алгоритм оценки движения, учитывающий многие особенности GUI-видеоданных [99], но не являющийся вычислительно эффективным, что препятствует его использованию для оперативного сжатия. В такой ситуации логичной выглядит идея провести классификацию движений в GUI-видео и разработать алгоритмы, выявляющие некоторые из этих классов движений, и работающие при этом значительно быстрее алгоритма, выявляющего все классы движений.

Предложена следующая классификация движений в GUI-видео:

1. Движения по вертикали и горизонтали (потенциально на большие расстояния). Это движения, осуществляемые вследствие вертикального и горизонтального скроллинга, нажатия пользователем на клавиши вниз, вверх, вправо, влево, Pg Up, Pg Dn.

2. Движения в ближайшей окрестности. Это движения, осуществляемые вследствие, например, достаточно плавного перетаскивания пользователем окна.

3. Прочие движения.

Как правило, подавляющее большинство движений различных объектов на экране, возникающих в ходе работы пользователя, относятся к (1) либо (2) классу. Например, такая закономерность наблюдается в видеозаписях [33, 60, 113].

Используя приведённую классификацию движений в GUI-видео, удалось разработать алгоритм оценки движения с учётом классификационных признаков. Рассмотрим его схему кодирования (рисунок 2.8).

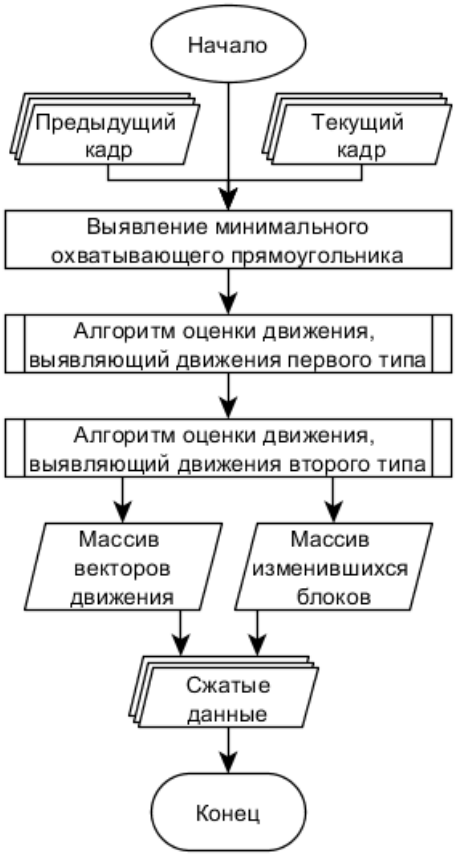


Рисунок 2.8 – Схема алгоритма оценки движения с учётом классификационных признаков

Для выявления движений классов (1) и (2) класса последовательно выполняются две модификации алгоритма оценки движения. Это модификации, осуществляющие поиск блока, соответствующего текущему блоку:

1. только по вертикали и по горизонтали;

2. во всех направлениях в ближайшей окрестности текущего блока.

Таким образом, выполнение алгоритма, осуществляющего полный перебор возможных векторов движения, заменяется выполнением двух существенно менее трудоёмких алгоритмов. При таком подходе будут выявлены только движения (1) и (2) классов. Предлагается кодировать блоки, для которых не найдено точное

соответствие, независимо от других кадров, так как в силу дискретно-тоновой природы GUI-видеоданных случай, когда блок изменился незначительно, встречается существенно реже, чем для традиционных видеоданных (раздел 1.3).

Для повышения вычислительной эффективности алгоритма оценки движения при обработке GUI-видеоданных предлагается использовать *технику предварительного сравнения диагональных элементов*, учитывающую горизонтальную и вертикальную корреляцию пикселей в кадре. При сравнении двух блоков (блока текущего кадра и блока предыдущего кадра, соответствующего текущему пробному вектору движения) сначала происходит сравнение диагональных элементов этих блоков. И только в случае, когда все диагональные элементы попарно равны между собой, происходит сравнение всех элементов этих блоков. Эта техника одновременного учитывает горизонтальную и вертикальную корреляцию пикселей движущегося объекта. В то время как при полном сравнении значений блоков учитывается горизонтальная, а затем вертикальная корреляция либо наоборот. В обеих предложенных модификациях алгоритма оценки движения используется эта техника.

Перейдём к детальному изложению предлагаемых алгоритмов (1) и (2), которые отличаются областью поиска вектора движения.

Шаг 1. Начало.

Шаг 2. Заполнить массив **F** (0 – вектор движения для блока с этим индексом ещё не найден, 1 – найден). При инициализации элементы массива **F** устанавливаются в 1 для тех блоков, которые не изменились по сравнению с предыдущим кадром.

Шаг 3. Вычислить минимальный прямоугольник, охватывающий все изменившиеся относительно предыдущего кадра области текущего кадра, *minimalRectangle*. Именно в пределах этого прямоугольника в дальнейшем будет осуществляться поиск соответствия для текущего блока. Для вычисления минимального охватывающего прямоугольника используется алгоритм отсечения неизменившихся строк и столбцов в кадре (п. 2.2.1).

Шаг 4. Заполнить массив начальных векторов движения **I** значением {0; 1}; задать $i = 0$.

Шаг 5. Если $F_i = 0$, то перейти на шаг 6, иначе на шаг 11.

Шаг 6. Если v_{cur} выходит за границы *minimalRectangle*, или выходит за пределы ближайшей окрестности текущего блока (для модификации (2)), то перейти на шаг 11, иначе на шаг 7**.**

Шаг 7. Задать значение пробного вектора движения $v_{cur} = I_i$. Это может быть начальный вектор движения по умолчанию {1; 0} или некоторое другое значение, установленное уже в ходе выполнения алгоритма. Во втором случае после проверки начального вектора движения выполняется присваивание $v_{cur} = \{1; 0\}$.

Шаг 8. Если диагональные элементы блоков B_i и B'_{cur} попарно равны между собой, где B'_{cur} – блок предыдущего кадра, соответствующий вектору движения v_{cur}**,** то перейти на шаг 9, иначе $i = i + 1$ и перейти на шаг 5.

Шаг 9. Если $B_i = B'_{cur}$ (попарно равны соответствующие элементы блоков), то $F_i = 1$ и перейти на шаг 10, иначе $v_{cur} = nextMotionVector(v_{cur})$, перейти на шаг 8.

Шаг 10. Задать $I_k = I_i$ для всех блоков B_k, смежных с B_i. Таким образом происходит предсказание вероятного вектора движения для соседних блоков.

Шаг 11. Установить $i = i + 1$. Если $i < n$, то перейти на шаг 5, иначе на шаг 12.

Шаг 12. Конец.

Рассмотрим принцип работы используемой на шаге 9 функции *nextMotionVector*(). Реализация этой функции различается для модификаций алгоритма оценки движения (1) и (2). Для модификации (1) функция *nextMotionVector*() последовательно проверяет векторы движения, начиная с $i = 1$, $\{0; i\}$, $\{i; 0\}$, $\{0; -i\}$, $\{-i; 0\}$. Для модификации алгоритма оценки движения (2) перебираются возможные векторы движения построчно в диапазоне [(i − *maxXShift*; j − *maxYShift*); (i + *maxXShift*; j + *maxYShift*)], где (i; j) – текущая позиция в текущем кадре, а (*maxXShift*; *maxYShift*) – максимальные отклонения от текущей позиции по оси x и y соответственно. При увеличении значений *maxXShift* и

maxYShift алгоритм оценки движения способен выявить больший процент движений, но при этом возрастает время выполнения алгоритма.

Стоит отметить, что в обоих алгоритмах используются техники, направленные на достижение более высокой степени сжатия GUI-видеоданных, рассмотренные в п. 1.3.2. В соответствии с рекомендациями экспертной группы MPEG для GUI-видео [93] в алгоритме используются только целочисленные пиксельные отступы в векторах движения.

Перейдём к сравнительному анализу трудоёмкостей существующих и разработанных алгоритмов в наилучшем и наихудшем случаях. В наилучшем случае, когда кадр не изменился относительно предыдущего, трудоёмкость всех рассмотренных алгоритмов оценки движения $O(n)$. Наихудшим случаем для алгоритма оценки движения будем считать полное изменение кадра относительно предыдущего кадра, когда невозможно найти соответствие для какого-либо блока:

$$\nexists i,j\ (B_i = B'_j),$$

где B_i – блок текущего кадра, B'_j – блок предыдущего кадра, а равенство блоков означает попарное равенство соответствующих элементов.

Трудоёмкость алгоритма оценки движения, осуществляющего полный перебор, в наихудшем случае квадратичная – $O(n^2)$, или более подробно:

$$T(n) = n^2 / k \times const, \qquad (2.4)$$

где k – количество пикселей в блоке, n – количество пикселей в изображении, *const* – константа, определяющая количество операций, выполняемых при сравнении пары блоков.

Трудоёмкость разработанного алгоритма, осуществляющего поиск в ближайшей окрестности текущего блока, всегда линейная – $O(n)$, так как для каждого блока рассматривается константное количество векторов движения. Трудоёмкость разработанного алгоритма, осуществляющего поиск по вертикали и горизонтали, в наихудшем случае $O(n \times \sqrt{n})$, или более подробно:

$$T(n) = n / k \times (M + N) \times const, \qquad (2.5)$$

где k, n, $const$ имеют то же значение, что и в формуле 2.4, M – ширина изображения в пикселях, N – высота изображения в пикселях.

На основании вышеизложенного можно сделать вывод, что разработанный алгоритм имеет более низкий порядок трудоёмкости в наихудшем случае, чем существующий, что позволяет выполнить ограничение $t(x) \leq T_{max}$ задачи 1.1.

На момент написания данной работы разрешение экрана современных мониторов достигает значения 2560 × 1600 пикселей. При таком разрешении экрана и размере блока 16 × 16 формула 2.5 примет вид

$$T(n) = n \times 16{,}25 \times const.$$

Исходя из вышеизложенного, можно сделать вывод, что при обработке GUI-видеоданных, полученных путём фиксации снимков экрана существующих мониторов, трудоёмкость алгоритма, осуществляющего поиск блоков по вертикали и горизонтали, сопоставима с линейной.

2.3. Требования и концептуальные основы создания программного обеспечения кодека для сжатия GUI-видеоданных

2.3.1. Требования к программному обеспечению кодека

Существуют базовые требования, выдвигаемые практически ко всем кодекам для сжатия видеоданных. Это в первую очередь возможность кодировать и декодировать кадры, возможность произвольного доступа к кадрам в закодированном массиве данных. Рассмотрим требования к кодекам для сжатия GUI-видеоданных (далее в разделе – кодеков).

1. Кодек должен осуществлять высокопроизводительную обработку данных, чтобы сжимать GUI-видеоданные, которые, как правило, характеризуются высоким разрешением кадров, в оперативном режиме. Это требование соответствует условию задачи 1.1 $\min_x C_{cpu}(x)$.

2. Кодек должен обладать высокой ресурсоэффективностью, то есть иметь низкий уровень использования центрального процессора и оперативной памяти.

Это позволит кодеку функционировать в фоновом режиме, не влияя на другие приложения и сервисы операционной системы. Это требование соответствует условию $\min_x C_{cpu}(x)$ и ограничению $M_{avr}(x) \leq M_{max}$ задачи 1.1.

3. Кодек должен сжимать данные с высокой степенью сжатия ввиду высокого разрешения кадров GUI-видео и, как следствие, сравнительно большого объёма входных данных. Такое требование соответствует ограничению $C_{comp}(x) \geq C_{min}$ задачи 1.1.

4. Кодек должен осуществлять сжатие без потерь информации, так как даже небольшой процент потерь может привести к неприемлемому визуальному ухудшению качества GUI-видеоданных (раздел 1.1). Для выполнения этого требования необходимо, чтобы входные данные кодека были сформированы из снимков экрана, сохранённых с исходным разрешением.

5. Для упрощения встраивания кодека в различные программные комплексы он должен предоставлять функциональный и объектный интерфейс. Функциональный интерфейс позволяет использовать кодек в приложениях, написанных на языках, ориентированных на процедурное программирование (например, C). Объектный интерфейс предназначен для использования в приложениях, написанных на объектно-ориентированных языках.

2.3.2. Концептуальные основы создания программного обеспечения кодека

Учитывая вышеизложенные требования, предъявляемые к кодеку сжатия GUI-видеоданных, определим общие принципы его построения. Архитектура кодека, иллюстрирующая применение таких принципов, представлена на рисунке 2.9. Блоки с серым и белым фоном соответствуют подсистемам, входящим и не входящим в состав ПО кодека, соответственно.

1. Язык программирования, выбранный для реализации кодека должен обеспечивать высокую гибкость при работе с оперативной памятью. Такой принцип выбора языка программирования направлен на соответствие требованию (1) к ПО кодека.

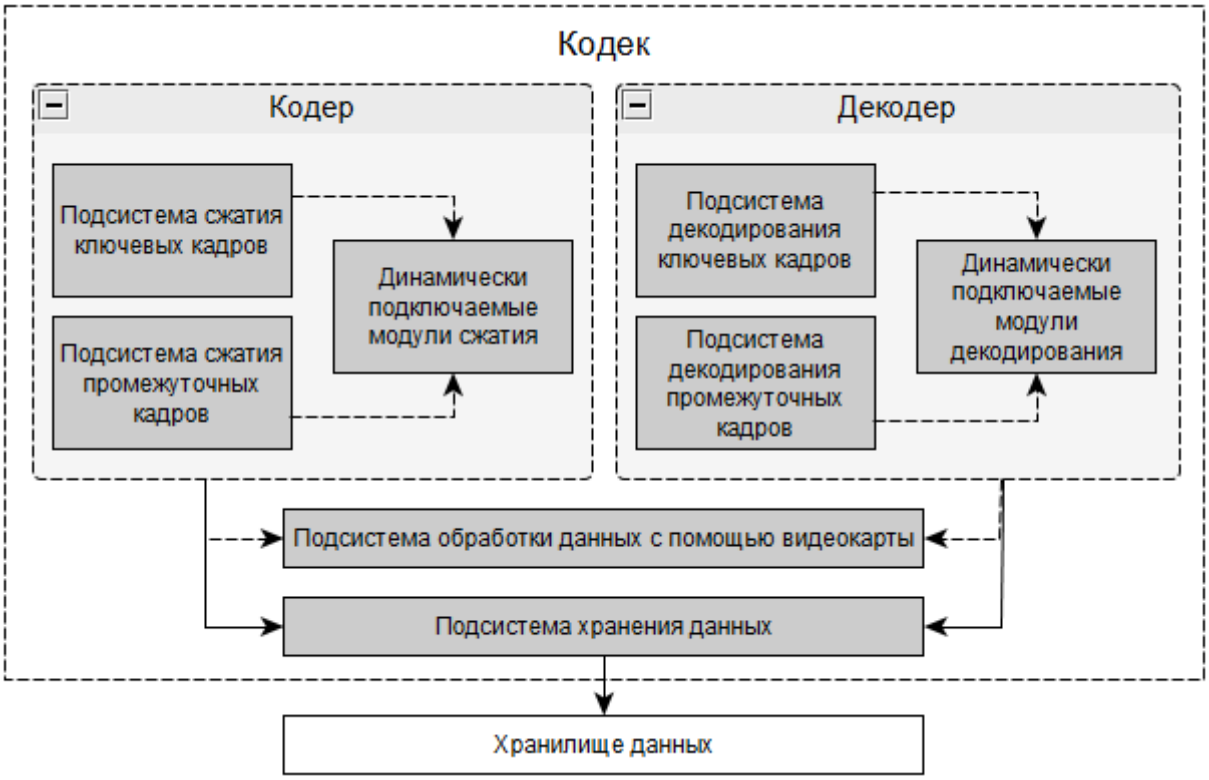


Рисунок 2.9 – Обобщённая архитектура кодека

2. В целях уменьшения нагрузки на ЦП отдельные этапы обработки данных, хорошо поддающиеся распараллеливанию, должны быть перенесены на видеокарту. Использование вычислительных ресурсов видеокарты направлено на удовлетворение требованию (2) к ПО кодека. Вместе с тем кодек должен оставаться полностью функциональным, если аппаратная платформа не оснащена видеокартой или видеокарта не поддерживает технологии GPGPU (п. 1.4.1). В этом случае все этапы обработки данных должны выполняться на ЦП. Поэтому на рисунке 2.9 к *Подсистеме обработки данных с помощью видеокарты* ведёт пунктирная линия.

3. Для достижения адаптивности к уровню производительности аппаратной платформы предлагается подключать отдельные модули сжатия динамически при условии наличия достаточного количества вычислительных ресурсов. В противном случае должен использоваться базовый набор модулей сжатия, что приведёт к уменьшению нагрузки на ЦП, но будет сопровождаться некоторым снижением степени сжатия. Поэтому на рисунке 2.9 к *Динамически подключаемым модулям* ведёт пунктирная линия.

4. Ввиду того, что алгоритмы сжатия ключевых и промежуточных кадров существенно отличаются (разделы 1.2, 1.3), целесообразно обособить подсистемы, выполняющие их обработку. Поэтому предлагается создать в рамках Кодера *Подсистему сжатия ключевых кадров* и *Подсистему сжатия промежуточных кадров*, а в рамках декодера *Подсистему декодирования ключевых кадров* и *Подсистему декодирования промежуточных кадров*.

Подсистема хранения данных предназначена для сохранения и чтения закодированных данных, поиска кадров в закодированном массиве данных. В качестве *Хранилища данных* может выступать файловая система или база данных.

Впервые концептуальные основы создания ПО кодека сжатия GUI-видеоданных представлены в [7].

2.4. Выводы

1. Разработан оригинальный алгоритм пространственного группового кодирования, имеющий линейную трудоёмкость и позволяющий устранять пространственную избыточность в дискретно-тоновых изображениях лучше, чем существующие алгоритмы группового кодирования. Алгоритм способен выявлять три часто встречающихся типа групп одноцветных пикселей (горизонтальные и вертикальные линии, прямоугольники).

2. Предложен оригинальный гибридный сдвигово-групповой алгоритм, объединяющий преимущества алгоритма группового кодирования и сдвигового алгоритма, заключающиеся в способности учитывать горизонтальную и вертикальную корреляцию пикселей кадра и устранять избыточность, связанную с частым чередованием цветов. На каждом шаге гибридного алгоритма выполняются шаги входящих в его состав алгоритмов, заключающиеся в выявлении группы одноцветных пикселей и замене байтов цвета пикселя, где возможно, на адресную ссылку. Показано, что гибридный сдвигово-групповой алгоритм способен устранять больше типов избыточности в дискретно-тоновых данных, чем каждый из входящих в его состав алгоритмов по отдельности.

3. Разработаны новые алгоритмы со сниженной пространственной избыточностью, позволяющие увеличить степень сжатия GUI-видеоданных по сравнению с гибридным сдвигово-групповым алгоритмом. Эти алгоритмы предполагают на первом этапе выполнение гибридного сдвигово-группового алгоритма. Один алгоритм со сниженной пространственной избыточностью нацелен на достижение максимальной степени сжатия, поэтому финальное сжатие осуществляется алгоритмом, обеспечивающим высокую степень сжатия. Другой алгоритм со сниженной пространственной избыточностью обладает сбалансированными показателями вычислительной эффективности и степени сжатия за счёт финального сжатия алгоритмом, имеющим более высокую вычислительную эффективность.

4. Разработан оригинальный алгоритм отсечения неизменившихся строк и столбцов в кадре, демонстрирующий более высокую степень сжатия, чем аналоги, в таких часто встречающихся при обработке GUI-видеоданных случаях, как наличие точечных или близких по форме к прямоугольной изменений текущего кадра относительно предыдущего.

5. Разработан адаптивный алгоритм отсечения неизменившихся областей кадра, осуществляющий выбор алгоритма с более высокой степенью сжатия для текущей пары кадров. В качестве такого алгоритма может быть выбран алгоритм отсечения неизменившихся строк и столбцов либо алгоритм отсечения неизменившихся блоков.

6. Разработан оригинальный алгоритм оценки движения с учётом классификационных признаков, обладающий более высокой вычислительной эффективностью по сравнению с существующими аналогами. Представленный алгоритм выполняет поиск наиболее часто встречающихся классов движения в соответствии с предложенной классификацией.

7. Сформулированы требования к ПО кодека для сжатия GUI-видеоданных. Такой кодек должен осуществлять высокопроизводительное сжатие данных без потерь с высокой степенью сжатия, обладать высокой ресурсоэффективностью, предоставлять функциональный и объектный интерфейс. Предложены

концептуальные основы создания ПО кодека для сжатия GUI-видеоданных, включающие обобщённую архитектуру такого кодека и упрощающие создание аналогичных систем.

ГЛАВА 3. ОСОБЕННОСТИ ПРАКТИЧЕСКОЙ РЕАЛИЗАЦИИ АЛГОРИТМОВ СЖАТИЯ GUI-ВИДЕОДАННЫХ

Проводится подбор оптимальных параметров функционирования разработанных алгоритмов сжатия GUI-видеоданных. В том числе, описаны способ выбора флагов, направленный на увеличение суммарной степени сжатия, дополнительные структуры данных, используемые в ходе выполнения алгоритма для повышения вычислительной эффективности. Приведены результаты сравнительного анализа реализаций разработанных и существующих алгоритмов.

3.1. Сжатие ключевых кадров

В целях повышения вычислительной эффективности и степени сжатия данных гибридным сдвигово-групповым алгоритмом проанализированы различные варианты реализации алгоритмов, входящих в его состав.

При реализации **алгоритма пространственного группового кодирования** рассмотрены варианты, различающиеся по следующим параметрам:

1. Максимальное количество байтов, отводимых для хранения количества пикселей в группе.

2. Использование битового или байтового формата.

При максимальном количестве в 1 байт может быть закодирована группа размером $2^8 – 1$ пикселей. В альтернативной реализации при различных условиях количество байтов, отводимых для хранения количества пикселей в группе, может составлять 1 или 2 байта. Если количество подряд идущих пикселей одного цвета меньше, чем 2^7, то используется только 1 байт, первый бит которого равен 0. В противном случае используются оба байта, причём первый бит первого из этих байтов равен 1, а второй байт является младшим.

Байтовый формат предполагает отсутствие битовых флагов. В этом случае даже при кодировании одиночного пикселя в результирующий массив записывается 3 байта цвета пикселя и байт, равный 0. Битовый формат

предполагает наличие битовых флагов. В этом случае при кодировании одиночного пикселя в результирующий массив записываются 3 байта цвета пикселя, а затем бит, равный 0. При кодировании группы пикселей в результирующий массив записываются 3 байта цвета пикселя, бит, равный 1, а затем однобайтовое количество подряд идущих пикселей этого цвета. Вследствие вставки битовых флагов, байты цвета пикселя или байт, кодирующий количество пикселей в группе, могут начинаться не с первого бита текущего байта результирующего файла.

Байтовый формат алгоритма группового кодирования будет более выгоден при длинных группах одноцветных пикселей, а битовый формат – при значительном количестве единичных пикселей определённого цвета (например, изображение, содержащее текст). Максимальная длина серии одноцветных пикселей в байтовом формате составляет $2^{15} - 1$, а в битовом формате – всего $2^8 - 1$. Вместе с тем на каждом одиночном пикселе определённого цвета битовый формат экономит 7 бит по сравнению с байтовым форматом. Таким образом, при сжатии GUI-видеоданных в различных случаях предпочтительнее может оказаться тот или иной формат алгоритма группового кодирования.

Для повышения степени сжатия данных при кодировании и декодировании алгоритма пространственного группового кодирования используется вспомогательная структура данных, которая представляет собой массив флагов, где один флаг соответствует одному пикселю исходного изображения. Если этот флаг равен 0, соответствующий ему пиксель ещё не закодирован, иначе – данный пиксель уже закодирован. Уже закодированные пиксели пропускаются при кодировании. При этом возможна следующая ситуация: пиксель $P[i]$ уже закодирован ранее, попав в вертикальную или прямоугольную группу одноцветных пикселей. Пусть текущий пиксель – это $P[i-1]$. Допустим также, что пиксель $P[i+1]$ имеет такой же цвет, как $P[i-1]$. В этом случае $P[i-1]$ и $P[i+1]$ могут быть закодированы, как горизонтальная группа пикселей.

Ниже описана закономерность (формула 3.1), выявленная в ходе анализа GUI-видеоданных и подтверждённая экспериментально, которая позволяет

оптимизировать способ задания флагов, записываемых в результирующий массив (таблица 3.1).

$$\begin{cases} P_{\text{п}} + P_{\text{г}} + P_{\text{в}} < P_{\text{о}} \\ P_{\text{г}} < P_{\text{п}} \\ P_{\text{в}} < P_{\text{п}} \end{cases}, \quad (3.1)$$

где $P_{\text{п}}$, $P_{\text{г}}$, $P_{\text{в}}$, $P_{\text{о}}$ – вероятность того, что на очередном шаге алгоритма для кодирования будет выбрана прямоугольная область, горизонтальная линия, вертикальная линия, одиночный пиксель соответственно.

Таблица 3.1 – Способ задания флагов алгоритма пространственного группового кодирования

№	Область	Флаг
1	Одиночный пиксель	0
2	Прямоугольная область	11
3	Горизонтальная линия	100
4	Вертикальная линия	101

Таким образом, в ходе выполнения алгоритма пространственного группового кодирования чаще встречающимся случаям сопоставляются более короткие флаги.

При реализации **сдвигового алгоритма** рассмотрены варианты, различающиеся по следующим параметрам:

1. Отсчёт адресной ссылки по исходным или закодированным данным.

2. Использование битового или байтового формата.

При отсчёте адресной ссылки по исходным данным становится возможным запоминать не количество байтов, а количество пикселей до предыдущего пикселя с таким же цветом. Следовательно, ссылаться можно до 256 пикселей назад. Такая модификация сдвигового алгоритма способна увеличить степень сжатия дискретно-тоновых изображений сдвиговым алгоритмом, как показали результаты тестирования, представленные ниже в этой главе. Но для сдвигового алгоритма в составе гибридного алгоритма такая модификация понизила бы степень сжатия, так как среднее количество пикселей в группе для него составляет несколько десятков (такие результаты получены опытным путём), а на одну группу при этом затрачивается всего несколько байтов. Поэтому при отсчёте

адресной ссылки по закодированному файлу можно покрыть гораздо большее количество пикселей.

Оптимального сочетания описанных выше подходов к отсчёту адресной ссылки позволяет добиться разделение результирующих данных гибридного сдвигово-группового алгоритма на несколько массивов, в одном из которых хранится остаточное изображение (раздел 2.1). При отсчёте адресной ссылки по остаточному изображению удаётся добиться максимального охвата пикселей исходного изображения.

Перейдём к выбору формата данных сдвигового алгоритма.

В случае байтового формата под адресную ссылку всегда отводится один байт. Если цвет встретился впервые или пиксель эквивалентного цвета встречался слишком далеко от текущей позиции, чтобы можно было на него сослаться, то указывается адресная ссылка равная 0. Затем следуют байты цвета пикселя. При использовании битового формата в случае, когда цвет встретился впервые или количество байтов до предыдущего пикселя с таким же цветом превышает $2^8 - 1$, в результирующий массив записывается бит, равный нулю, а затем байты цвета пикселя. Иначе в результирующий файл записывается бит, равный 1, а затем однобайтовая адресная ссылка.

Байтовый формат сдвигового алгоритма предпочтительнее в двух случаях:

1. Когда количество цветов, присутствующих в изображении, мало́.

2. Когда расстояние между пикселями одного цвета в большинстве случаев не превышает длину адресной ссылки: $2^8 - 1$ байтов.

Условие (1) является одной из предпосылок для возникновения условия (2). В таких случаях байтовый формат выигрывает за счёт меньшего числа накладных расходов (не используются флаги).

Битовый формат напротив, экономичнее при невыполнении вышеописанных условий, так как экономия битового формата относительно байтового составляет 7 бит на каждом цвете пикселя, встреченном впервые, а также на пикселе, цвет которого встречался слишком далеко от текущей позиции, чтобы на него ссылаться.

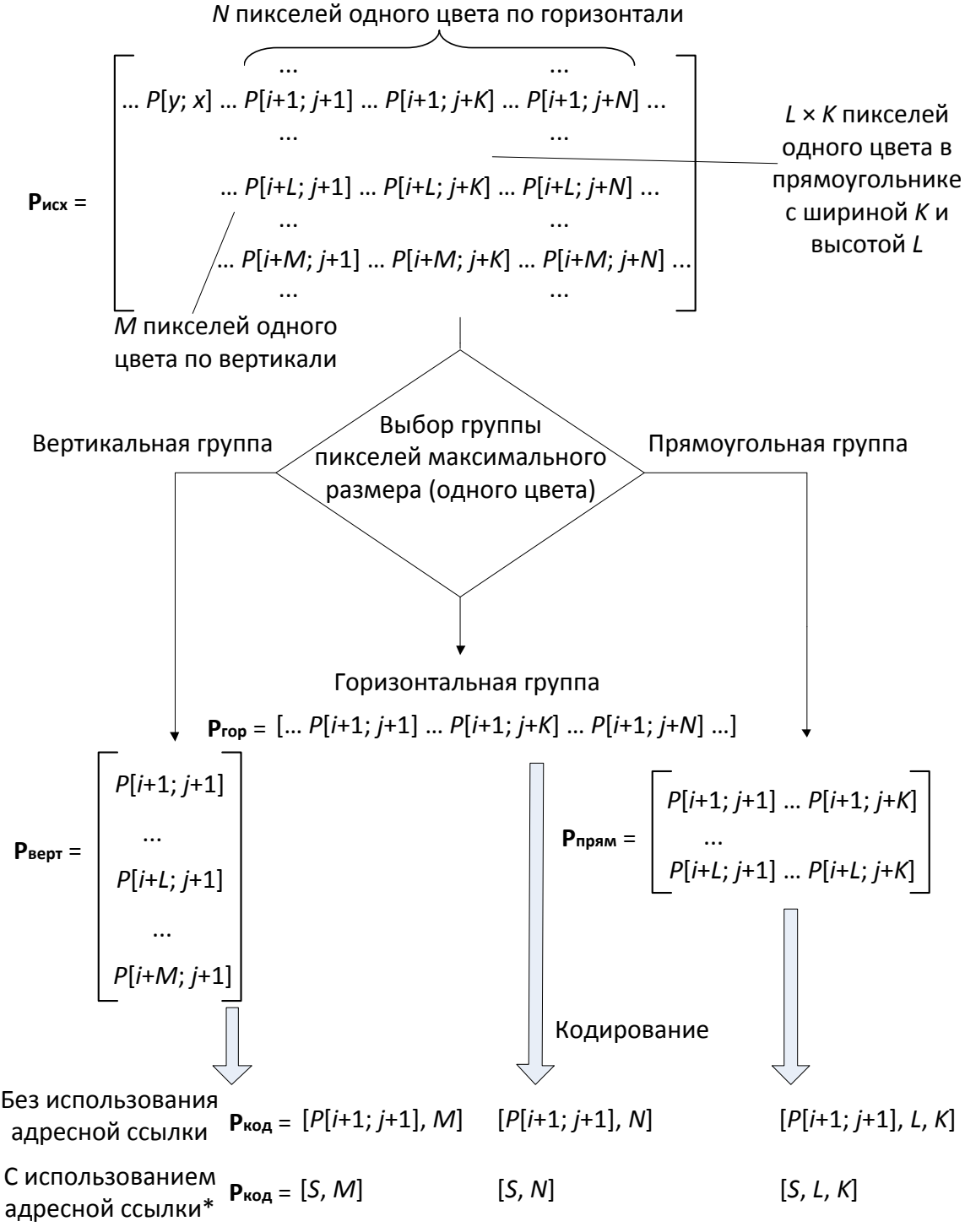
При использовании битового формата для алгоритма группового кодирования и сдвигового алгоритма в составе гибридного алгоритма частоты различных значений байтов результирующего формата гибридного сдвигово-группового алгоритма в значительной степени уравняются, так как каждый флаг будет производить сдвиг на 1 бит. Поэтому эффективность финального сжатия статистическими алгоритмами, основанными на сопоставлении часто встречающимся символам более коротких кодов и сопоставлении редко встречающимся символам более длинных кодов, резко падает.

При реализации гибридного сдвигово-группового алгоритма применена техника, позволяющая повысить степень статистического сжатия и заключающаяся в объединении флагов в группы по 8 бит. Для этого заводится небольшой буфер (несколько десятков байтов), в который записываются данные нескольких последовательных групп, которым соответствует в сумме 8 флагов. Сначала в результирующий файл записывается байт флагов, а затем все данные из буфера. Буфер очищается. При использовании этого способа, хаотичные изменения, вносимые в результирующий формат битами флагов, локализуются в отдельных байтах, а частоты байтов цвета пикселей не изменяются по сравнению с байтовым форматом.

Для ускорения работы алгоритма, как при кодировании, так и при декодировании используется хэш-таблица. При кодировании ключом хэш-таблицы является цвет, а значением – номер последнего просмотренного пикселя с таким цветом в остаточном изображении. При декодировании ключом хэш-таблицы является индекс последнего просмотренного пикселя с таким цветом в остаточном изображении, а значением – цвет.

Рассмотрим более подробно модификацию гибридного сдвигово-группового алгоритма, подтвердившую эффективность в ходе экспериментальных исследований (п. 3.2.1), в которой используются битовый формат, и включающую алгоритм пространственного группового кодирования.

На рисунке 3.1 описан формат данных этого алгоритма.



* – условие использования адресной ссылки *S* приведено за пределами диаграммы

Рисунок 3.1 – Варианты формирования результирующей структуры данных гибридного сдвигово-группового алгоритма с битовым форматом, включающего алгоритм пространственного группового кодирования

Условие использования адресной ссылки *S* может быть записано в следующем виде:

$$(\exists y \exists x)\ (P[y; x]=P[i+1; j+1]) \cap$$
$$(0 < S(P[y; x], P[i+1; j+1]) \le S_{max}),$$

где S_{max} – максимальное расстояние между пикселями, один из которых может быть заменён адресной ссылкой на второй пиксель, W – количество пикселей в одной строке изображения, функция S задана как:

$$S(P[i_1; j_1], P[i_2, j_2]) = W \times i_2 + j_2 - W \times i_1 - j_1$$

В таблице 3.2 описан способ задания флагов этого алгоритма.

Таблица 3.2 – Способ задания флагов гибридного сдвигово-группового алгоритма с битовым форматом, включающего алгоритм пространственного группового кодирования

№	Область	Использование адресной ссылки	Флаг
1	Одиночный пиксель	Да	10
		Нет	00
2	Прямоугольная область	Да	111
		Нет	011
3	Горизонтальная линия	Да	1100
		Нет	0100
4	Вертикальная линия	Да	1101
		Нет	0101

C_{min}: 24 / 26.

C_{max}: (256 × 256 × 24) / 27.

3.2. Результаты экспериментальных исследований

Перейдём к экспериментальному анализу эффективности представленных алгоритмов сжатия GUI-видеоданных. В разделе приведены результаты исследований, полученных при практическом сравнении разработанных и существующих алгоритмов сжатия.

При тестировании алгоритмов сжатия применялась методика, предполагающая выполнение для каждой реализации алгоритма и каждого тестового набора данных следующей процедуры. Каждый элемент набора данных (кадр при сжатии ключевых кадров, пара кадров при сжатии промежуточных кадров) в наборе сжимался 100 раз с усреднением полученных результатов. Таким образом формировались таблицы, каждой строке которой соответствовал алгоритм, а каждому столбцу – элемент набора данных. После этого результаты

тестирования по каждому набору данных усреднялись для каждого алгоритма. Проверка статистической значимости отличий в скорости и степени сжатия различных реализаций алгоритмов проводилась с помощью критерия Вилкоксона [3], а доверительная вероятность при этом принята равной 95%.

Тестирование проводилось на платформе со следующими характеристиками: процессор Intel Core 2 Duo E6750 2,66 ГГц; ОП DDR2 2Гб; ОС Windows 7. На момент подготовки диссертационной работы тестовая платформа находилась в среднем сегменте по производительности.

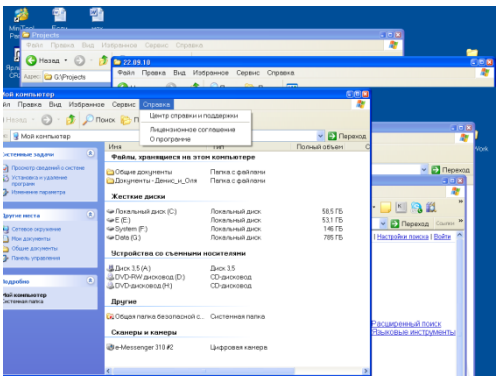
3.2.1. Сжатие ключевых кадров

Обратимся к результатам экспериментальных исследований, полученных для алгоритмов сжатия ключевых кадров. Применительно к GUI-видеоданным дискретно-тоновые изображения разделяют на изображения с преобладанием текста и с преобладанием деловой графики. При практическом сравнении алгоритмов сжатия GUI-видеоданных представляют интерес изображения, полученные при работе пользователя со стандартными приложениями различных ОС (рисунок 3.2).

При тестировании использованы снимки экрана, сделанные в трёх ОС с различным соотношением дискретно-тоновой и непрерывно-тоновой графики (таблица 3.3). Для Windows XP характерно преобладание дискретно-тоновой графики, в интерфейсе Ubuntu Gnome 14 в значительной степени присутствуют непрерывно-тоновые данные. Графический интерфейс Windows 7 занимает промежуточное положение по соотношению дискретно-тоновой и непрерывно-тоновой графики по сравнению с Windows XP и Ubuntu Gnome 14.

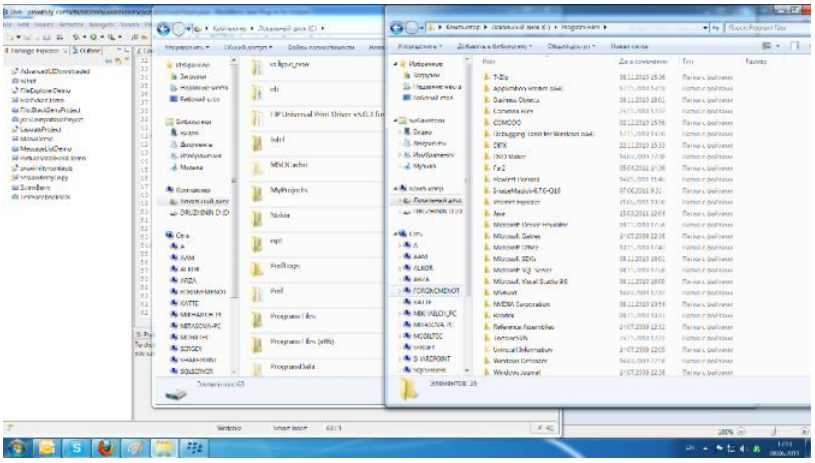
Таблица 3.3 – Количество непрерывно-тоновых элементов в графическом интерфейсе различных ОС

№	Операционная система	Количество непрерывно-тоновых элементов
1	Windows XP	Малое
2	Windows 7	Среднее
3	Ubuntu Gnome 14	Большое



а

б



в

Рисунок 3.2 – Изображения, полученные при работе пользователя со стандартными приложениями различных ОС: а) Windows XP; б) Ubuntu Gnome 14; в) Windows 7

Для тестирования использовались снимки экрана следующих типов:

1. Изображения с преобладанием текста.
2. Изображения с преобладанием деловой графики.
3. Windows XP.
4. Windows 7.
5. Ubuntu Gnome 14.04.

Каждый тестовый набор содержит 10 изображений. Снимки экрана (3-5) сделаны при работе пользователя со стандартными приложениями Windows 7, Windows XP, Ubuntu Gnome (список файлов и папок в разных видах, панель управления и т. д.). Типы снимков экрана (1), (2) являются более общими, а типы снимков экрана (3-5) – более специфичными. При этом типы снимков экрана (3-5)

пересекаются с первыми двумя. Снимки экрана (1-3) доступны по ссылке [37]. Снимки экрана (4) и (5) доступны по ссылкам [38], [39] соответственно.

При тестировании каждое изображение из наборов данных (1-3), (5) имело разрешение 1024 × 768 и глубину цвета в 32 бита. Изображения из набора данных (4) имели разрешение 1920 × 1080.

Обратимся к результатам практического сравнения разработанных алгоритмов для выявления лучших из них по степени сжатия и вычислительной эффективности. В тестировании принимали участие следующие реализации разработанных алгоритмов:

1. Сдвиговый алгоритм с байтовым форматом.

2. Сдвиговый алгоритм с битовым форматом.

3. Гибридный алгоритм с байтовым форматом и алгоритмом одномерного группового кодирования. Сокращённо – гибридный алгоритм с байтовым форматом (одномерный).

4. Гибридный алгоритм с битовым форматом и алгоритмом одномерного группового кодирования. Сокращённо – гибридный алгоритм с битовым форматом (одномерный).

5. Гибридный алгоритм с битовым форматом и алгоритмом CCITT Group 4 (п. 1.2.1). Сокращённо – гибридный алгоритм с битовым форматом и CCITT Group 4.

6. Гибридный алгоритм с битовым форматом и алгоритмом пространственного группового кодирования. Сокращённо – гибридный алгоритм с битовым форматом (пространственный).

На рисунке 3.3 приведены результаты тестирования разработанных алгоритмов на наборе тестовых данных № 2. Для остальных наборов данных соотношение показателей рассматриваемых алгоритмов аналогично. Полные результаты тестирования различных реализаций представлены в приложении В в таблице 1.

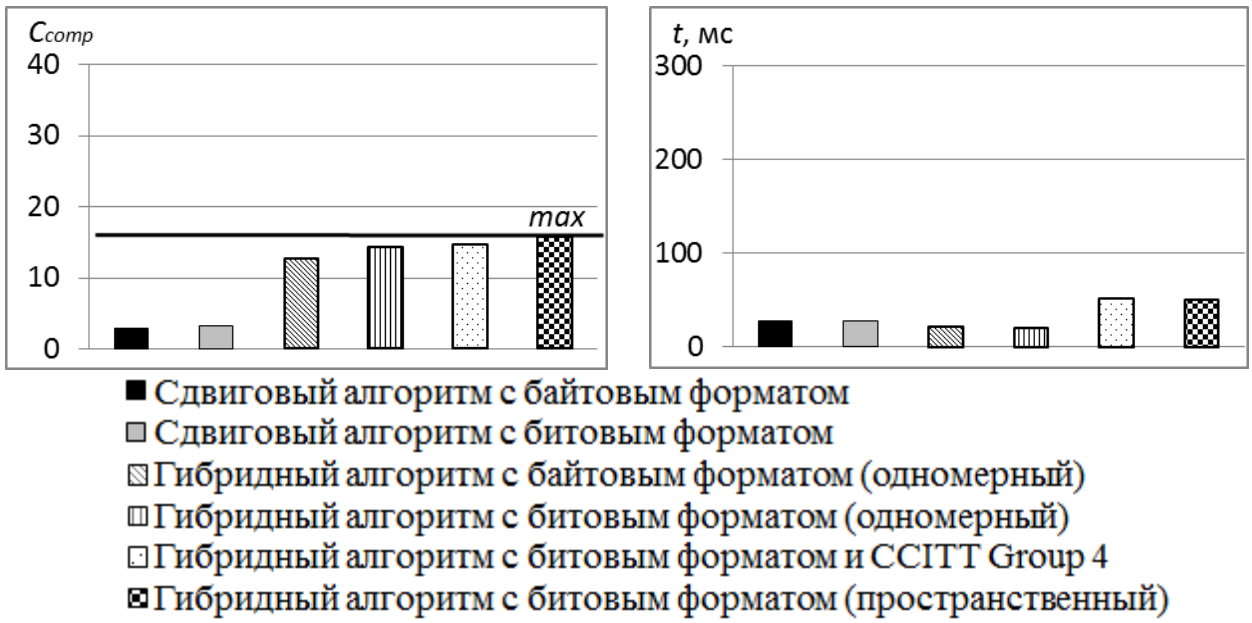


Рисунок 3.3 – Эффективность разработанных алгоритмов (степень и время сжатия) на наборе тестовых данных № 2

Как показали результаты тестирования (рисунок 3.3), использование битового формата как сдвигового, так и гибридного алгоритмов позволяет увеличить степень сжатия при близких временны́х затратах. Сдвиговый алгоритм значительно уступает гибридному алгоритму по степени сжатия. Модификация гибридного алгоритма (6) обеспечила наивысшую степень сжатия на всех тестовых наборах данных при приемлемых временны́х затратах, что позволяет использовать её для сжатия GUI-видеоданных в оперативном режиме. В дальнейшем при упоминании гибридного сдвигово-группового алгоритма речь будет идти именно об этой модификации.

Модификацию гибридного алгоритма (4) целесообразно использовать на платформах с низкой производительностью ввиду её более высокой вычислительной эффективности.

Статистическая значимость различий модификаций гибридного алгоритма (5) и (6) по степени и скорости сжатия подтверждены применением критерия Вилкоксона на уровне значимости 0,01 на всех тестовых наборах данных.

Те же выводы верны и при сравнительном анализе обоих АСПИ, предполагающих выполнение указанных выше реализаций на первом этапе (таблицы 2 – 6 приложения В). На втором этапе этих АСПИ использовались

алгоритмы, рассмотренные в п. 2.1.4 (для второго АСПИ на втором этапе использовалась библиотека zlib, являющаяся реализацией алгоритма стандарта Deflate и включающая как словарное, так и статистическое сжатие). Поэтому в дальнейшем при упоминании АСПИ речь будет идти об алгоритме, использующем модификацию гибридного алгоритма (6).

Обратимся к результатам практического сравнения разработанных и существующих алгоритмов. В тестировании принимали участие реализации 14 алгоритмов, некоторые из них с различными настройками. Протестированы следующие алгоритмы: LZO_X_1, LZO_X_999 с уровнем сжатия 1, 4, 6, 9; zlib (реализация алгоритма стандарта Deflate) с уровнем сжатия 1, 4, 6, 9; гибридный сдвигово-групповой алгоритм, АСПИ, использующий LZO_X_999 с уровнем сжатия 6, 9; АСПИ, использующий zlib с уровнем сжатия 6, 9; реализация FastAC арифметического сжатия [70]; реализация Main Concept JPEG 2000 [86]; реализации Microsoft алгоритмов JPEG, JPEG Lossless; реализация CharLS алгоритма JPEG_LS [58], реализация LEAD Technologies алгоритма CCITT Group 4 [55]. Также в тестировании принимала участие реализация алгоритма стандарта Deflate от Microsoft. Для алгоритмов семейства LZO приведены результаты с учётом финального сжатия алгоритмом Хаффмана.

На рисунке 3.4 приведены результаты тестирования для наборов данных № 1, 3, 5. Для остальных наборов данных (№ 2, 4) и набора данных № 3 соотношение показателей анализируемых алгоритмов близко. Полные результаты тестирования различных реализаций представлены в приложении В в таблице 7.

На рисунке 3.4 представлены данные экспериментальных исследований тех реализаций, которые продемонстрировали показатели близкие к лучшим по выборке:

1. АСПИ, использующий zlib уровня 6 (АСПИ zlib).
2. АСПИ, использующий LZO_X_999 уровня 6 (АСПИ LZO).
3. zlib уровень 9 (zlib 9).
4. zlib уровень 6 (zlib 6).
5. LZO_X_999 уровень 6 (LZO 6).

6. zlib уровень 4 (zlib 4).
7. zlib уровень 1 (zlib 1).
8. LZO_X_1 (LZO_X_1).

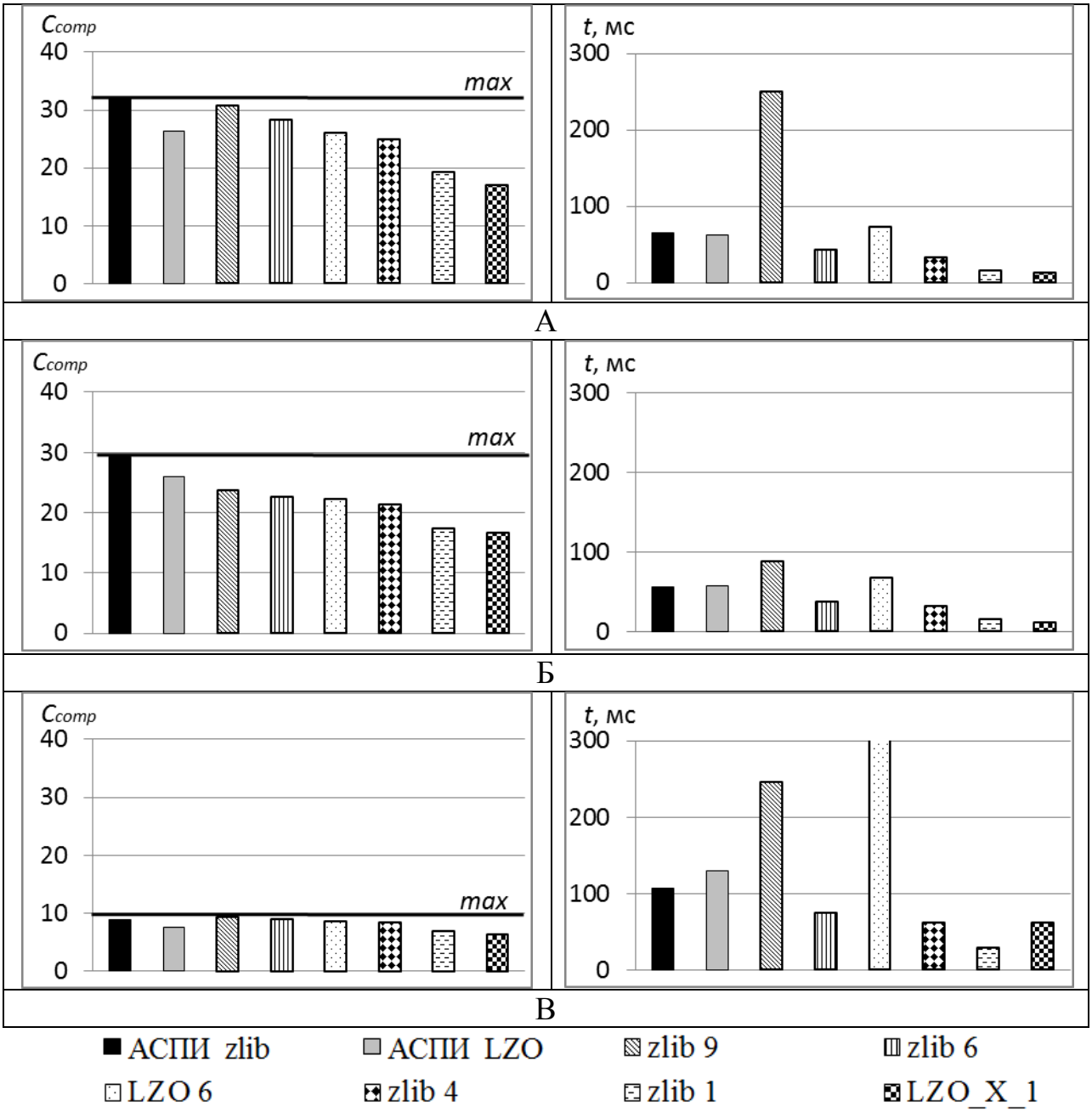


Рисунок 3.4 – Эффективность основных алгоритмов (степень и время сжатия) на наборах тестовых данных: а) № 1; б) № 3; в) № 5

Проведём анализ результатов тестирования, представленных на рисунке 3.4. Время кодирования данных каждой из этих реализаций не превышает 100 мс, что является допустимым временем обработки кадра GUI-видео (раздел 1.1).

АСПИ, использующий zlib уровня 6, обеспечил наивысшую степень сжатия на первых трёх типах изображений при допустимых временны́х затратах. АСПИ, использующий zlib уровня 9, показал близкие результаты по степени сжатия (отличие не является статистически значимым) при более низкой вычислительной эффективности. Сам по себе гибридный сдвигово-групповой алгоритм демонстрирует меньшую степень сжатия по сравнению с АСПИ, построенном на его основе.

Сравним АСПИ, использующий zlib уровня 6, с zlib уровня 9. АСПИ обеспечивает на 23,5 % более высокую степень сжатия для изображений, типичных для Windows XP, и на 6 % более высокую степень сжатия для изображений, типичных для Windows 7, при сходном времени выполнения.
Преимущество АСПИ по степени сжатия для изображений, содержащих текст, составляет 4 %. При этом АСПИ выполняется почти в 4 раза быстрее. При сжатии изображений, содержащих графики и диаграммы, АСПИ демонстрирует на 9,5 % более высокую степень сжатия при близком времени выполнения. Статистическая значимость преимущества АСПИ, использующего zlib уровня 6, по степени сжатия подтверждена применением критерия Вилкоксона на уровне значимости 0,05 для набора данных № 4 и на уровне значимости 0,01 для наборов данных № 1, 2, 3. При сжатии набора данных № 5 zlib уровня 9 продемонстрировал на 5 % более высокую степень сжатия, чем АСПИ, использующий zlib. При этом zlib уровня 9 проигрывает такому АСПИ по вычислительной эффективности более чем вдвое.

Как видно по результатам тестирования, АСПИ, использующий zlib уровня 6, превосходит по степени сжатия АСПИ, использующий LZO и алгоритм Хаффмана на всех тестовых наборах данных, кроме изображений, типичных для Windows 7. Для последнего набора данных оба АСПИ демонстрируют близкий уровень сжатия (разница в степени сжатия не является статистически значимой в соответствии с критерием Вилкоксона).

Алгоритмы JPEG-LS, JPEG 2000, JPEG (в режиме без потерь информации) показали степень сжатия значительно ниже, чем лидеры тестирования, так как эти

алгоритмы ориентированы на сжатие фотографий (непрерывно-тоновой графики), а не дискретно-тоновой графики.

Таким образом, разработанный АСПИ, основанный на гибридном сдвигово-групповом алгоритме и zlib, подтвердил свою эффективность при тестировании. Такой АСПИ позволил значительно увеличить степень сжатия кадров GUI-видео на большинстве тестовых наборов данных по сравнению с алгоритмами семейства LZO и zlib (до 23,5% по сравнению с zlib с уровнем сжатия 9). Удалось увеличить скорость сжатия по сравнению с zlib с уровнем сжатия 9 до 4 раз (при сжатии изображений, содержащих текст). Поэтому представленный АСПИ позволяет выполнить ограничения $C_{comp}(x) \geq C_{min}$, $t(x) \leq T_{max}$ задачи 1.1, и может быть использован на практике для сжатия GUI-видеоданных.

3.2.2. Сжатие промежуточных кадров

Обратимся к результатам экспериментальных исследований существующих и разработанных **алгоритмов оценки движения**. Поскольку не удалось найти реализацию существующего алгоритма оценки движения, адаптированного для обработки GUI-видеоданных (п. 1.3.2), то в тестировании принимала участие реализация, созданная автором этой работы. Эта реализация выполняет поиск подходящего вектора движения по всему кадру. При тестировании выбран размер блока 16 × 16 пикселей, как обеспечивающий сбалансированные показатели по степени сжатия и вычислительной эффективности. При тестировании каждый кадр имел разрешение 1024 × 768 и глубину цвета в 32 бита.

Максимальные отклонения по оси x и y для алгоритмов-участников тестирования (5) и (6) рассчитывались по формуле $n \times blockSize$, где $blockSize$ – размер стороны блока, n – некоторая константа. При этом выбрано значение $n = 3$. Степень сжатия определялась как

$$C_{comp} = B_{total} / (B_{total} - B_{found}),$$

где B_{found} – количество блоков, для которых найдено соответствие данной реализацией, B_{total} – количество блоков в кадре. При выбранном размере кадра и блока $B_{total} = 1024 \times 768 / (16 \times 16) = 3072$.

В тестировании принимали участие следующие реализации:

1. Последовательное выполнение модификации с линейным поиском и проверкой диагонали и модификации с поиском в окрестности блока и проверкой диагонали. Сокращённо – Модификация с линейным поиском → модификация с поиском в окрестности блока.

2. Последовательное выполнение модификации с поиском в окрестности блока и проверкой диагонали и модификации с линейным поиском и проверкой диагонали. Сокращённо – Модификация с поиском в окрестности блока → модификация с линейным поиском.

3. Модификация алгоритма оценки движения, осуществляющая поиск соответствующего блока только по вертикали и по горизонтали. Сокращённо – Модификация с линейным поиском.

4. Модификация с линейным поиском и начальной проверкой диагональных элементов. Сокращённо – Модификация с линейным поиском и проверкой диагонали.

5. Модификация алгоритма оценки движения, осуществляющая поиск соответствующего блока в ближайшей окрестности текущего блока. Сокращённо – Модификация с поиском в окрестности блока.

6. Модификация с поиском в окрестности блока с начальной проверкой диагональных элементов. Сокращённо – Модификация с поиском в окрестности блока и проверкой диагонали.

7. Существующий алгоритм оценки движения, адаптированный для обработки GUI-видеоданных. Сокращённо – Существующий алгоритм.

8. Существующий алгоритм оценки движения, адаптированный для обработки GUI-видеоданных, с начальной проверкой диагональных элементов. Сокращённо – Существующий алгоритм с проверкой диагонали.

Участники тестирования (1) и (2) являются реализациями алгоритма оценки движения с учётом классификационных признаков (п. 2.2.3).

Для тестирования использовались наборы данных, соответствующие типам движения в соответствии с классификацией, приведённой в п. 2.2.3:

1. Движения по вертикали и горизонтали потенциально на большие расстояния. Для возникновения движения такого типа использовался скроллинг.

2. Движения в произвольном направлении на небольшие расстояния. Для возникновения движения этого типа использовалось плавное перетаскивание окна.

Каждый из тестовых наборов данных содержит 10 пар кадров, полученных в ОС Windows 7, Linux (графическая оболочка Ubuntu Gnome 14). Результаты тестирования алгоритмов оценки движения приведены на рисунке 3.5. Отличия между участниками тестирования в степени и скорости сжатия подтверждены на уровне значимости 0,01. Результаты тестирования также представлены в табличном виде в Приложении В в таблицах 12 и 13.

Как правило, GUI-видеоданные требуется сжимать в оперативном режиме (раздел 1.1). Очевидно, что существующий алгоритм оценки движения (7) неприменим для таких целей, так как время обработки одного кадра варьируется в интервале [3,5; 10] с. Время выполнения реализации (6) в значительной степени варьируется в зависимости от типа движения и находится в интервале [47; 510] мс. Быстрее всего выполняется реализация (4), что позволяет использовать её при сжатии GUI-видеоданных в оперативном режиме. Но реализация (4) не в состоянии выявить движения второго типа, поэтому перейдём к рассмотрению реализаций алгоритма с учётом классификационных признаков (1) и (2). Оба этих алгоритма предполагают последовательное выполнение реализаций (4) и (6), но в разном порядке. Время последовательного выполнения реализаций (4) и (6) (в любом порядке) при обоих типах движения оказывается ниже, чем сумма времени выполнения отдельно (4) и (6) реализации. Это объясняется тем, что соответствие для части блоков находит первая из выполняемых реализаций, уменьшая тем самым размер входных данных для второй реализации.

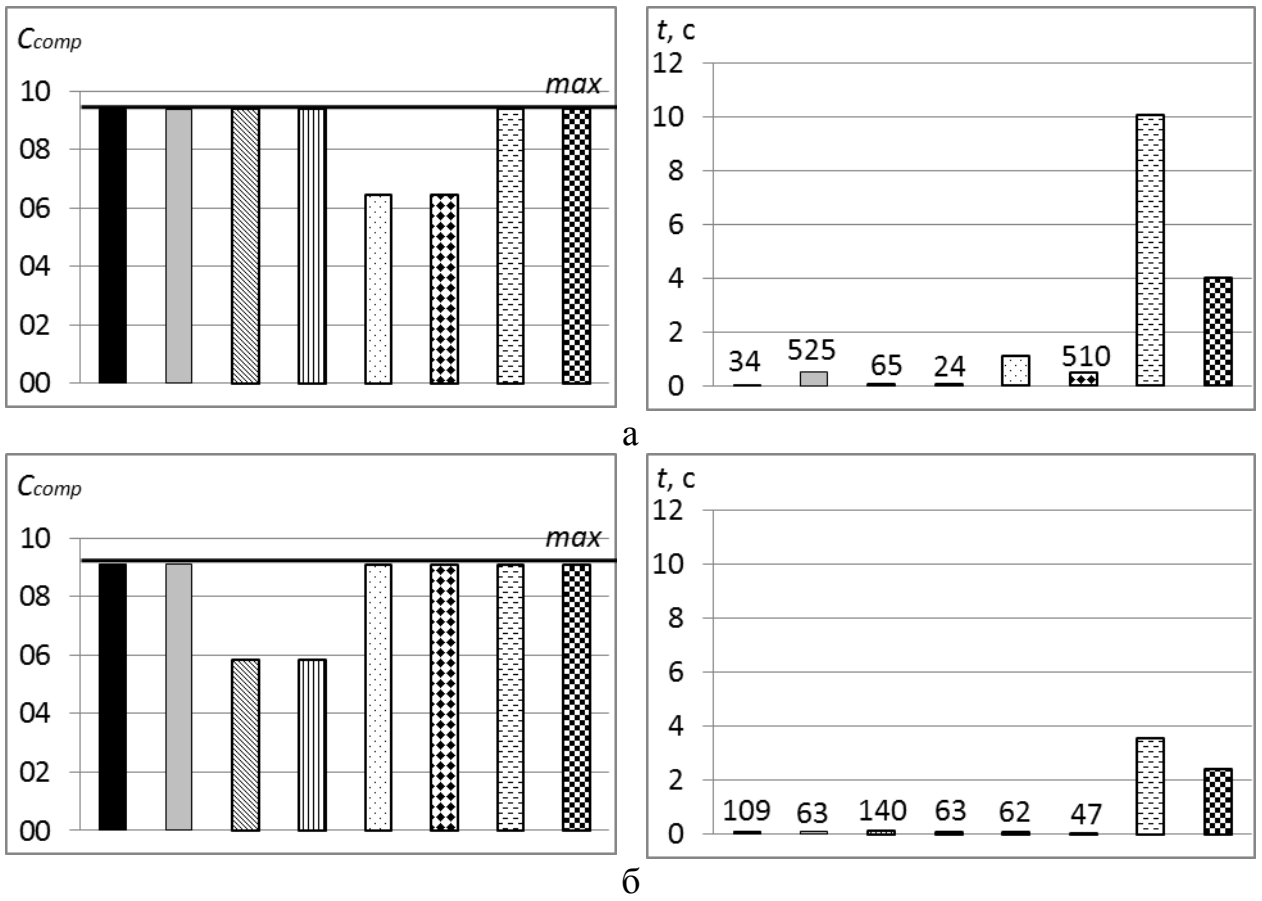


Рисунок 3.5 – Эффективность алгоритмов оценки движения (степень и время сжатия) на наборах тестовых данных: а) № 1; б) № 2

Такой подход даёт наилучшие результаты, когда первой выполняется реализация, созданная для данного типа движения. Например, при скроллинге время выполнения алгоритма с учётом классификационных признаков (1) значительно меньше, чем алгоритма с учётом классификационных признаков (2). Если учитывать время выполнения при обоих типах движения, то наилучшим по этому критерию является алгоритм с учётом классификационных признаков (1). Время его выполнения более стабильно при различных типах движения и не превышает 109 мс. Таким образом, при записи 10 кадров в секунду это время

близко к показателю, обеспечивающему сжатие в оперативном режиме. При этом алгоритм с учётом классификационных признаков находит соответствие для эквивалентного количества блоков, что и существующий алгоритм (7).

Как видно по результатам тестирования, начальная проверка диагональных элементов уменьшает время выполнения оценки движения для всех рассмотренных реализаций.

Учитывая всё вышеизложенное, можно сделать вывод, что разработанный алгоритм оценки движения с учётом классификационных признаков позволяет выполнить ограничения $C_{comp}(x) \geq C_{min}$, $t(x) \leq T_{max}$ задачи 1.1.

Обратимся к результатам экспериментальных исследований **алгоритмов отсечения неизменившихся областей кадра**. Каждый из тестовых наборов данных содержит 10 пар кадров, которые сравнивались для отсечения неизменившихся областей. Для тестирования использованы следующие наборы данных, воспроизводящие те из типичных изменений в GUI-видеоданных, для которых проводилось аналитическое сравнение алгоритмов отсечения неизменившихся областей (п. 2.2.1):

1. Изменение диагональных пикселей изображения. Для получения этого типа изменений использовалось перемещение окна по направлению, близкому к диагональному. Сокращённо – Изменение диагональных пикселей.

2. Изменение области, близкой по форме к прямоугольной. Для получения этого типа изменения использовалось сворачивание и восстановление свёрнутого окна. Сокращённо – Изменение прямоугольной области.

Эти наборы данных получены в ОС Windows 7, Linux (графическая оболочка Ubuntu Gnome 14) и доступны по ссылке [43]. В тестировании принимали участие реализации следующих алгоритмов:

1. Алгоритм отсечения неизменившихся строк и столбцов. Сокращённо – Алгоритм отсечения линий.

2. Алгоритм отсечения неизменившихся блоков. Сокращённо – Алгоритм отсечения блоков.

3. Адаптивный алгоритм отсечения неизменившихся областей кадра. Сокращённо – Алгоритм адаптивного отсечения.

Усреднённые результаты тестирования этих алгоритмов представлены на рисунке 3.6. Отличия между участниками тестирования в степени и скорости сжатия подтверждены на уровне значимости 0,01. Для алгоритма, выявляющего изменившиеся блоки, выбран размер блока 8 × 8 пикселей, так как при таком размере блока на большинстве тестов достигается максимальная степень сжатия с учётом количества передаваемых номеров блоков.

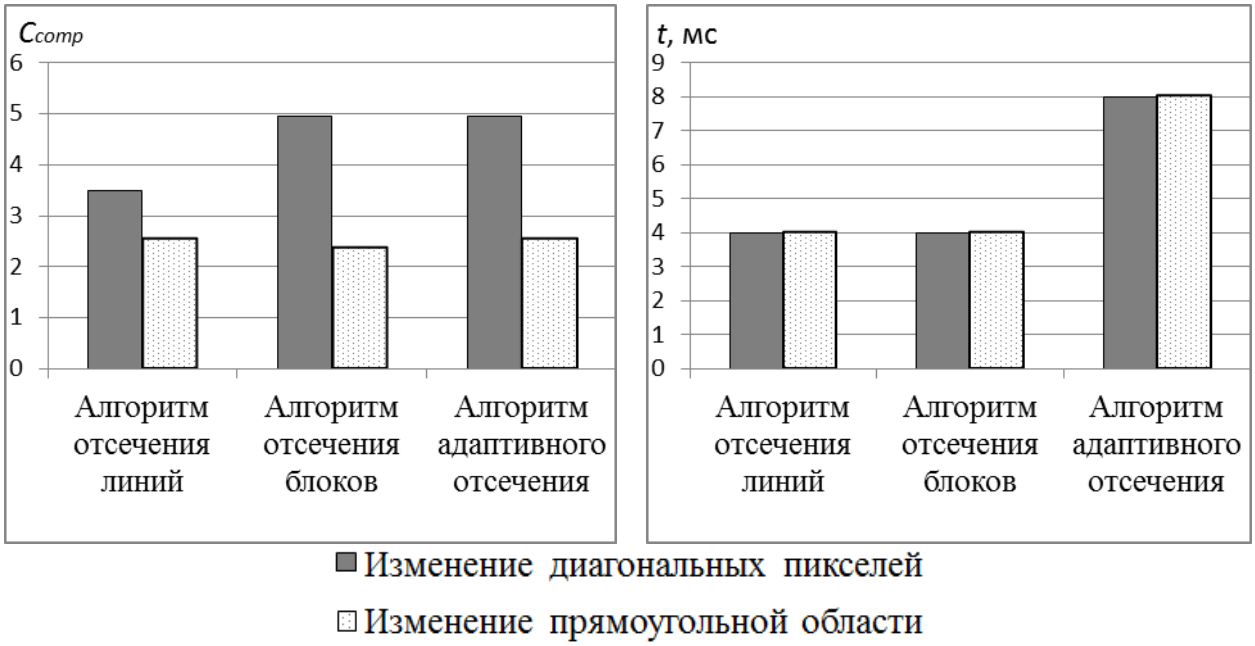


Рисунок 3.6 – Эффективность алгоритмов отсечения неизменившихся областей кадра (степень и время сжатия)

Поскольку сжатие промежуточных кадров проводится в несколько этапов, имеет смысл рассмотреть результаты тестирования всей последовательности обработки промежуточных кадров, представленные на рисунке 3.7. Такая последовательность обработки кадров включает следующие алгоритмы:

1. Один из приведённых выше алгоритмов отсечения неизменившихся областей кадра.

2. Алгоритм оценки движения с учётом классификационных признаков (реализация № 7 в списке, приведённом выше).

3. АСПИ, использующий zlib уровня 6.

Как видно по результатам тестирования (рисунок 3.6), алгоритм отсечения линий демонстрирует более высокую степень сжатия, чем алгоритм отсечения блоков на наборе тестовых данных (2), в то время как на (1) наборе данных алгоритм отсечения блоков демонстрирует более высокую степень сжатия, чем алгоритм отсечения линий. Эти алгоритмы имеют близкую вычислительную эффективность. Такой результат подтверждает предварительные выводы, сделанные в п. 2.2.1.

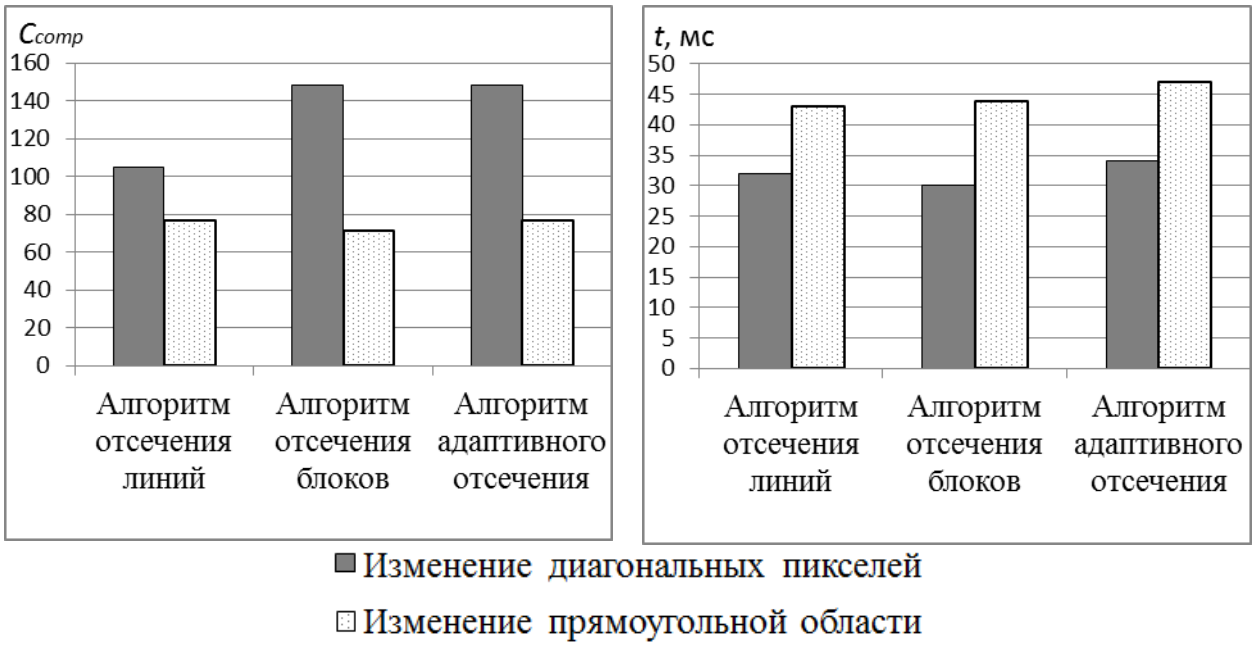


Рисунок 3.7 – Эффективность последовательности алгоритмов обработки промежуточных кадров (степень и время сжатия)

Адаптивный алгоритм работает несколько медленнее прочих за счёт выполнения и блочного и линейного сравнения, но на всех тестовых наборах данных способен выбрать конкретный алгоритм отсечения неизменившихся областей кадра, обеспечивающий более высокую степень сжатия. Высокая скорость сжатия кадра (< 10 мс) адаптивным алгоритмом позволяет обрабатывать кадры GUI-видео по мере их фиксации (рисунок 3.6). Те же выводы верны и для всей последовательности обработки промежуточных кадров (рисунок 3.7) Таким образом, результаты тестирования подтверждают предварительные выводы, сделанные в п. 2.2.2 о характеристиках адаптивного алгоритма отсечения неизменившихся областей кадра.

Результаты тестирования алгоритмов отсечения неизменившихся областей и последовательности алгоритмов обработки промежуточных кадров также представлены в Приложении В в таблицах 10 и 11.

3.3. Выводы

1. Предложен способ формирования вспомогательных структур данных, позволяющих повысить вычислительную эффективность кодирования и декодирования данных гибридным сдвигово-групповым алгоритмом. К таким структурам данных относятся битовый массив, где каждый бит соответствует пикселю исходного изображения, и хэш-таблица, ключом которой при кодировании является цвет, а значением – номер последнего просмотренного пикселя с таким цветом. При декодировании ключом хэш-таблицы является номер последнего просмотренного пикселя с таким цветом, а значением – цвет.

2. Предложен способ формирования служебных данных, записываемых гибридным сдвигово-групповым алгоритмом в результирующий массив, учитывающий частоту, с которой встречаются группы пикселей различных типов. Отмечается, что битовый формат позволяет повысить степень сжатия по сравнению с байтовым форматом при частой встречаемости одиночных пикселей.

3. Проведён сравнительный анализ разработанных и существующих алгоритмов сжатия дискретно-тоновых изображений. Отмечается превосходство разработанного алгоритма со сниженной пространственной избыточностью над существующими алгоритмами по степени сжатия. В то же время алгоритм со сниженной пространственной избыточностью демонстрирует высокие показатели вычислительной эффективности.

4. Представлены результаты практического сравнения алгоритмов отсечения неизменившихся областей кадра. Показано, что на всех тестовых наборах данных адаптивный алгоритм способен выбрать конкретный алгоритм отсечения неизменившихся областей кадра, обеспечивающий более высокую степень сжатия.

5. Представлены результаты практического сравнения разработанного и существующих алгоритмов оценки движения. Показано, что разработанный алгоритм оценки движения с учётом классификационных признаков позволяет ускорить выполнение оценки движения до десяти раз при эквивалентном количестве распознанных движений определённых типов.

ГЛАВА 4. ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОДЕКА ДЛЯ СЖАТИЯ GUI-ВИДЕОДАННЫХ

Описываются программные средства кодека, предназначенного для преобразования GUI-видеоданных, в том числе модуль, использующий вычислительные ресурсы видеокарты. Приводятся результаты практического сравнения разработанного кодека с существующими кодеками аналогичного назначения.

4.1. Особенности программной реализации

В дальнейшем в этой главе под *кодеком* будем понимать именно кодек, предназначенный для преобразования GUI-видеоданных. ПО кодека создавалось с учётом требований, выдвинутых к кодекам сжатия GUI-видеоданных в п. 2.3.1.

4.1.1. Среда разработки программного обеспечения

В качестве языка разработки выбран C++, так как этот язык обеспечивает высокую гибкость при работе с оперативной памятью, что немаловажно при разработке ПО, которое должно производить высокопроизводительную обработку данных в оперативном режиме (п. 2.3.2). В качестве интегрированной среды разработки выбрана Microsoft Visual C++ 9.0 [104], так как включает все необходимые инструменты для разработки ПО для ОС Windows.

При выборе компилятора рассматривались следующие альтернативы: Microsoft Visual C++ Compiler 9.0 [104] и Intel C++ Compiler 11 [85]. Компилятор фирмы Intel генерирует ассемблерный код, работающий быстрее, чем ассемблерный код, генерируемый компилятором фирмы Microsoft, но только для процессоров Intel [80, 85]. Поскольку ПО разрабатываемой системы должно функционировать на аппаратных платформах, оснащённых процессорами различных производителей, выбран компилятор Microsoft Visual C++ Compiler 9.0.

4.1.2. Архитектура кодека

На основе описанных в предыдущих разделах алгоритмов автором разработан кодек Butterfly Screen Video Codec, характеризующийся оригинальной архитектурой (рисунок 4.1), соответствующей обобщённой архитектуре кодека сжатия GUI-видеоданных (п. 2.3.2).

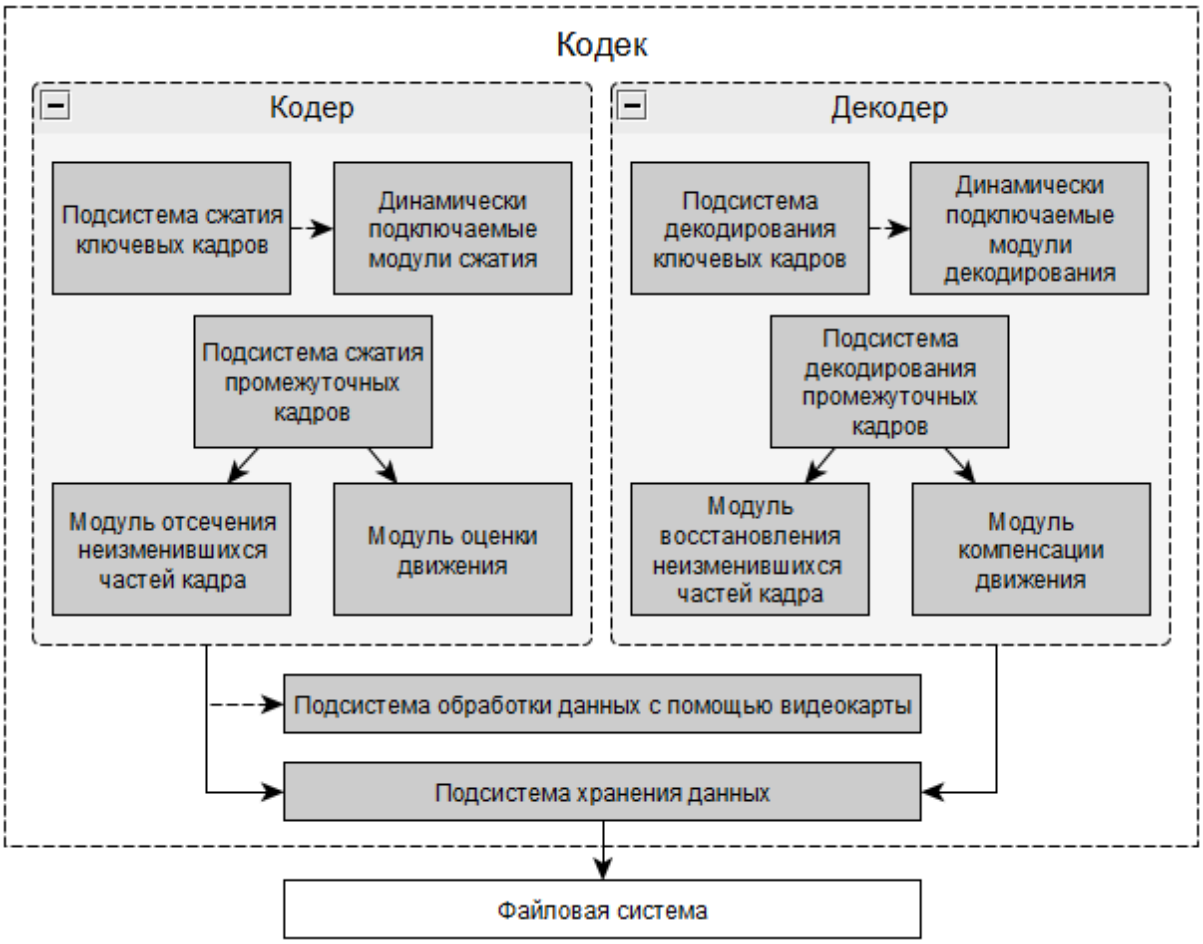


Рисунок 4.1 – Архитектура кодека

Блоки с серым и белым фоном соответствуют подсистемам, входящим и не входящим в состав ПО кодека, соответственно.

Подсистема обработки данных с помощью видеокарты. Эта подсистема предназначена для выполнения отдельных этапов сжатия на видеокарте, что позволяет высвободить ресурсы ЦП для других задач. Подсистема подключается в ходе работы кодера, только если аппаратная платформа оснащена видеокартой, поддерживающей технологию Nvidia CUDA (п. 1.4.1), поэтому на рисунке 4.1 к

этой подсистеме ведёт пунктирная линия. Более подробно эта подсистема рассмотрена в разделе 4.2.

Подсистема хранения данных. Данная подсистема предназначена для сохранения, чтения закодированных данных, поиска кадров по номеру. В качестве основного хранилища данных используется файл, содержащий закодированные видеоданные. Подсистема поддерживает также чтение и запись отдельных кадров в ОП.

Подсистемы сжатия и декодирования ключевых кадров. Данные подсистемы предназначены для обработки ключевых кадров. Содержат реализацию кодера и декодера, входящих в состав АСПИ (п. 2.1.4), соответственно. Предлагается модуль, реализующий второй этап АСПИ, сделать динамически подключаемым (поэтому на рисунке 4.1 к нему ведёт пунктирная линия). Этот модуль используется только при достаточном уровне производительности аппаратной платформы.

Подсистема сжатия промежуточных кадров. Данная подсистема координирует работу *Модуля отсечения неизменившихся частей кадра* и *Модуля оценки движения*. *М*одуль отсечения неизменившихся частей кадра реализует адаптивный алгоритм отсечения неизменившихся областей в кадре (п. 2.2.2). Модуль оценки движения реализует алгоритм оценки движения с учётом классификационных признаков (п. 2.2.3). Эта подсистема использует *Подсистему обработки данных с помощью видеокарты* для выполнения отсечения неизменившихся частей кадра на видеокарте.

Подсистема декодирования промежуточных кадров. Данная подсистема координирует работу *Модуля восстановления неизменившихся частей кадра* и *Модуля компенсации движения*. Эти модули реализуют декодеры, соответствующие реализациям алгоритмов сжатия, содержащихся в описанных выше модулях *Подсистемы сжатия промежуточных кадров*.

Теперь рассмотрим структуру программных средств кодера и декодера **на уровне классов**. На рисунках 4.2 и 4.3 представлены основные классы кодера и декодера соответственно.

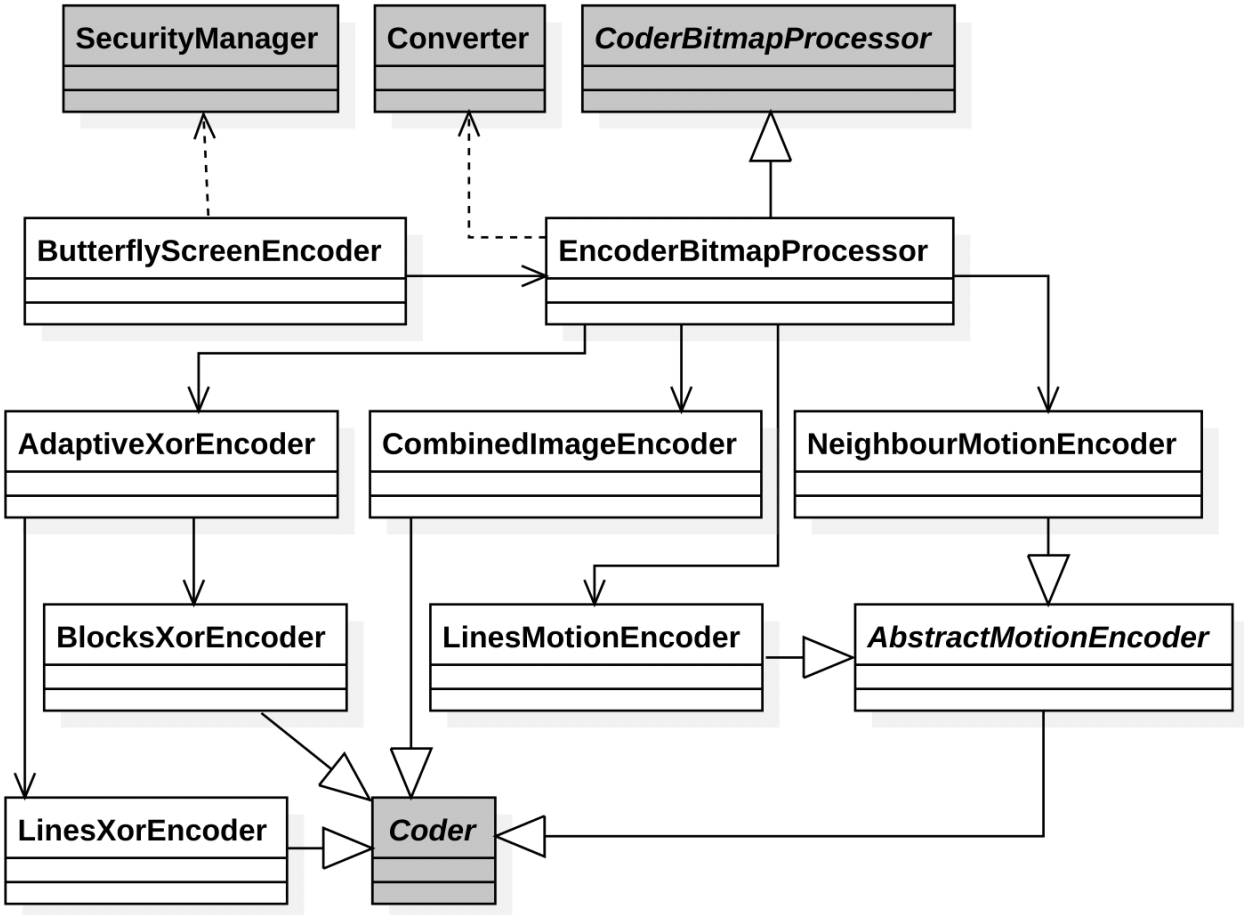


Рисунок 4.2 – Диаграмма основных классов кодера

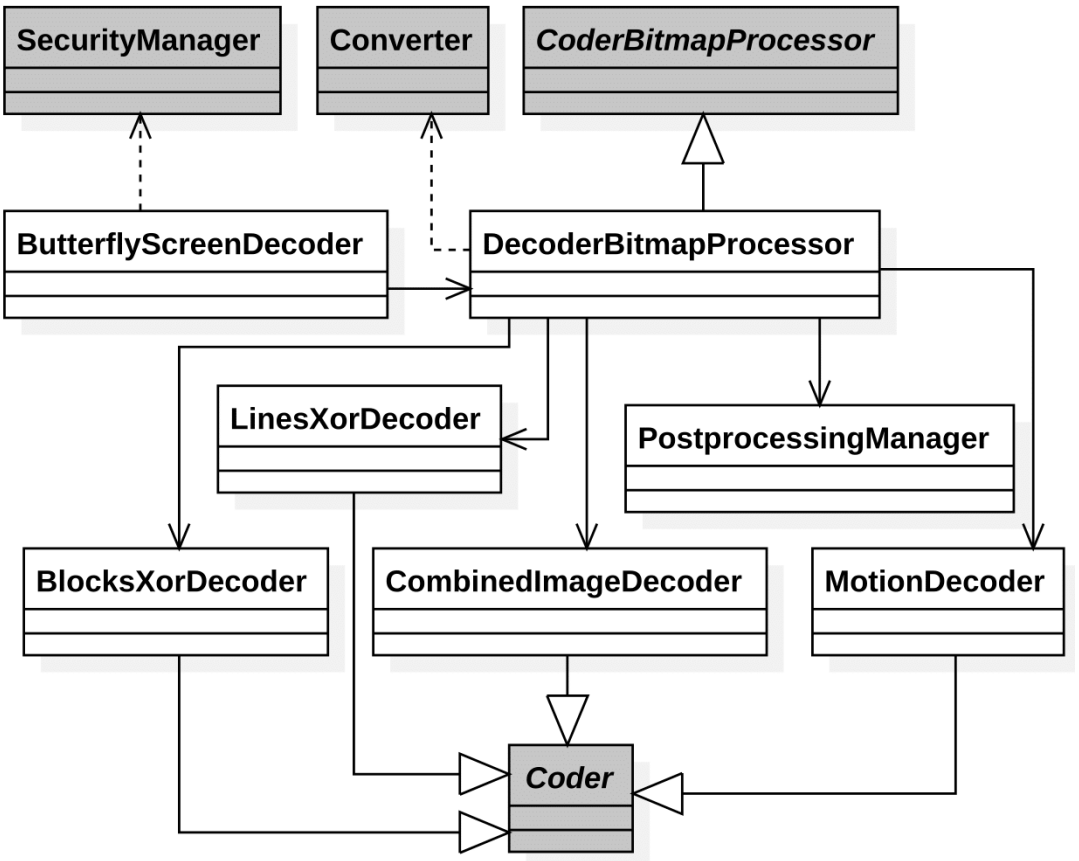


Рисунок 4.3 – Диаграмма основных классов декодера

Серым цветом закрашены классы из разделяемой библиотеки CodecShared, входящей в состав кодека. Эта библиотека используется как в кодере, так и в декодере. Рассмотрим основные классы библиотеки CodecShared.

Coder. Это абстрактный класс, являющийся базовым для большинства классов, выполняющих кодирование и декодирование.

SecurityManager. Класс, реализующий политику лицензирования.

Converter. Класс, содержащий набор методов для преобразования типов данных.

CoderBitmapProcessor. Абстрактный класс, являющийся базовым для классов, координирующих процесс кодирования и декодирования.

Рассмотрим основные классы кодера (рисунок 4.2).

ButterflyScreenEncoder. Класс, предоставляющий объектный интерфейс к кодеру.

EncoderBitmapProcessor. Класс, координирующий процесс кодирования.

AdaptiveXorEncoder. Класс, реализующий адаптивный алгоритм отсечения неизменившихся областей кадра (п. 2.2.2).

BlocksXorEncoder. Класс, реализующий алгоритм отсечения неизменившихся блоков кадра (раздел 1.3).

LinesXorEncoder. Класс, реализующий алгоритм отсечения неизменившихся строк и столбцов кадра (п. 2.2.1).

CombinedImageEncoder. Класс, реализующий кодер АСПИ (п. 2.1.4).

AbstractMotionEncoder. Абстрактный класс, являющийся базовым для классов, осуществляющих оценку движения.

LinesMotionEncoder. Класс, реализующий алгоритм оценки движения, осуществляющий поиск по вертикали и по горизонтали от текущего блока.

NeighbourMotionEncoder. Класс, реализующий алгоритм оценки движения, осуществляющий поиск в ближайшей окрестности текущего блока.

Рассмотрим основные классы декодера (рисунок 4.3).

ButterflyScreenDecoder. Класс, предоставляющий объектный интерфейс к декодеру.

DecoderBitmapProcessor. Класс, координирующий процесс декодирования.

BlocksXorDecoder. Класс, реализующий алгоритм восстановления неизменившихся блоков кадра.

LinesXorDecoder. Класс, реализующий алгоритм восстановления неизменившихся строк и столбцов кадра (п. 2.2.1).

CombinedImageDecoder. Класс, реализующий декодер АСПИ (п. 2.1.4)

MotionDecoder. Класс, реализующий алгоритм компенсации движения.

PostprocessingManager. Класс, предназначенный для проведения постобработки декодированного кадра. В частности этот класс производит наложение рисунка курсора мыши на декодированный кадр.

4.1.3. Программный интерфейс кодека

Кодек предоставляет объектный и функциональный интерфейс (рисунки 4.4 и 4.5 соответственно). Функциональный интерфейс позволяет использовать кодек в приложениях, написанных на языке C. Объектный интерфейс предназначен для использования в приложениях, написанных на объектно-ориентированных языках, например на C++. Оба интерфейса обеспечивают доступ к идентичному набору возможностей кодека.

Отличие функционального интерфейса заключается в наличии дополнительных методов для создания и уничтожения кодера и декодера (например, BSC_CreateEncoderInstance(), BSC_DestroyDecoderInstance()). Во всех остальных функциях, входящих в состав функционального интерфейса, присутствует дополнительный параметр по сравнению с объектным интерфейсом – указатель на созданный ранее кодер или декодер (HANDLE a_handle).

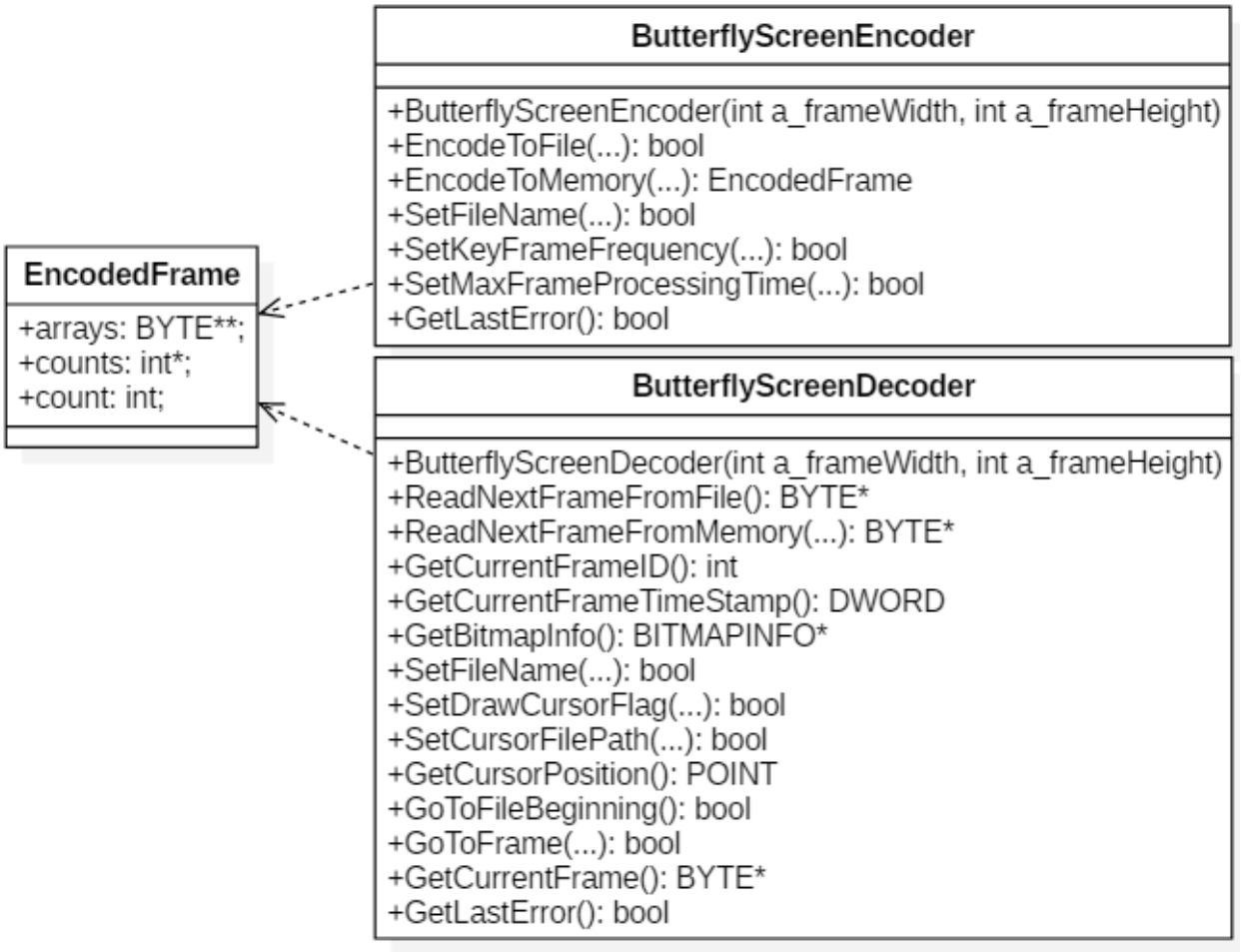


Рисунок 4.4 – Объектный интерфейс кодека

Интерфейсы кодера позволяют записывать закодированный кадр в указанный файл или в оперативную память. Возможна настройка максимального времени обработки кадра, а также частоты, с которой в результирующий видеопоток будут записываться ключевые кадры. Интерфейсы декодера позволяют считывать закодированный файл из указанного файла либо из оперативной памяти, выполнять поиск кадра по его порядковому номеру, получать текущую позицию курсора и настраивать способ его отображения. Более полная информация об интерфейсе кодека приведена в Приложении Г.

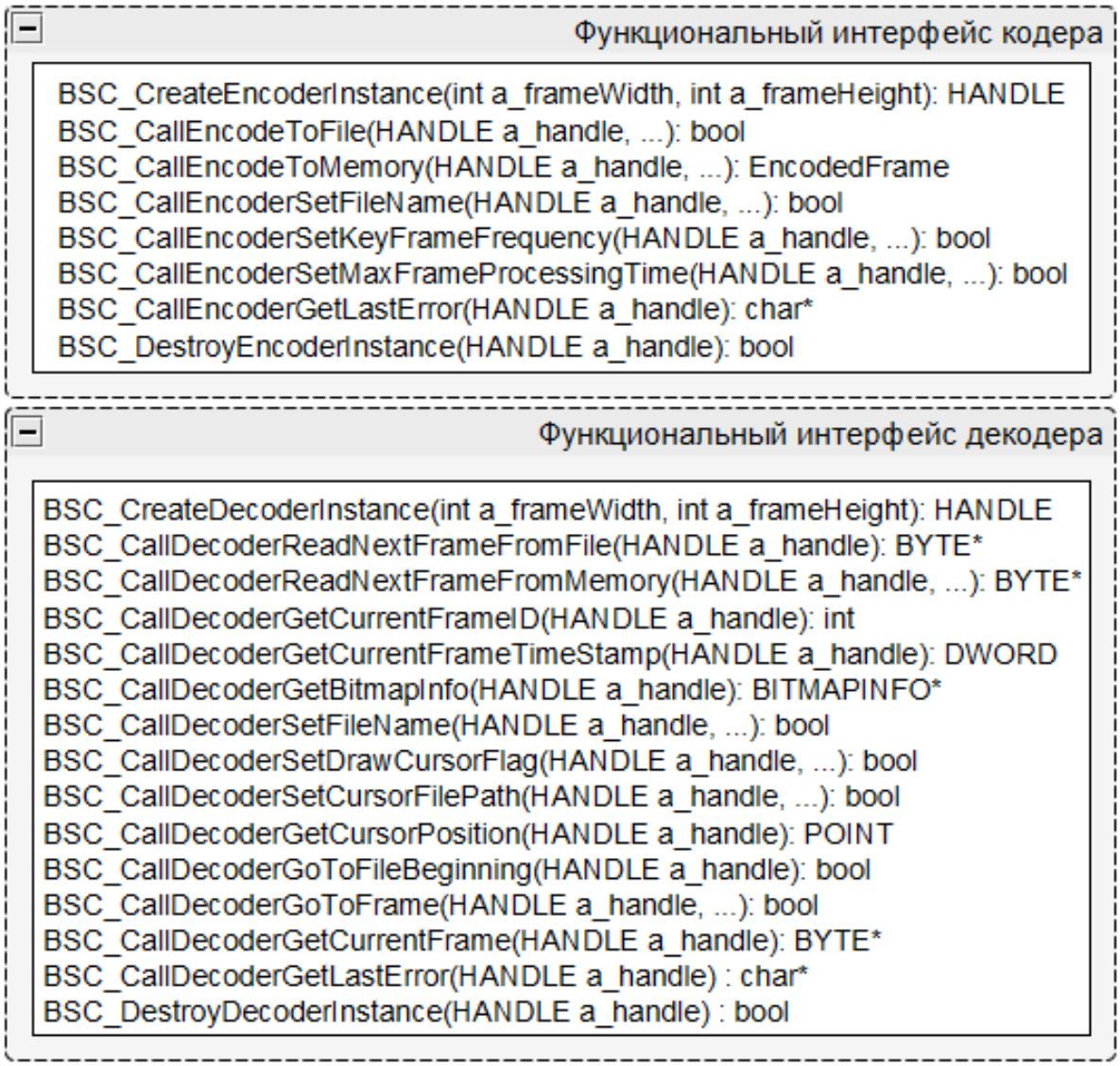


Рисунок 4.5 – Функциональный интерфейс кодека

4.1.4. Технология обработки данных

Рассмотрим технологию кодирования, реализованную в кодеке (рисунок 4.6). Сначала происходит проверка, является ли текущий кадр ключевым или нет. Если кадр оказался ключевым, то он сжимается гибридным сдвигово-групповым алгоритмом (первый этап АСПИ, п. 2.1.4). Затем рассчитывается время t_{cur}, прошедшее с момента начала кодирования кадра. Если $t_{cur} < t_{max}$, где t_{max} – максимальное время обработки кадра (задаётся вызовом функции BSC_CallEncoderSetMaxFrameProcessingTime), выполняется второй этап АСПИ (zlib).

Если кадр оказался промежуточным, то сначала происходит выявление изменившихся относительно предыдущего кадра областей текущего кадра с помощью адаптивного алгоритма (п. 2.2.2). Если подключена подсистема обработки данных с помощью видеокарты (рисунок 4.1), этот этап сжатия выполняется на видеокарте, в противном случае на центральном процессоре.

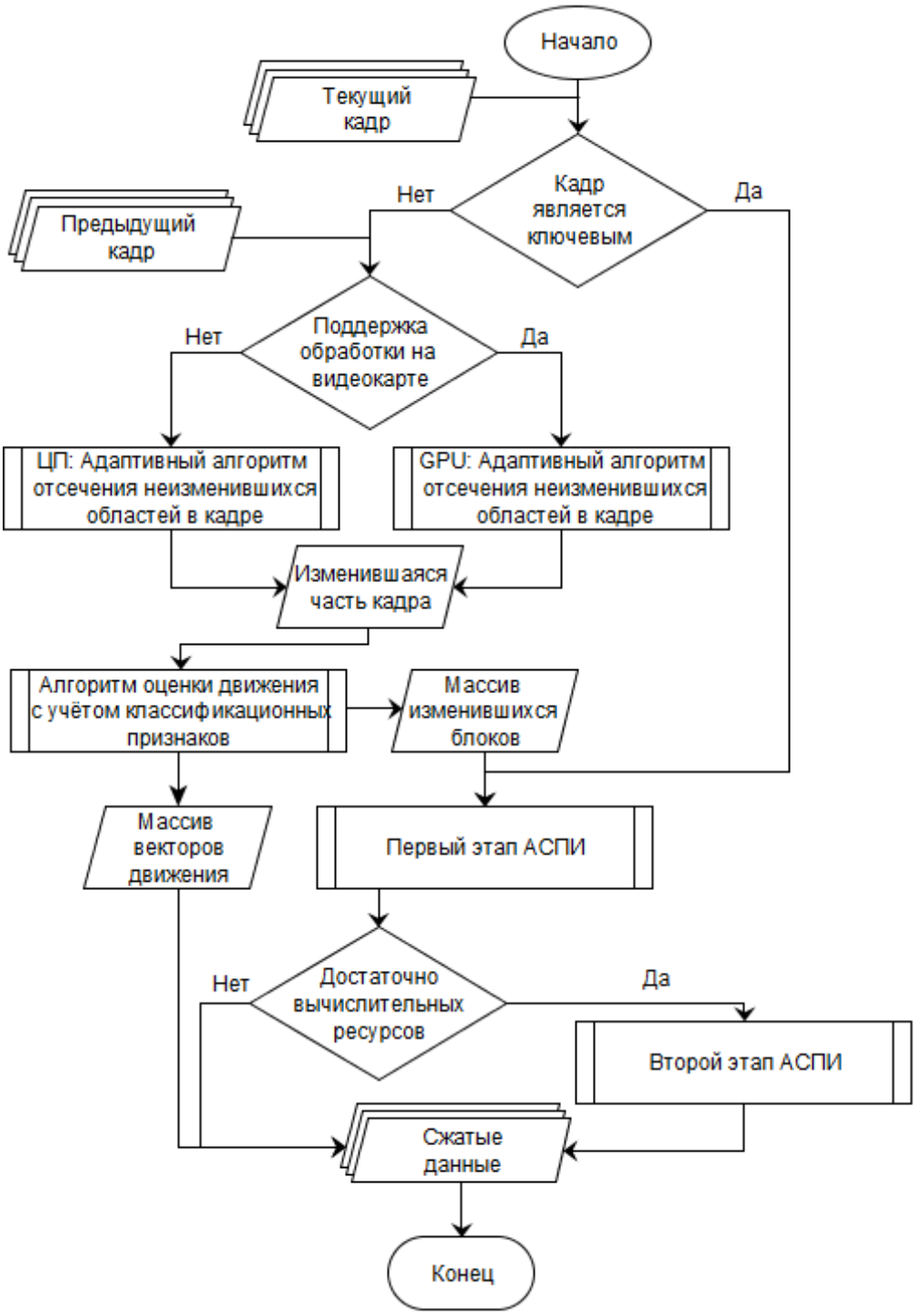


Рисунок 4.6 – Технология кодирования данных кодека

В ходе выполнения этого этапа в результирующий массив записываются идентификаторы изменившихся областей кадра. Затем выполняется оценка движения для изменившихся блоков текущего кадра относительно предыдущего кадра при помощи алгоритма оценки движения с учётом классификационных признаков (п. 2.2.3). При этом размер изменившейся части кадра может уменьшиться в случае, если для некоторых блоков найдено соответствие в ходе оценки движения. В ходе выполнения этого этапа в результирующий массив записываются вектора движения для блоков, для которых найдено соответствие.

После этого промежуточный кадр сжимается аналогично ключевому кадру за тем исключением, что АСПИ обрабатывает только те области текущего кадра, для которых не было найдено соответствие на предыдущих этапах.

Стоит отметить, что разработанный кодек сжимает данные без потерь информации за счёт того, что все входящие в его состав алгоритмы обладают этим свойством. Напомним, что сжатие данных без потерь информации является одним из требований, предъявляемых к кодекам для сжатия GUI-видеоданных (п. 2.3.1).

На рисунке 4.7 представлена технология декодирования, реализованная в кодеке. Сначала сжатые данные обрабатываются декодером АСПИ, использующим на втором этапе zlib. Если текущий кадр является ключевым, то на этом декодирование заканчивается. Если текущий кадр является промежуточным, то происходит компенсация движения, а затем восстановление части кадра, неизменившейся относительно предыдущего.

4.2. Подсистема высокопроизводительной обработки данных с использованием видеокарты

Как отмечено в п. 1.4.1, использование вычислительных ресурсов видеокарты в ряде случаев позволяет ускорить выполнение ресурсоёмких вычислений и одновременно высвободить ЦП для выполнения других задач. В пп. 4.2.1 и 4.2.2 представлены адаптированные для выполнения на видеокарте

реализации некоторых из представленных в работе алгоритмов. При этом использовались пиксельные шейдеры и технология Nvidia CUDA. В п. 4.3 представлены результаты практического сравнения реализаций, использующих вычислительные ресурсы видеокарты, и ЦП-реализаций этих алгоритмов.

В этом разделе все размеры результирующих массивов указаны при ширине и высоте исходных текстур 1024 и 768 пикселей соответственно.

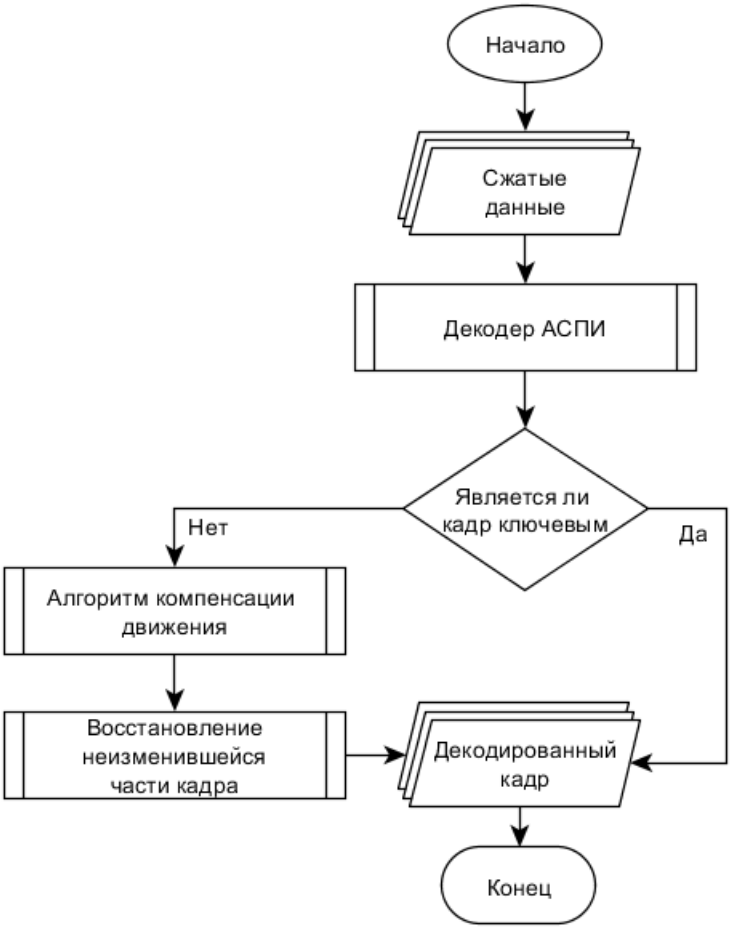


Рисунок 4.7 – Технология декодирования данных кодека

4.2.1. Реализация линейного и блочного сравнения изображений с помощью пиксельных шейдеров и Nvidia CUDA

В созданной реализации с помощью видеокарты выполняется сравнение промежуточных кадров с ключевым кадром с целью выявления совпадающих и различающихся элементов.

Реализация линейного сравнения изображений с помощью пиксельных шейдеров. Входные данные этой операции (назовём её *линейное исключающее ИЛИ*) – 2 изображения одинакового размера. Результирующие данные – набор элементов, количество которых равно количеству пикселей во входном изображении. Каждый такой элемент является индикатором равенства или неравенства цветов двух пикселей, поэтому в идеале это – 1 бит.

Каждый элемент, который является индикатором равенства или неравенства цветов двух пикселей, при использовании пиксельных шейдеров занимает 1 байт. Однобитовый и однобайтовый форматы не поддерживаются видеокартой, которая использовалась при тестировании, поэтому используется 2-байтовый формат D3DFMT_R5G6B5 [61]. Но применена следующая техника: каждый байт в 2-байтовом значении цвета пикселя результирующей текстуры используется независимо для кодирования результата операции исключающего ИЛИ. Это достигается за счёт независимого использования каналов цвета R и B пикселя результирующей текстуры. Таким образом, размер результирующей текстуры составляет $1024 \times 768 = 786432$ байтов.

Реализация блочного сравнения изображений с помощью пиксельных шейдеров. Назовём эту операцию *блочное исключающее ИЛИ*. Входные данные те же, что и при линейном исключающем ИЛИ. Существенное отличие состоит в значительно меньшем (в $2^6 = 8 \times 8$ раз) размере результирующего массива, так как каждый его элемент является индикатором равенства или неравенства двух блоков пикселей. В идеале 1 элемент результирующего массива – 1 бит.

Формат результирующей текстуры выбран тот же, что и при линейном исключающем ИЛИ – D3DFMT_R5G6B5. Поскольку независимое использование каждого байта в этом двухбайтовом формате привело к замедлению в работе алгоритма, предлагается не использовать эту технику при блочном исключающем ИЛИ. Тем более что размер результирующей текстуры и так невелик ($1024 \times 768 \times 2 / 64 = 24576$ байтов).

Реализация линейного исключающего ИЛИ с помощью CUDA. Для хранения результата выполнения функции, реализующей операцию

исключающего ИЛИ, использован однобитовый формат как обеспечивающий минимальный размер результирующих данных. При этом один поток обрабатывает 8 пар подряд идущих пикселей и сохраняет результат в одном байте результирующего массива. Использование однобитового формата этого массива позволило уменьшить время его копирования из видеопамяти в оперативную память, так как в этом случае размер результирующего массива в 8 раз меньше, чем при использовании однобайтового формата, и составляет 1024 × 768 / 8 = 98304 байтов.

Для уменьшения количества вычислений программа, выполняемая на видеокарте, оперирует с цветом пикселя исходной текстуры, как со значением типа unsigned int. Условный переход является одной из самых дорогих операций при выполнении на видеокарте (п. 1.4.1), поэтому реализация операции линейного исключающего ИЛИ не использует условных переходов. Для этого сначала вычисляется разница *difference* между максимальным и минимальным из двух рассматриваемых значений цветов пикселей, а затем текущий бит в байте, который будет результатом выполнения потока, вычисляется следующим образом:

$$output = output \mid (difference \mathbin{\&\&} 1) \times byteMask,$$

где | – битовое ИЛИ, «&&» – логическое И.

Результатом выполнения операции (*difference* && 1) будет 1, если $difference \neq 0$, и 0, если $difference = 0$. В маске *byteMask* единице равен только один бит, соответствующий текущему биту в байте результата. Таким образом, если цвета сравниваемых пикселей равны ($difference = 0$), то текущий бит в байте результата *output* останется равным 0, иначе будет установлен в 1.

Реализация блочного исключающего ИЛИ с помощью CUDA. Для реализации блочного исключающего ИЛИ применены все те же способы оптимизации, что и при линейном исключающего ИЛИ. Но здесь появляются условные переходы. Они необходимы, чтобы остановить линейное сравнение при обнаружении первой пары неравных пикселей в блоке. Существенная экономия времени здесь достигается, если все потоки в warp (п. 1.4.1) вышли из цикла

проверки пикселей блока досрочно. Иначе такие потоки вынуждены будут ожидать завершения выполнения проверки остальными потоками.

Размер результирующего массива составляет 1024 × 768 / 8 / 64 = 1536 байта.

4.2.2. Алгоритм классификации блоков изображений, оптимизированный для выполнения на видеокарте

Разработана следующая модификация алгоритма классификации блоков изображения (п. 1.2.4), адаптированная для обработки GUI-видеоданных. Алгоритм относит блок к непрерывно-тоновому типу, если для блока одновременно выполнены 2 условия:

$(|R_1 - R_2| < CompDiff_{max})$ && $(|G_1 - G_2| < CompDiff_{max})$ (У. 1)
&& $(|B_1 - B_2| < CompDiff_{max})$

$|R_1 + G_1 + B_1 - R_2 - G_2 - B_2| < Diff_{max}$, (У. 2)

Где «&&» – логическое И; R_1, G_1, B_1 – компоненты текущего пикселя; R_2, G_2, B_2 – компоненты соседнего пикселя; $CompDiff_{max}$, $Diff_{max}$ – заранее заданные константы.

Иначе блок считается дискретно-тоновым. Таким образом, решение о дискретно-тоновой природе блока может быть принято не только при резких перепадах значения отдельной компоненты цвета, но и при изменении одновременно всех компонент цвета. При этом возможна ситуация, когда величина изменения каждой компоненты цвета будет ниже порога $CompDiff_{max}$.

Входными данными является изображение в формате RGB. Результирующими данными является массив элементов, при этом количество элементов равно количеству блоков изображения. Если элемент равен 0, то соответствующий блок дискретно-тоновый; иначе – непрерывно-тоновый.

Описание алгоритма: в цикле от левого верхнего угла блока к правому нижнему происходит проверка условий (У. 1) и (У. 2) для соседних пикселей, при этом ближайшие по диагонали пиксели не считаются соседними. Пусть текущий столбец i, текущая строка j, а **array** – двумерный массив пикселей, тогда условия (У. 1) и (У. 2) будут проверяться для *array*[j, i] и *array*[j, i + 1], а также для *array*[j,

i] и *array*[*j* + 1, *i*]. Если хотя бы для одной пары соседних пикселей блока не выполняется условие (У. 1) или условие (У. 2), то такой блок считается дискретно-тоновым. Если для всех пар соседних пикселей блока выполнены условия (У. 1) и (У. 2), то такой блок считается непрерывно-тоновым.

Если в одном блоке встречаются как дискретно-тоновые, так и непрерывно-тоновые подобласти, то такой блок считается дискретно-тоновым, чтобы не допустить потерь информации в дискретно-тоновых областях. Разработанная модификация алгоритма классификация блоков изображения обладает трудоёмкостью $O(n)$ и характеризуется высокой вычислительной эффективностью за счёт использования сравнительно быстрых операций «+», «–».

Рассмотрим детали реализации разработанной модификации алгоритма классификации блоков изображения, оптимизированной для выполнения на видеокарте. Впервые эта реализация представлена автором в [12].

Для осуществления доступа к вычислительным ресурсам видеокарты использована технология Nvidia CUDA. При этом для реализации алгоритма классификации выбран уровень CUDA Runtime, как обеспечивающий оптимальный баланс предоставляемых возможностей и простоты разработки [109].

Для хранения результирующего массива использован однобайтовый формат. Один поток обрабатывает один блок входного изображения и сохраняет результат в одном байте результирующего массива.

Условный переход является одной из самых дорогих операций при выполнении на видеокарте, поэтому реализация алгоритма классификации блоков изображения не использует условные переходы. Для этого используется следующая техника. Пусть в массиве **array** хранятся цвета пикселей, а *index1* и *index2* – индексы *R*-компонент цвета двух соседних пикселей, для которых должны быть осуществлены проверки, *continuousBlockFlag* – результат выполнения классификации для одного блока, $CompDiff_{max}$ – максимальная разница компонент цвета соседних пикселей.

continuousBlockFlag = *continuousBlockFlag* | (Abs(*array*[*l_index1*] – *array*[*l_index2*]) – $CompDiff_{max}$ > 0),

где «|» – битовое ИЛИ, «Abs» – операция взятия абсолютного значения.

Сначала вычисляется разность по модулю R-компонент двух соседних пикселей. Если эта разность по модулю меньше, чем $CompDiff_{max}$, то результатом сравнения с нулём будет 0, иначе 1. Значит, если условие непрерывно-тонового блока выполнено, то эта запись равносильна

continuousBlockFlag = *continuousBlockFlag* | 0,

иначе

continuousBlockFlag = *continuousBlockFlag* | 1.

Аналогичным образом осуществляются все проверки условий непрерывно-тонового блока.

4.2.3. Результаты экспериментальных исследований

При тестировании каждый кадр имел разрешение 1024 × 768 и глубину цвета в 32 бита. Таким образом, размер исходного изображения составляет 1024 × 768 × 4 байтов. Тестирование проводилось на платформе со следующими характеристиками: процессор Intel Core 2 Duo E6750 2,66 ГГц, ОП DDR2 2Гб, видеокарта Nvidia GeForce 8600 GTS (подключена через интерфейс PCI Express), ОС Windows 7. В целях повышения статистической достоверности результатов исследований каждый эксперимент повторялся 100 раз, в качестве оценки использовалось среднее значение, полученное по результатам каждого из экспериментов, а доверительная вероятность при этом принята равной 95%.

Обратимся к результатам экспериментальных исследований, полученных для реализаций **алгоритмов отсечения неизменившихся областей кадра**. Ниже приведены результаты тестирования временны́х показателей реализаций, использующих пиксельные шейдеры и технологию Nvidia CUDA. Для тестирования использовались те же наборы данных, что и в п. 3.2.2. При обработке промежуточного кадра, следующего после ключевого кадра,

необходимо копировать в видеопамять оба этих кадра. При обработке прочих промежуточных кадров достаточно скопировать в видеопамять один кадр, потому что предыдущий промежуточный кадр уже скопирован в видеопамять ранее. Поэтому приведено время выполнения операции исключающего ИЛИ, как при копировании в видеопамять двух кадров, так и при копировании одного кадра. Из видеопамяти в ОП всегда копируется 1 результирующий массив. Интересно также сравнить скорость выполнения операции исключающего ИЛИ без учёта времени копирования входных и результирующих данных. Для алгоритма, выполняемого на ЦП, указывается только время выполнения, так как в этом случае все данные находятся в ОП, и их не нужно копировать в видеопамять и обратно.

Поскольку соотношение результатов, продемонстрированных различными реализациями при проведении операций блочного и линейного исключающего ИЛИ, близко (таблица 4.1), то можно провести общий анализ результатов тестирования. Реализация, использующая пиксельный шейдер, выполняется быстрее, чем реализация, использующая технологию CUDA, но с учётом копирования данных в видеопамять и обратно CUDA-реализация оказывается более быстрой. Это достигается за счёт того, что технология CUDA обеспечивает более эффективную работу с памятью, нежели могут обеспечить пиксельные шейдеры. Пиксельные шейдеры в 2 – 3 раза проигрывают технологии Nvidia CUDA по времени копирования данных из ОП в видеопамять, что является важным параметром при реализации алгоритмов сжатия данных, поэтому пиксельные шейдеры нецелесообразно применять для реализации алгоритмов сжатия данных. В дальнейшем данные о реализациях, использующих пиксельные шейдеры, не приводятся.

В ходе тестирования установлено, что использование условных переходов для выхода из цикла сравнения пикселей при обнаружении первой пары неравных пикселей в блоке ускоряет выполнение операции блочного исключающего ИЛИ CUDA-реализацией в среднем на 10 % по сравнению с CUDA-реализацией, не использующей условные переходы.

Таблица 4.1 – Результаты тестирования линейного и блочного исключающего ИЛИ

№	Технология / параметр	Алгоритм	Время выполнения при копировании двух кадров, мс	Время выполнения при копировании одного кадра, мс	Время выполнения без учёта времени копирования, мс
1	Пиксельные шейдеры	Линейное исключающее ИЛИ	13	6	< 1 (менее 1 мс)
		Блочное исключающее ИЛИ	15	8	< 1 (менее 1 мс)
2	Nvidia CUDA	Линейное исключающее ИЛИ	6	4	2
		Блочное исключающее ИЛИ	5	4	2
3	ЦП	Линейное исключающее ИЛИ			4
		Блочное исключающее ИЛИ			4

Суммарное время выполнения пиксельного шейдера и копирования данных значительно превышает время выполнения ЦП-реализации даже при копировании в видеопамять одного кадра. По совокупному времени исполнения CUDA-реализация при копировании одного кадра в видеопамять показала те же результаты, что и ЦП-реализация. При этом удаётся высвободить ЦП для выполнения других операций, что немаловажно при сжатии GUI-видеоданных, так как приложение, фиксирующее и сжимающее GUI-видеоданные зачастую должно функционировать в фоновом режиме (раздел 1.1).

Обратимся к результатам экспериментальных исследований, полученных для реализаций **алгоритма классификации блоков изображения**. В ходе тестирования использовались те же наборы данных, что и в п. 3.2.1. В случае видеокарты следует разделять накладные расходы, связанные с копированием данных из ОП в видеопамять и обратно и, собственно, выполнение функции. Эти показатели, полученные экспериментальным путём при размере блока 16 × 16 пикселей, приведены в таблице 4.2. При размере блока 8 × 8 пикселей все рассматриваемые показатели оказались очень близки к показателям, полученным при размере блока 16 × 16 пикселей.

Таблица 4.2 – Классификация блоков изображения

Время выполнения на ЦП, мс	**Время выполнения на видеокарте**		
	Время копирования из ОП в видеопамять, мс	**Время выполнения, мс**	**Время копирования из видеопамяти в ОП, мс**
6	2,2	6	0,035

Как видно по результатам тестирования, время выполнения классификации блоков на ЦП и время непосредственного выполнения на видеокарте совпадают. В ходе тестирования также установлено, что техника замены условных переходов на операцию присваивания с вычислением результирующего значения позволяет ускорить выполнение классификации блоков на видеокарте с 7 мс до 6,4 мс, т. е. на 9 %. Таким образом, выполнение классификации блоков на видеокарте позволяет высвободить ЦП для выполнения других операций.

Разработанная модификация алгоритма классификации блоков изображения успешно сопоставляет блоку изображения, являющегося кадром GUI-видео, дискретно-тоновый или непрерывно-тоновый тип. Так значительная часть блоков фонового рисунка распознана, как непрерывно-тоновые, а все блоки, содержащие текст или иные резкие цветовые переходы распознаны, как дискретно-тоновые на всех тестовых наборах данных.

4.3. Результаты практического сравнения кодеков

В разделе представлены результаты экспериментальных исследований разработанного кодека Butterfly Screen Video Codec с другими кодеками.

При тестировании кодеков применялась следующая методика. Для получения идентичной последовательности действий пользователя при тестировании различных кодеков, для каждого набора видеоданных производилась запись действий пользователя с помощью приложения MacroExpress 3 [96]. В ходе тестирования в Диспетчере задач Windows замерялись параметры «Память – частный рабочий набор» и «Время ЦП» для процесса, выполняющего фиксацию и сжатие GUI-видеоданных. Параметр «Память –

частный рабочий набор» замерялся каждую минуту проигрывания сценария с усреднением полученных результатов. Размер сжатых данных вычислялся в расчёте на одну минуту видеозаписи.

Таким образом формировались таблицы, каждой строке которой соответствовал кодек, а каждому столбцу – сценарий. После этого результаты тестирования по каждому набору данных усреднялись для каждого алгоритма. Степень загруженности ЦП другими процессами, исполняемыми в ОС, и средний уровень использования ими ОП были аналогичны во всех экспериментах. Проверка статистической значимости отличий в степени сжатия и уровне использования системных ресурсов различных реализаций алгоритмов проводилась с помощью критерия Вилкоксона, а доверительная вероятность при этом принята равной 95%.

Для тестирования использовались видеоданные, полученные при взаимодействии пользователя с различными приложениями:

1. Прикладные приложения (Eclipse [66], ДубльГИС [21], Microsoft Word [102], Microsoft PowerPoint [102], FireFox [71] и др.).

2. Системные приложения (список файлов и папок в разных видах, панель управления, диспетчер устройств и др.).

Каждый набор данных содержит 10 сценариев взаимодействия пользователя с приложениями, записанных в ОС Windows 7, 8. Каждый такой сценарий имеет продолжительность 10 минут. Кадры видео в первом тестовом наборе имеют размер 1920 × 1080 пикселей, во втором наборе – 1280 × 1024 пикселей. Тестирование проводилось на платформе со следующими характеристиками: процессор Intel Core 2 Quad CPU Q9500 2,83 GHz; ОП DDR3 4Gb, ОС Windows 7.

В тестировании принимали участие 12 кодеков, некоторые из которых с различными настройками: Butterfly Screen Video Codec; BB FlashBack 4.1.5 [49], H.264 без потерь и с уровнем потерь 20 %, входящий в состав ПО для фиксации и локального сохранения GUI-видео Bandicam [48]; CamStudio Lossless Codec v1.5 [53] в режимах LZO и GZIB; Microsoft Video 1 [103]; Cinepak [57]; MSU Screen Capture Lossless Codec v1.2 [108] в режимах «with Fades detection» и «without

Fades detection»; Microsoft Expression Encoder 4 [101] в режимах «30 kbps» и «10 kbps»; Windows Media Encoder 9 [140]; Google VP8 Video Codec [74] в режимах VBR и CBR; inno Screen Capture Codec 1.9 [84]; TechSmith Screen Codec 2 [127].

На рисунке 4.8 приведены результаты тестирования тех кодеков, предназначенных для сжатия GUI-видеоданных, которые продемонстрировали показатели близкие к лучшим по выборке для обоих наборов данных:

1. Butterfly Screen Video Codec. Сокращённо – BSVC.
2. H.264 (без потерь). Сокращённо – H.264.
3. CamStudio Lossless Codec v1.5 (GZIB). Сокращённо – CamStudio Ls (GZIP).
4. MSU Screen Capture Lossless Codec v1.2 (without Fades detection). Сокращённо – MSU Codec.
5. Microsoft Expression Encoder 4 (30 kbps). Сокращённо – Expr. Encoder (30 kbps).
6. Microsoft Expression Encoder 4 (10 kbps). Сокращённо – Expr. Encoder (10 kbps).
7. Windows Media Encoder 9. Сокращённо – Media Encoder.
8. Google VP8 Video Codec (VBR, 5000 kbps). Сокращённо – Google Codec (VBR).
9. inno Screen Capture Codec 1.9. Сокращённо – inno Codec.

Результаты тестирования различных реализаций также представлены в Приложении В в таблицах 8 и 9. Впервые результаты практического сравнения кодека Butterfly Screen Video Codec с другими кодеками представлены в [14].

Степень загруженности ЦП на рисунке 4.8 приведена в расчёте на одно ядро процессора. Поэтому значения свыше 100 означают, что процесс в среднем использовал более одного ядра процессора.

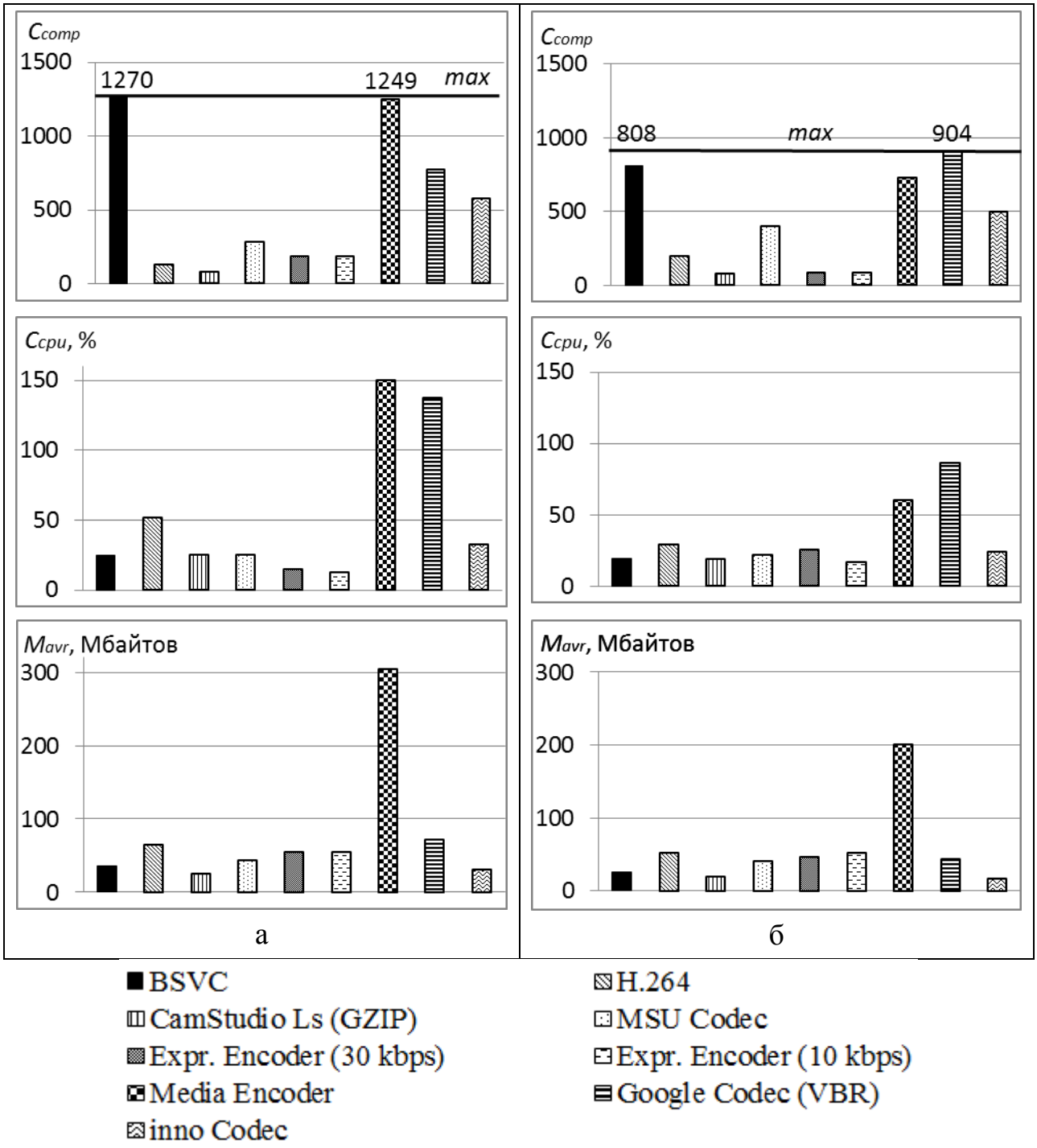


Рисунок 4.8 – Эффективность основных кодеков (степень сжатия и загруженности ЦП, средний уровень использования ОП) на наборах тестовых данных: а) № 1; б) № 2

Для всех кодеков, для каких это возможно, выбран режим сжатия без потерь информации. Для остальных кодеков (участники тестирования под номерами 5, 6 и 8) выбраны настройки, обеспечивающие минимальные искажения.

BSVC и Media Encoder обеспечили наивысшую степень сжатия первого набора видеоданных. Уровень использования ресурсов ЦП у BSVC ниже, чем у

Media Encoder в 6 раз на этом наборе видеоданных. Степень сжатия второго набора видеоданных, продемонстрированная BSVC, чуть ниже (на 11 %), чем у Google VP8 Video Codec (VBR). При этом уровень использования ресурсов ЦП у представленного кодека ниже, чем у Google VP8 Video Codec (VBR) в 4,3 раза. Уровень использования ОП кодеком BSVC близок к минимальному по выборке на всех тестовых наборах данных.

На основании вышеизложенного следует сделать вывод о том, что разработанный кодек, занимая лидирующие позиции по степени сжатия и среднему уровню использования ОП, демонстрирует значительно более низкую степень загруженности ЦП, и по совокупности характеристик превосходит аналоги на всех тестовых наборах видеоданных. Разработанный кодек осуществляет высокопроизводительную обработку данных с высокой степенью сжатия и обладает высокой ресурсоэффективностью, что соответствует требованиям, предъявляемым к кодекам сжатия GUI-видеоданных (п. 2.3.1), а также условию $\min_x C_{cpu}(x)$ и ограничениям $C_{comp}(x) \geq C_{min}$, $M_{avr}(x) \leq M_{max}$ задачи 1.1.

Проведённые экспериментальные исследования позволяют сделать вывод о том, что разработанные алгоритмы могут применяться не только в приложениях для фиксации и локального сохранения GUI-видео, но и в приложениях для мультимедийного общения в режиме демонстрации экрана компьютера собеседнику. При этом битрейт должен быть не ниже 500 Кбит/с при разрешении кадров 1280 × 1024.

4.4. Выводы

1. Предложена оригинальная архитектура кодека обработки GUI-видеоданных, отвечающая требованиям к программным средствам таких систем, выдвинутым выше. Такая архитектура предполагает наличие групп подсистем кодера и декодера, взаимодействующих с независимыми подсистемами как

входящими в состав кодека, так и программными средствами других производителей.

2. Разработан объектный и функциональный интерфейсы кодека, обеспечивающие доступ к идентичному набору возможностей кодека. Наличие таких интерфейсов кодека позволяет упростить его использование в приложениях, написанных как на функциональных, так и на объектно-ориентированных языках.

3. Представлены реализации алгоритма классификации блоков изображения и реализации алгоритмов отсечения неизменившихся областей в кадре, использующие технологию Nvidia CUDA для доступа к ресурсам видеокарты. Эти реализации по уровню вычислительной эффективности схожи с реализациями, использующими только вычислительные ресурсы ЦП. При этом удаётся высвободить ЦП для выполнения других операций, что немаловажно при сжатии GUI-видеоданных.

4. Представлены результаты экспериментальных исследований разработанного кодека в сравнении с существующими аналогами. Показано, что разработанный кодек по совокупности характеристик превосходит аналоги на всех тестовых наборах видеоданных. Так, разработанный кодек обеспечивает степень сжатия, близкую к максимальной по выборке при существенно меньшем уровне загруженности ЦП (до 6 раз ниже).

ЗАКЛЮЧЕНИЕ

В ходе выполнения диссертационной работы получены следующие основные научные и практические результаты:

1. Проведён анализ существующего алгоритмического и программного обеспечения сжатия видеоданных, особенностей традиционного видео и GUI-видео. Показано, что требуется значительная модификация существующих, либо разработка новых алгоритмов для сжатия GUI-видеоданных, обладающих более высокими показателями эффективности, позволяющими производить сжатие GUI-видеоданных без потерь информации не только с высокой степенью сжатия, но и с минимальным использованием вычислительных ресурсов компьютера.

2. Разработано семейство алгоритмов сжатия промежуточных кадров GUI-видео, включающее алгоритм оценки движения с учётом классификационных признаков, использующий информацию о том, что одни типы движений встречаются в GUI-видео чаще, чем другие. Предложенный алгоритм оценки движения имеет более низкую трудоёмкость и позволяет ускорить выполнение оценки движения до 10-ти раз при эквивалентном количестве распознанных движений определённых типов.

3. Разработано семейство алгоритмов сжатия ключевых кадров GUI-видео, включающее сдвиговый алгоритм, алгоритм пространственного группового кодирования, гибридный сдвигово-групповой алгоритм, а также алгоритм сжатия со сниженной пространственной избыточностью.

4. Проведены комплексные экспериментальные исследования с варьированием параметров в обширных пределах на широком спектре GUI-видеоданных, позволившие подтвердить эффективность разработанных алгоритмов.

5. Предложены концептуальные основы создания ПО кодека сжатия GUI-видеоданных, включающие обобщённую архитектуру такого кодека, упрощающие и ускоряющие создание аналогичных систем. Представленная

архитектура кодека характеризуется наличием подсистемы обработки данных на видеокарте и динамическим подключением модулей сжатия.

6. Разработан высокопроизводительный кодек для сжатия GUI-видеоданных, ядро которого основано на разработанных алгоритмах и результатах исследований. Тестирование показало, что этот кодек по совокупности показателей превосходит аналоги на широком наборе тестовых видеоданных.

7. Созданное алгоритмическое и программное обеспечение кодека сжатия GUI-видеоданных, основанное на результатах диссертационного исследования, внедрено в Югорском НИИ информационных технологий в качестве компонента пользовательского приложения для записи GUI-видео, а также в компании-разработчике ПО ООО «Армадэйт» в качестве самостоятельного программного продукта.

Разработанные алгоритмы могут быть применены не только в приложениях для фиксации и локального сохранения GUI-видео, но и в приложениях для мультимедийного общения в режиме демонстрации экрана компьютера собеседнику. Планируется в дальнейшем провести исследования с целью адаптации разработанных алгоритмов для использования в приложениях мультимедийного общения, в частности минимизации битрейта в потоке закодированных данных.

СПИСОК УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

C_{max} – максимальная степень сжатия данных

C_{min} – минимальная степень сжатия данных

C_{comp} – степень сжатия данных

C_{cpu} – степень загруженности центрального процессора

E_c – вычислительная эффективность

M_{avr} – средний уровень использования оперативной памяти

GPGPU (General-Purpose computation on Graphics Processing Units) – универсальные вычисления на видеокарте

GUI-видеоданные – видеоданные графического интерфейса пользователя (Graphical User Interface видеоданные). В англоязычной литературе можно встретить также похожие по семантике термины «screen video», «screen content».

RGB (Red, Green, Blue – красный, зелёный, синий) – цветовая модель, в которой пиксель представлен тремя компонентами (R-, G-, B-компоненты)

АСПИ – алгоритм со сниженной пространственной избыточностью

ОП – оперативная память

ОС – операционная система

ПО – программное обеспечение

ПК – персональный компьютер

ЦП – центральный процессор

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Бабкин В.Ф., Книжный И.М., Хрекин К.Е. Сжатие многоспектральных изображений для задач дистанционного зондирования земли из космоса // Современные проблемы дистанционного зондирования Земли из космоса. 2004. – В.1. Т.1. – С. 330-332
2. Беляев Е.А. Управление параметрами алгоритма сжатия видеоинформации при передаче данных в системах мобильной связи. : диссертация к.т.н.: 05.13.01. – СПб., 2008. – 172 с.
3. Большев Л.Н., Смирнов Н.В. Таблицы математической статистики. – М.: Наука, 1983. – 416 с.
4. Ватолин Д. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео./ Д. Ватолин и др. – М. : Диалог-МИФИ, 2003. – 384 с.
5. Дворкович А., Мингазов И. Методика построения промежуточных кадров видеопоследовательности. [Электронный ресурс] URL: http://www.autex.spb.ru/cgi-bin/download.cgi?dspa2004_1_48, (дата обращения 25.06.2013).
6. Дружинин Д.В., Замятин А.В. Адаптивный алгоритм сжатия видеоданных графического интерфейса пользователя // Информационные технологии и математическое моделирование (ИТММ-2016): материалы XV Международной конференции имени А. Ф. Терпугова (12 – 16 сентября 2016 г.) – 2016. – Т. 2. – С. 96–102.
7. Дружинин Д.В., Замятин А.В. Концептуальные основы создания программного обеспечения кодека для сжатия видеоданных графического интерфейса пользователя // Информационные технологии и математическое моделирование (ИТММ-2017): материалы XVI Международной конференции имени А.Ф. Терпугова (29 сентября – 3 октября 2017 г.) – 2017. – Т. 2. – С. 34–40.
8. Дружинин Д.В. Гибридный алгоритм сжатия изображения. Сравнение алгоритмов сжатия изображений // Информационные технологии и

математическое моделирование: Материалы VI Международной научно-практической конференции (9 – 10 ноября 2007 г.). – 2007 – Т. 2. – С. 70–73.

9. Дружинин Д.В. Модификации гибридного алгоритма сжатия изображений // Обратные задачи и информационные технологии рационального природопользования : Материалы IV научно-практической конференции. – 2008. – С. 218–222.
10. Дружинин Д.В. Комбинированный алгоритм сжатия ключевых кадров экранного видео // Вестник Томского государственного университета. Управление, вычислительная техника и информатика: Научный журнал. – 2011. – № 3(16). – С. 67–77.
11. Дружинин Д.В. Алгоритм оценки движения, адаптированный для обработки экранного видео // Вычислительные методы и программирование : Научный журнал. – 2009. Т. 10, № 2. – С. 228–233.
12. Дружинин Д.В. Алгоритм классификации блоков изображения, оптимизированный для выполнения на видеокарте // Информационные технологии и математическое моделирование: Материалы VIII Всероссийской научно-практической конференции с международным участием. (Анжеро-Судженск, 12 – 13 ноября 2009) – 2009. – С. 206–211.
13. Дружинин Д.В. Сжатие экранного видео с помощью видеокарты. Сравнение технологий // Вычислительные методы и программирование : Научный журнал. – 2008. – Т. 9, № 2. – С. 242–250.
14. Дружинин Д.В. Практическое сравнение кодека butterfly screen video codec с его аналогами // Информационные технологии и математическое моделирование: Материалы IX Всероссийской научно-практической конференции с международным участием (19 – 20 ноября 2010 г.). – 2010. – Т. 2. – С. 36–41.
15. Дружинин Д.В. Комбинированные алгоритмы сжатия ключевых кадров экранного видео // Вестник Томского государственного университета. Управление, вычислительная техника и информатика: Научный журнал. –

Томск, Издательство научно-технической литературы, 2013. – № 4(25). – С. 129–136.

16. Дружинин Д.В. Алгоритмы сжатия экранного видео, использующие корреляцию соседних кадров // Известия Алтайского государственного университета: Научный журнал. – 2014. – № 1/2(81). – С. 91–95.
17. Дружинин Д.В. Гибридный алгоритм сжатия дискретно-тоновой графики // Компьютерная безопасность и криптография: Труды Всероссийской конференции «XII Сибирская школа-семинар с международным участием» (8 – 13 сентября 2014 г.). – 2014 – № 7. – С. 116–118.
18. Дружинин Д.В. Применение гибридного алгоритма для сжатия экранного видео // Информационные технологии и математическое моделирование: Материалы XII Всероссийской научно-практической конференции с международным участием им. А. Ф. Терпугова (29 – 30 ноября 2013 г.). – 2013. – Т. 1. – С. 16–20.
19. Дружинин Д.В. Развитие гибридного алгоритма сжатия дискретно-тоновой графики // Научное творчество молодежи. Математика. Информатика: Материалы XVIII Всероссийской научно-практической конференции (24 – 25 апреля 2014 г.). – 2014. – Т. 1. – С. 124–128.
20. Дружинин Д.В. Эволюция гибридного алгоритма сжатия дискретно-тоновой графики // Интеллектуальный потенциал XXI века: ступени познания: Материалы XIX Молодежной международной научно-практической конференции (18 декабря 2013 г.). – 2013. – С. 117–121.
21. ДубльГИС. [Электронный ресурс] / ДубльГИС. URL: https://2gis.ru (дата обращения 25.06.2013).
22. Замятин А.В., Саринова А.Ж. Алгоритм сжатия гиперспектральных аэрокосмических изображений с использованием предварительной байтовой обработки и учетом междиапазонной корреляции // Прикладная информатика. – 2013. – №5(47). – С. 38–45.
23. Замятин А.В. Дифференцированное сжатие аэрокосмических изображений с потерями // Информационные технологии, № 6. – 2011.– С.60–65.

24. Замятин А.В., То Динь Чыонг. Повышение эффективности алгоритма сжатия многозональных аэрокосмических изображений // Известия Томского политехнического университета. – 2008. – Т.313, № 5. – С. 24–28.
25. Замятин А.В., То Динь Чыонг. Сжатие многозональных аэрокосмических изображений с использованием вейвлет-преобразования и учетом междиапазонной зависимости // Известия Томского политехнического университета. – 2008. – Т. 313, № 5. – С. 20–24.
26. Книжный И.М. Сжатие статических изображений с постоянной скоростью сжимающего кодирования в задачах дистанционного зондирования Земли : диссертация к.т.н.: 05.12.04. – М., 2006. – 153 с.
27. Кобелев В.Ю. Сжатие сигналов и изображений при помощи оптимизированных вейвлет-фильтров : диссертация к.т.н.: 05.12.04. – М., 2006. – 166 с.
28. Малахов Э.В., Замятин А.В. Сжатие в архивировании и каталогизации данных дистанционного зондирования Земли // «Земля из космоса: наиболее эффективные решения», 5-ая международная конференция: Тезисы докладов. – 2011. – С. 140–142.
29. Лавров В.А. Написание своего mirror видеодрайвера // Вестник Томского государственного университета. – 2004. – № 284. – С. 191–194.
30. Лавров В.А., Чертов А.А. Методы и алгоритмы для оптимизации видеороликов формата fbr по размеру // Вестник Томского государственного университета. – 2006. – № 290. – С. 297–299.
31. Луна Ф.Д. Введение в программирование трехмерных игр с DirectX 9.0. [Электронный ресурс] URL: http://www.proklondike.com/file/Other/Frank_Luna__3dGamesProgrammingIntro (RUS).rar, (дата обращения 25.06.2013).
32. Осокин А.Н., Сидоров Д.В. Модифицированный кодер стандарта jpeg с контролем битрейта // Интернет-журнал Науковедение. – 2013. – № 5(18). – С. 1–9.

33. Резервное копирование данных. [Электронный ресурс] / http://screentube.ru URL: http://screentube.ru/2009/02/10/резервное-копирование-данных (дата обращения 25.06.2013).
34. Рубина И. С., Тропченко А. Ю. Исследование алгоритмов кодирования преобразованием в задачах сжатия кадров видеопоследовательности // Известия вузов. Приборостроение. – 2012. – № 10(55). – С. 26–30.
35. Саринова А.Ж., Афанасьев А.А., Замятин А.В. Программное обеспечение для сжатия гиперспектральных аэрокосмических изображений без потерь // Свидетельство о государственной регистрации программы для ЭВМ № 2014660049 от 1.10.2014 г.
36. Свидетельство о государственной регистрации программы для ЭВМ «Butterfly Screen Video Codec» № 2009615754. Роспатент, 2009 г.
37. Снимки экрана с преобладанием текста, деловой графики, сделанные в Windows XP. [Электронный ресурс] / Д.В. Дружинин. URL: https://docs.google.com/file/d/0B_2xi7pVvd23RzAyNVJWWDJBSUU/edit?usp=sharing (дата обращения 25.06.2013).
38. Снимки экрана, сделанные в Ubuntu Gnome 14.04 [Электронный ресурс] / Д.В. Дружинин. URL: https://drive.google.com/file/d/0B_2xi7pVvd23VjIwY213dEtwa3M/view?usp=sharing (дата обращения 27.09.2015).
39. Снимки экрана, сделанные в Windows 7. [Электронный ресурс] / Д.В. Дружинин. URL: https://docs.google.com/file/d/0B_2xi7pVvd23VXdGMHlPT21JVms/edit?usp=sharing (дата обращения 25.06.2013).
40. Соловьев А. В. Метод автоматического набора связующих точек для выполнения взаимного ориентирования аэрокосмических снимков // Наукоемкие технологии в космических исследованиях Земли. 2016. №3. URL: https://cyberleninka.ru/article/n/metod-avtomaticheskogo-nabora-svyazuyuschih-tochek-dlya-vypolneniya-vzaimnogo-orientirovaniya-aerokosmicheskih-snimkov (дата обращения: 21.08.2018).

41. Стандарт MPEG-2. [Электронный ресурс] URL: http://mpeg.chiariglione.org/standards/mpeg-2 (дата обращения 25.06.2013).
42. Сэломон Д. Сжатие данных, изображений и звука. / перевод с англ. В.В. Чепыжова – М. : Техносфера, 2006. – 365 с. – (Мир программирования).
43. Тестовые данные для алгоритмов отсечения неизменившихся областей. [Электронный ресурс] / Д.В. Дружинин. URL: https://drive.google.com/file/d/0B_2xi7pVvd23SE1maU5CR0d0MlU/view?usp=sharing (дата обращения 01.02.2016).
44. Тропченко А.Ю., Тропченко А.А. Методы сжатия изображений, аудиосигналов и видео: Учебное пособие – СПб: СПбГУ ИТМО, 2009. – 108 с.
45. Штарьков Ю.М., Чокенс Ф.М.Дж., Виллемс Ч.Дж. Оптимальное универсальное кодирование по критерию максимальной индивидуальной относительной избыточности // Проблемы передачи информации. – 2004. – Т. 40, № 1. – С. 98–110.
46. Agha S., Dwyer V. M. Algorithms and VLSI Architectures for MPEG-4 Motion Estimation. [Электронный ресурс] URL: http://www.lboro.ac.uk/departments/el/research/esc-miniconference/papers/agha.pdf (дата обращения 25.06.2013).
47. Akare U.P., Bawane N.G. Compression of old marathi manuscript images using context-based, adaptive, lossless image coding. // 2017 International Conference on Computing Methodologies and Communication (ICCMC). Erode, India, 2017, С. 745-750.
48. Bandicam. [Электронный ресурс] / Bandicam Company. URL: https://www.bandicam.com/ru (дата обращения 25.03.2018).
49. BB FlashBack. [Электронный ресурс] / Blueberry software. URL: www.bbsoftware.co.uk/bbflashback.aspx (дата обращения 25.06.2013).
50. Bhusnurmath A., Camillo J. Graph cuts l1 norm minimization. [Электронный ресурс] // IEEE Transactions on pattern analysis and machine intelligence. 2008. Том 30, № 10. – С. 1866-1871. URL:

http://www.cis.upenn.edu/~cjtaylor/PUBLICATIONS/pdfs/BhusnurmathPAMI08.pdf (дата обращения 25.06.2013).

51. Frames Per Second during screen-sharing in Connect Meeting. [Электронный ресурс] / Adobe Systems Incorporated. URL: http://blogs.adobe.com/connectsupport/frames-per-second-during-screen-sharing-in-connect-meeting (дата обращения 25.05.2016).
52. Campbell M., Calvo M. The Facts About Mirror Drivers in Screen readers. [Электронный ресурс] URL: http://serotek.com/mirror-driver-paper.html (дата обращения 25.06.2013).
53. CamStudio Video Software. [Электронный ресурс] / CamStudio. URL: http://camstudio.org (дата обращения 25.06.2013).
54. Carpentieri B. LZ-based Adaptive Compression for Images. [Электронный ресурс] URL: http://www.wseas.us/e-library/conferences/2011/Prague/AICT/AICT-13.pdf (дата обращения 26.12.2013).
55. CCITT G3/G4 Image Compression SDK Technology [Электронный ресурс] / LEAD Technologies. URL: https://www.leadtools.com/sdk/compression/ccitt
56. Chandra Krintz, Sezgin Sucu. Adaptive On-the-Fly Compression. [Электронный ресурс] // IEEE Transactions on parallel and distributed systems. Том 17, № 1. июнь 2006. – С. 15-24. URL: http://www.cs.ucsb.edu/~ckrintz/papers/acejournal.pdf (дата обращения 25.06.2013).
57. Cinepak [Электронный ресурс] / Codec Central. URL: http://www.siggraph.org/education/materials/HyperGraph/video/codecs/Cinepak.html (дата обращения 25.06.2013).
58. CharLS. [Электронный ресурс] / CodePlex. URL: http://charls.codeplex.com (дата обращения 25.06.2013).
59. Colantoni P., Boukala N., Da Rugna J. Fast and Accurate Color Image Processing Using 3D Graphics Cards. [Электронный ресурс] URL: http://colantoni.nerim.net/articles/VMV2003.pdf (дата обращения 25.06.2013).

60. C++ Programming Tutorial for Beginners in English - Part 1. [Электронный ресурс] / Google Inc. URL: http://www.youtube.com/watch?v=S3t-5UtvDN0 (дата обращения 25.06.2015).
61. D3DFORMAT. [Электронный ресурс] / Microsoft Corporation. URL: https://msdn.microsoft.com/ru-ru/library/windows/desktop/bb172558(v=vs.85).aspx (дата обращения 25.06.2016).
62. Deuerling-Zheng Y. Motion compensation in digital subtraction angiography using graphics hardware. / Y. Deuerling-Zheng и др. // Computerized Medical Imaging and Graphics. – 2006. – С. 279–289.
63. Dirac Specification. [Электронный ресурс] // Dirac video codec. URL: http://dirac.sourceforge.net/DiracSpec2.2.0.pdf (дата обращения 25.06.2013).
64. Dondi P., Lombardi L., Cinque L. RDVideo: A new lossless video codec on GPU. // Image analysis and processing – ICIAP 2011: 16th International Conference, Ravenna, Italy, September 14-16, 2011, Proceedings, Part II. – С. 158–167.
65. Duh D.J., Jeng J.H., Chen S.Y. DCT based simple classification scheme for fractal image compression. [Электронный ресурс] URL: http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/2220/18469/00850550.pdf?temp=x (доступ платный, дата обращения 25.06.2013).
66. Eclipse. [Электронный ресурс] / The Eclipse Foundation. URL: https://eclipse.org (дата обращения 25.06.2013).
67. El-Sakka M.R. Adaptive digital image compression based on segmentation and block classification. [Электронный ресурс] URL: http://uwspace.uwaterloo.ca/bitstream/10012/465/1/NQ44784.pdf (дата обращения 25.06.2013).
68. El-Sakka M.R., Kamel M.S. Adaptive Image Compression Based on Segmentation and Block Classification. [Электронный ресурс] URL: http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel4/5852/15601/00723520.pdf?arnumber=723520 (доступ платный, дата обращения 25.06.2013).

69. El-Sakka M.R. Context-based Dictionary Image Compression. [Электронный ресурс] URL: http://www.taibahu.edu.sa/iccit/allICCITpapers/pdf/p140-el-sakka.pdf (дата обращения 26.12.2013).
70. FastAc. [Электронный ресурс] / A. Said. URL: http://www.cipr.rpi.edu/research/SPIHT/spiht3.html (дата обращения 25.06.2013).
71. Firefox. [Электронный ресурс]. / Mozilla Foundation. URL: https://www.mozilla.org
72. Fu S.W., Ding J.J., Huang Y.W. и другие. Collagen image compression using the JPEG-based predictive lossless coding scheme. // 2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC). Kuala Lumpur, Malaysia, 2017, С. 524–533.
73. GPUDecoder. [Электронный ресурс] // DivideFrame. URL: http://www.divideframe.com (дата обращения 25.06.2013).
74. Google VP8 VideoCodec. [Электронный ресурс] / Softpile. URL: http://google-vp8-video-for-windows-codec.softpile.com (дата обращения 25.06.2013).
75. Grois D., Marpe D. Performance Comparison of H.265/MPEG-HEVC, VP9, and H.264/MPEG-AVC Encoders. [Электронный ресурс] / D. Grois, D. Marpe. URL: http://iphome.hhi.de/marpe/download/Performance_HEVC_VP9_X264_PCS_2013_preprint.pdf (дата обращения 25.05.2016).
76. Hangouts. [Электронный ресурс] / Google. URL: https://hangouts.google.com (дата обращения 25.04.2016).
77. Hartley R., Vidal R. Three-View Multibody Structure from Motion // IEEE Transactions on Pattern Analysis & Machine Intelligence, vol. 30, 2007. – pp. 214–227.
78. Hernández-Cabronero M., Marcellin M.W., Blanes I., Serra-Sagristà J. Lossless Compression of Color Filter Array Mosaic Images With Visualization via JPEG 2000. // IEEE Transactions on Multimedia. vol. 20, no. 2, С. 257–270.
79. High Quality DXT Compression using CUDA. [Электронный ресурс] / NVIDIA Corporation. URL:

http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/dxtc/doc/cuda_dxtc.pdf (дата обращения 25.06.2013).

80. Holwerda T. Intel Forced to Remove "Cripple AMD" Function from Compiler? [Электронный ресурс] / T. Holwerda. URL: http://www.osnews.com/story/22683/Intel_Forced_to_Remove_quot_Cripple_AMD_quot_Function_from_Compiler_ (дата обращения 25.06.2015).
81. Hooks. [Электронный ресурс] / Microsoft Corporation. URL: http://msdn.microsoft.com/en-us/library/ms632589.aspx (дата обращения 25.06.2013).
82. Huffman D.A., A Method for the Construction of Minimum-Redundancy Codes. // Proceedings of the IRE, vol. 40, no. 9, pp. 1098-1101, Sept. 1952.
83. Image Denoising. [Электронныйресурс] / NVIDIA Corporation. URL: http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/imageDenoising/doc/imageDenoising.pdf (дата обращения 25.06.2013).
84. inno Screen Capture Codec. [Электронный ресурс] / RSupport. URL: http://www.innoheim.com/iscc.php (дата обращения 25.06.2013).
85. Intel C++ Compilers in Intel INDE. [Электронный ресурс] / Intel. URL: https://software.intel.com/en-us/c-compilers/inde (дата обращения 25.06.2015).
86. JPEG 2000. [Электронный ресурс] / MainConcept. URL: http://www.mainconcept.com/products/sdks/video/jpeg-2000.html (дата обращения 25.06.2013).
87. Kanade T., Khan S., Akhter I. and Sheikh Y. Trajectory Space: A Dual Representation for Nonrigid Structure from Motion // IEEE Transactions on Pattern Analysis & Machine Intelligence. – 2010. – Vol. 33, pp. 1442–1456.
88. Kane J., Yang Q. Compression Speed Enhancements to LZO for Multi-core Systems. // 2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing. New York, NY, 2012, С. 108–115.
89. Keller Y., Averbuch, A. Fast motion estimation using bidirectional gradient methods. [Электронный ресурс] URL:

http://www.eng.biu.ac.il/_kellery1/publications/pdf/optical_flow_ieee_final.pdf, (дата обращения 25.06.2013).

90. Kim T.K. Blocking effect reduction of compressed images using classification-based constrained optimization. [Электронный ресурс] / T. K. Kim, J.K. Paik, C.S. Won и другие. URL: http://cat.inist.fr/?aModele=afficheN&cpsidt=1420117 (доступ платный, дата обращения 25.06.2013).
91. Kruger J., Schneider J., Westermann R. A Structure for Point Scan Compression and Rendering. [Электронный ресурс] / отв. ред. M. Pauly, M. Zwicker. 2005. URL: http://wwwcg.in.tum.de/Research/data/Publications/pbg05.pdf (дата обращения 25.06.2013).
92. Lee Y., Hirakawa K. and Nguyen T.Q. Lossless compression of CFA sampled image using decorrelated Mallat wavelet packet decompositionю // 2017 IEEE International Conference on Image Processing (ICIP), Beijing, China, 2017, СС. 2721–2725.
93. Li B., Xu J., Sullivan G., Zhou Y., Lin B. Adaptive motion vector resolution for screen content. // Joint Collaborative Team on Video Coding, Doc. JCTVC-S0085, Oct. 2014.
94. Lin T., Zhang P., Wang S., Zhou K., Chen X. Mixed chroma sampling-rate high efficiency video coding for full-chroma screen content. // IEEE Trans. Circuits Syst. Video Technol., vol. 23, no. 1, Jan. 2013, pp. 173–185.
95. LZO. [Электронный ресурс] / oberhumer.com. URL: http://www.oberhumer.com/opensource/lzo (дата обращения 25.06.2013).
96. Ma Z., Wang W., Xu M., Yu H. Advanced screen content coding using color table and index map. // IEEE Trans. Image Process., Oct. 2014, vol. 23, no. 10, pp. 4399–4413.
97. Ma Z., Yue T., Cao X., Xu Y., Li X., Wang Y. Interactive Screen Video Streaming-Based Pervasive Mobile Workstyle. // IEEE Transactions on Multimedia, Oct. 2017, vol. 19, no. 10, pp. 2322-2332.
98. Macroexpress. [Электронный ресурс]. URL: www.macros.com (дата обращения 25.06.2013).

99. Mehrotra S. Motion estimation/compensation for screen capture video. [Электронный ресурс] // Microsoft Corp URL: http://www.freepatentsonline.com/7224731.html (дата обращения 25.06.2013).
100. Mehrotra S. Rate allocation for mixed content video. [Электронный ресурс] // Microsoft Corp. URL: https://patents.google.com/patent/US7200276B2/en (дата обращения 01.04.2018).
101. Microsoft Expression Encoder 4. [Электронный ресурс] / Microsoft. URL: http://www.microsoft.com/en-us/download/details.aspx?id=18974 (дата обращения 25.06.2013).
102. Microsoft Office. [Электронный ресурс] / Microsoft. URL: https://products.office.com (дата обращения 25.06.2013).
103. Microsoft Video 1. [Электронный ресурс] / Microsoft. URL: http://www.moviecodec.com/video-codecs/microsoft-video-1-4275 (дата обращения 25.06.2013).
104. Microsoft Visual C++ 9.0. [Электронный ресурс] / Microsoft. URL: https://msdn.microsoft.com/en-us/library/bb384632(v=vs.90).aspx (дата обращения 25.06.2015).
105. Min C., Kim K., Cho H. и др. SFS: random write considered harmful in solid state drives. [Электронный ресурс] / C. Min, K. Kim, H. Cho и др. URL: https://www.usenix.org/legacy/event/fast12/tech/full_papers/Min.pdf (дата обращения 25.01.2018).
106. Mortensen J. Effect of Image Linearization on Normalized Compression Distance. [Электронный ресурс] / J. Mortensen и др. URL: http://jonathanmortensen.com/publications/MortensenWuNCD.pdf (дата обращения 26.12.2013).
107. Morvan Y., Farin D., P.H.N. de With. Incorporating depth-image based view-prediction into h.264 for multiview-image coding. [Электронный ресурс] URL: http://vca.ele.tue.nl/publications/data/Morvan2007d.pdf, (дата обращения 25.06.2013).

108. MSU Screen Capture Lossless Codec. [Электронный ресурс] / Д. Попов. URL: http://compression.ru/video/ls-codec/screen_capture_codec_en.html (дата обращения 25.06.2013).
109. NVIDIA CUDA Compute Unified Device Architecture. Programming Guide. [Электронный ресурс] / NVIDIA Corporation. URL: http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf (дата обращения 25.06.2013).
110. Pan Z., Shen H., Lu Y., Li S. A low-complexity screen compression scheme for interactive screen sharing. // IEEE Trans. Circuits Syst. Video Technol. – Jun. 2013, vol. 23, no. 6, pp. 949–960.
111. Pesquet-Popescu B., Cagnazzo M., Dufaux F. Motion Estimation Techniques. [Электронный ресурс] / TELECOM ParisTech, 2013. URL: https://www.semanticscholar.org/paper/Motion-Estimation-Techniques-Pesquet-Popescu-Cagnazzo/98caf8325abb40aa6bbddd0e7f5d3a6c366d03e6 (дата обращение 20.08.17)
112. Pieters B., Van Rijsselbergen D., De Neve W. Motion Compensation and Reconstruction of H.264/AVC-coded Pictures using the GPU. [Электронный ресурс] URL: http://symposium.elis.ugent.be/archive/symp2006/papers_poster/paper084_Bart_Pieters.pdf (дата обращения 25.06.2013).
113. Quality Center 6 Bug Tracking [Электронный ресурс] / http://www.youtube.com URL: http://www.youtube.com/watch?v=T2V8I80X8g0 (дата обращения 25.06.2015).
114. Radmin. [Электронный ресурс] URL: www.radmin.ru (дата обращения 25.06.2013).
115. RemotePC. [Электронный ресурс] / Pro Softnet Corporation. URL: www.remotepc.com (дата обращения 25.06.2013).
116. Reˇr´abek M., Ebrahimi T. Comparison of compression efficiency between HEVC/H.265 and VP9 based on subjective assessments. [Электронный ресурс] / M. Reˇr´abek, T. Ebrahimi. URL:

https://infoscience.epfl.ch/record/200925/files/article-vp9-submited-v2.pdf (дата обращения 25.05.2016).

117. Rijsselbergen D. YCoCg(-R) Color Space Conversion on the GPU // Sixth FirW Symposium. Ghent University, 2005. - статья № 102.
118. Said A. Image block classification based on entropy of differences. [Электронный ресурс] // FreePatentsOnline.com. URL: http://www.freepatentsonline.com/7397953.html (дата обращения 25.06.2013).
119. Singh S., Kumar V., Verma H.K. Adaptive threshold-based block classification in medical image compression for teleradiology. [Электронный ресурс] URL: http://www.journals.elsevierhealth.com/periodicals/cbm/article/PIIS0010482506001533/fulltext (доступ платный, дата обращения 25.06.2013).
120. Shader Model 3 (Direct3D 9). [Электронный ресурс] / Microsoft Corporation // DirectX SDK (August 2007) Documentation. – 1 электрон. опт. диск (CD).
121. Shader Model 4 Features. [Электронный ресурс] / Microsoft Corporation. URL: http://msdn.microsoft.com/en-us/library/bb509657(VS.85).aspx (дата обращения 25.06.2013).
122. Shahbahrami A. Evaluation of Huffman and Arithmetic Algorithms for Multimedia Compression Standards. / A. Shahbahrami и др. // International Journal of Computer Science, Engineering and Applications (IJCSEA). 2011, Volume 1, Number 4.
123. Shannon C. E. A Mathematical Theory of Communication // Bell System Technical Journal. – 1948. – Т. 27. – С. 379–423, 623–656.
124. Shen G. Accelerate Video Decoding With Generic GPU. [Электронный ресурс] / G. Shen и др. // IEEE transactions on circuits and systems for video technology. 2005. Том 15, № 5. – С. 685-693. URL: http://wenku.baidu.com/view/463158150b4e767f5acfce85.html (дата обращения 25.06.2013).
125. Skype. [Электронный ресурс] / Microsoft. URL: https://www.skype.com (дата обращения 25.04.2016).

126. Strzodka R., Garbe C. Real-Time Motion Estimation and Visualization on Graphics Cards // Proceedings IEEE Visualization. 2004. – С. 545–552.
127. TechSmith Screen Codec. [Электронный ресурс] / TechSmith. URL: www.techsmith.com/download.html (дата обращения 25.06.2013).
128. Terminal equipment and protocols for telematic services. Facsimile coding schemes and coding control functions for group 4 facsimile apparatus. ITU-T Recommendation T.6. [Электронный ресурс] / International Telecommunication Union. URL: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-T.6-198811-I!!PDF-E&type=items (дата обращения 25.04.2016).
129. TestFairy. [Электронный ресурс] / TestFairy. URL: https://testfairy.com (дата обращения 25.04.2016).
130. TestFire. [Электронный ресурс] / TestFire INC. URL: https://testfire.io (дата обращения 25.04.2016).
131. TightVNC. [Электронный ресурс] / AT&T Laboratories Cambridge. URL: http://www.sfr-fresh.com/windows/misc/tightvnc-1.3.9_winsrc.zip (дата обращения 25.06.2013).
132. Timoner J., Freeman D. M. Multi-Image Gradient-Based Algorithms for Motion Estimation. [Электронный ресурс] URL: http://people.csail.mit.edu/samson/papers/Timoner_Freeman_MultiImageGradient.pdf (дата обращения 25.06.2013).
133. Tropchenko A., Tropchenko A., V. T. Nguyen. Research of Block-Based Motion Estimation Methods for Video Compression. // TEM Journal. – August 2016. Volume 5, Issue 3, Pages 277-283.
134. UltraVNC. [Электронный ресурс] URL: http://www.uvnc.com (дата обращения 25.06.2013).
135. Van der LaanJ. W., Jalba A. C., Roerdink J. B. Wavelet Lifting on Graphics Hardware for Faster Video Decoding. [Электронный ресурс] URL: http://www.ictonderzoek.net/3/assets/File/posters/2007_102/2007_102.pdf (дата обращения 25.06.2013).

136. Wada M., Yamaguchi H. Movement estimation system for video signals using a recursive gradient method. [Электронный ресурс] // FreePatentsOnline.com. URL: http://www.freepatentsonline.com/4695882.html (дата обращения 25.06.2013).
137. Wang S., Zhang X., Liu X. и другие. Utility-driven adaptive preprocessing for screen content video compression. // IEEE Trans. Multimedia. – May 2017. Vol. 19, no. 3, pp.660–667.
138. Weise T., Leibe B., Van Gool L. A Fast 3D Scanning with Automatic Motion Compensation. [Электронный ресурс] URL:http://www.vision.ee.ethz.ch/˜bleibe/papers/weise-motioncompensation-cvpr07.pdf (дата обращения 25.06.2013).
139. Windows Aero. [Электронный ресурс] / Microsoft. URL: http://windows.microsoft.com/en-us/windows-vista/what-is-windows-aero (дата обращения 25.05.2016).
140. Windows Media Encoder 9 Series. [Электронный ресурс] / Microsoft. URL: http://www.microsoft.com/en-us/download/details.aspx?id=17792 (дата обращения 25.06.2013).
141. Xu J., Joshi R., Cohen R. A. Overview of the Emerging HEVC Screen Content Coding Extension. // IEEE Transactions on Circuits and Systems for Video Technology. Jan. 2016. – vol. 26, no. 1, pp. 50–62.
142. Zamyatin A. Multistage algorithm for lossless compression of multispectral remote sensing images // The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences (IAPRS), Vol. 38, Part 7A, Vienna, Austria, 2010, pp. 304–309.
143. Zamyatin A., Sarinova A., Cabral P. The Compression Algorithm of Hyperspectral Space Images Using Pre-byte Processing and Intra-bands Correlation // GEOProcessing 2014: The Sixth International Conference on Advanced Geographic Information Systems, Applications, and Services (IARIA). – pp. 70–75.

144. Zandonai D. An Architecture for MPEG Motion Estimation. [Электронный ресурс] / D. Zandonai и др. URL: http://www.iberchip.org/VII/cdnav/pdf/38.pdf (дата обращения 25.06.2013).
145. Ziv J., Lempel A. Compression of individual sequences via variable-rate coding. // IEEE Transactions on Information Theory. September 1978. Vol. 24, no. 5, pp. 530–536.
146. Zlib [Электронный ресурс] / J. GaillyandM. Adler. URL: http://www.zlib.net/ (дата обращения 25.06.2013).

ПРИЛОЖЕНИЕ А. КОПИЯ СВИДЕТЕЛЬСТВА О ГОСУДАРСТВЕННОЙ РЕГИСТРАЦИИ ПРОГРАММЫ ДЛЯ ЭВМ «BUTTERFLY SCREEN VIDEO CODEC»

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2009615754

Butterfly Screen Video Codec

Правообладатель(ли): *Дружинин Денис Вячеславович (RU)*

Автор(ы): *Дружинин Денис Вячеславович (RU)*

Заявка № 2009614530
Дата поступления 19 августа 2009 г.
Зарегистрировано в Реестре программ для ЭВМ
16 октября 2009 г.

Руководитель Федеральной службы по интеллектуальной собственности, патентам и товарным знакам

Б.П. Симонов

ПРИЛОЖЕНИЕ Б. КОПИИ АКТОВ О ВНЕДРЕНИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Акт о внедрении программного обеспечения
«Butterfly Screen Video Codec»

Настоящий акт свидетельствует, что программное обеспечение «Butterfly Screen Video Codec», разработанное Дружининым Денисом Вячеславовичем, внедрено в ООО «Армадэйт».

Процесс внедрения проходил с сентября 2011 года по ноябрь 2012 года.

Кодек осуществляет сжатие экранного видео без потерь информации, обеспечивая при этом высокую степень сжатия (до 1000 раз относительно несжатых данных). При этом кодек предъявляет низкие требования к системным ресурсам.

Заявленные характеристики системы предполагали наличие следующих основных возможностей:

1. Сжатие кадров экранного видео с последующим сохранением в указанный файл;
2. Сжатие кадров экранного видео с последующим возвратом указателя на закодированный кадр;
3. Декодирование кадров экранного видео из указанного файла;
4. Декодирование кадров экранного видео из структуры данных, создаваемой при кодировании;
4. Возможность установить кодеру частоту вставки ключевого кадра;
5. Возможность передать декодеру иконку курсора мыши, которая будет использована при отрисовке, или отключить отрисовку курсора мыши;
6. Переход к кадру с указанным номером;

В ходе эксплуатации программы подтверждено, что она обладает всеми заявленными возможностями.

Внедрение кодека «Butterfly Screen Video Codec» позволило компании «Армадэйт» выйти на рынок кодеков для экранного видео. Кодек стал первым в линейке продуктов в сфере обработки видео.

КОПИЯ ВЕРНА
Служба документоведения и делами
Г. Михеева

ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ «Армадэйт» ТОМСК * РОССИЯ *

Технический директор
ООО "Армадэйт" Михайлов И.В.
03.07.13

Утверждаю
Заместитель директора по науке
АУ «Югорский НИИ
информационных технологий»
Парегородцев А.Л.

АКТ ВНЕДРЕНИЯ

результатов научно-исследовательских работ

Настоящим актом подтверждается, что результаты работы: «Разработка и исследование эффективных методов сжатия экранного видео», **выполненной:** соискателем кафедры теоретических основ информатики Федерального государственного автономного образовательного учреждения высшего образования «Национальный исследовательский Томский государственный университет»
Дружининым Денисом Вячеславовичем,
проводимой: с 12 сентября 2012 года по 27 ноября 2013 года,
внедрены: ЮНИИТ.

1. Вид внедренных результатов: кодек для сжатия экранного видео «Butterfly Screen Video Codec».

2. Характеристика масштабов внедрения: АУ «Югорский НИИ информационных технологий».

3. Форма внедрения: создание пользовательского приложения для записи экранного видео, использующего кодек «Butterfly Screen Video Codec».

4. Описание научно-технической продукции: кодек для сжатия экранного видео, обеспечивающий высокую степень сжатия и предъявляющий низкие требования к системным ресурсам. Кодек производит сжатие без потерь информации и предоставляет удобный интерфейс.

5. Социальный, экономический и научно-технический эффект: внедрение кодека «Butterfly Screen Video Codec» позволило создать наглядные обучающие материалы, иллюстрирующие использование создаваемых в АУ «Югорский НИИ информационных технологий» приложений. В свою очередь пользователи этих приложений получили возможность создавать отчёты об ошибках, проиллюстрированные видео сценария работы с приложением, приводящего к возникновению конкретной ошибки.

Заведующий лабораторией
Сетевых информации оных технологий Огородников И.Н.

ПРИЛОЖЕНИЕ В. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ЭФФЕКТИВНОСТИ ПРЕДЛОЖЕННЫХ АЛГОРИТМОВ

В приложении приведены расширенные результаты тестирования в табличном виде.

В таблице 1 приведены результаты тестирования нескольких разработанных алгоритмов, при этом для гибридного сдвигово-группового алгоритма данные приведены без учёта финального статистического сжатия. В таблицах 2 – 6 приведены результаты тестирования двух АСПИ, при этом на втором этапе второго из них использовался zlib – реализация алгоритма стандарта Deflate с уровнем сжатия 6 и LZO_X_999 с уровнем сжатия 6, как обладающие сбалансированными показателями степени сжатия и вычислительной эффективности.

Таблица В.1 – Разработанные алгоритмы сжатия ключевых кадров GUI-видео (первый этап сжатия)

№	Алгоритм / Параметр	Тип изображения	Время кодирования / декодирования, мс	Размер после сжатия, Б
1	Сдвиговый алгоритм с байтовым форматом	Windows XP	26 / 125	849346
		Windows 7	104 / 375	2558339
		Gnome 14	73 / 298	2474928
		Текст	32 / 132	850672
		Деловая графика	28 / 136	839296
2	Сдвиговый алгоритм с битовым форматом	Windows XP	24 / 120	744736
		Windows 7	97 / 362	2224242
		Gnome 14	71 / 295	2170102
		Текст	30 / 125	739714
		Деловая графика	28 / 131	732842
3	Гибридный алгоритм с байтовым форматом	Windows XP	5 / 8	190697
		Windows 7	78 / 54	1511222,8
		Gnome 14	18 / 22	555676
		Текст	8 / 14	287404
		Деловая графика	22 / 21	184605,5
4	Гибридный алгоритм с битовым форматом	Windows XP	6 / 9	140468
		Windows 7	76 / 51	1315515,5
		Gnome 14	19 / 23	409312
		Текст	8 / 14	231238
		Деловая графика	20 / 19	165598,7
5	Гибридный алгоритм с битовым форматом и алгоритмом CCITT Group 4	Windows XP	44 / 22	138846
		Windows 7	195 / 74	1309385
		Gnome 14	89 / 50	404285

№	Алгоритм / Параметр	Тип изображения	Время кодирования / декодирования, мс	Размер после сжатия, Б
		Текст	57 / 21	202572
		Деловая графика	52 / 25	160925
6	Гибридный алгоритм с битовым форматом и алгоритмом пространственного группового кодирования	Windows XP	42,2 / 20	130742
		Windows 7	189 / 72	1226477,7
		Gnome 14	86 / 48	380972,6
		Текст	50,5 / 18,7	118380,2
		Деловая графика	48,2 / 24	149065,8

Таблица В.2 – АСПИ (набор данных «Windows XP»)

№	Первый шаг АСПИ	Второй шаг АСПИ	Время кодирования / декодирования, мс	Размер после сжатия, Б
1	Сдвиговый алгоритм с байтовым форматом	LZO, алгоритм Хаффмана	53 / 137	594383
		zlib	50 / 133	524287
2	Сдвиговый алгоритм с битовым форматом	LZO, алгоритм Хаффмана	49 / 130	521315
		zlib	48 / 128	459713
3	Гибридный алгоритм с байтовым форматом	LZO, алгоритм Хаффмана	25 / 12	133487
		zlib	24 / 11	117714
4	Гибридный алгоритм с битовым форматом	LZO, алгоритм Хаффмана	24 / 10	98327
		zlib	24 / 10	86708
5	Гибридный алгоритм с битовым форматом и алгоритмом CCITT Group 4	LZO, алгоритм Хаффмана	62 / 25	97104
		zlib	58 / 24	86162
6	Гибридный алгоритм с битовым форматом и алгоритмом пространственного группового кодирования	LZO, алгоритм Хаффмана	58,1 / 24,6	91029,1
		zlib	56 / 22	80693,2

Таблица В.3 – АСПИ (набор данных «Изображения с преобладанием текста»)

№	Первый шаг АСПИ	Второй шаг АСПИ	Время кодирования / декодирования, мс	Размер после сжатия, Б
1	Сдвиговый алгоритм с байтовым форматом	LZO, алгоритм Хаффмана	57 / 141	644448
		zlib	59 / 140	531670
2	Сдвиговый алгоритм с битовым форматом	LZO, алгоритм Хаффмана	53 / 136	560389
		zlib	55 / 133	462321
3	Гибридный алгоритм с байтовым форматом	LZO, алгоритм Хаффмана	24 / 24	217730
		zlib	27 / 22	179627

№	Первый шаг АСПИ	Второй шаг АСПИ	Время кодирования / декодирования, мс	Размер после сжатия, Б
4	Гибридный алгоритм с битовым форматом	LZO, алгоритм Хаффмана	23 / 23	175180
		zlib	26 / 22	144523
5	Гибридный алгоритм с битовым форматом и алгоритмом CCITT Group 4	LZO, алгоритм Хаффмана	64 / 27	164926
		zlib	69 / 26	135016
6	Гибридный алгоритм с битовым форматом и алгоритмом пространственного группового кодирования	LZO, алгоритм Хаффмана	62,5 / 26,7	89682,6
		zlib	66 / 25	73976,5

Таблица В.4 – АСПИ (набор данных «Деловая графика»)

№	Первый шаг АСПИ	Второй шаг АСПИ	Время кодирования / декодирования, мс	Размер после сжатия, Б
1	Сдвиговый алгоритм с байтовым форматом	LZO, алгоритм Хаффмана	52 / 146	533563
		zlib	52 / 145	466275
2	Сдвиговый алгоритм с битовым форматом	LZO, алгоритм Хаффмана	51 / 141	465888
		zlib	51 / 140	407134
3	Гибридный алгоритм с байтовым форматом	LZO, алгоритм Хаффмана	39 / 27	117358
		zlib	39 / 26	107558
4	Гибридный алгоритм с битовым форматом	LZO, алгоритм Хаффмана	38 / 26	108275
		zlib	28 / 25	104998
5	Гибридный алгоритм с битовым форматом и алгоритмом CCITT Group 4	LZO, алгоритм Хаффмана	64 / 29	106726
		zlib	63 / 28	102187
6	Гибридный алгоритм с битовым форматом и алгоритмом пространственного группового кодирования	LZO, алгоритм Хаффмана	62,5 / 27,6	94742,1
		zlib	62 / 27	82764,7

Таблица В.5 – АСПИ (набор данных «Windows 7»)

№	Первый шаг АСПИ	Второй шаг АСПИ	Время кодирования / декодирования, мс	Размер после сжатия, Б
1	Сдвиговый алгоритм с байтовым форматом	LZO, алгоритм Хаффмана	329 / 477	1247970
		zlib	318 / 485	1282375
2	Сдвиговый алгоритм с битовым форматом	LZO, алгоритм Хаффмана	309 / 461	1084996
		zlib	305 / 466	1114908
3	Гибридный алгоритм	LZO, алгоритм Хаффмана	183 / 108	737181

№	Первый шаг АСПИ	Второй шаг АСПИ	Время кодирования / декодирования, мс	Размер после сжатия, Б
	с байтовым форматом	zlib	177 / 116	757504
4	Гибридный алгоритм с битовым форматом	LZO, алгоритм Хаффмана	181 / 106	641714
		zlib	175 / 110	659406
5	Гибридный алгоритм с битовым форматом и алгоритмом CCITT Group 4	LZO, алгоритм Хаффмана	280 / 115	632107
		zlib	272 / 122	651964
6	Гибридный алгоритм с битовым форматом и алгоритмом пространственного группового кодирования	LZO, алгоритм Хаффмана	275 / 113	597407
		zlib	266 / 119	614805

Таблица В.6 – АСПИ (набор данных «Ubuntu Gnome 14»)

№	Первый шаг АСПИ	Второй шаг АСПИ	Время кодирования / декодирования, мс	Размер после сжатия, Б
1	Сдвиговый алгоритм с байтовым форматом	LZO, алгоритм Хаффмана	118 / 262	2006250
		zlib	95 / 224	1719891
2	Сдвиговый алгоритм с битовым форматом	LZO, алгоритм Хаффмана	109 / 248	1759620
		zlib	91 / 215	1508060
3	Гибридный алгоритм с байтовым форматом	LZO, алгоритм Хаффмана	56 / 23	450565
		zlib	45 / 19	386153
4	Гибридный алгоритм с битовым форматом	LZO, алгоритм Хаффмана	53 / 19	331888
		zlib	45 / 17	284440
5	Гибридный алгоритм с битовым форматом и алгоритмом CCITT Group 4	LZO, алгоритм Хаффмана	132 / 49	327017
		zlib	108 / 39	280752
6	Гибридный алгоритм с битовым форматом и алгоритмом пространственного группового кодирования	LZO, алгоритм Хаффмана	129 / 47	307255
		zlib	106 / 37	264709

При тестировании разработанных и существующих алгоритмов сжатия ключевых кадров (Таблица 7) использовались реализация FastAC арифметического сжатия; реализация Main Concept JPEG 2000; реализации Microsoft алгоритмов JPEG Lossless; реализация CharLS алгоритма JPEG_LS; реализация LEAD Technologies алгоритма CCITT Group 4. Также в тестировании

принимала участие реализация алгоритма стандарта Deflate от Microsoft. Для гибридного сдвигово-группового алгоритма данные о степени сжатия и времени сжатия в таблице 7 приведены с учётом финальной обработки алгоритмом Хаффмана. Для этого все результирующие данные гибридного сдвигово-группового алгоритма объединяются в один массив.

Таблица В.7 – Разработанные и существующие алгоритмы сжатия ключевых кадров GUI-видео

№	Алгоритм / Параметр	Тип изображения	Время кодирования / декодирования, мс	Размер после сжатия, Б
1	LZO_X_999 уровень 1	Windows XP	56 / 11	124405
		Windows 7	199 / 58	764559,2
		Gnome 14	310 / 33	322303
		Текст	56,1 / 10,9	118184,1
		Деловая графика	53,9 / 9,1	106542,4
2	LZO_X_999 уровень 4	Windows XP	56, 6 / 10	112427,4
		Windows 7	203 / 56	715905,5
		Gnome 14	314 / 30	291272
		Текст	57,4 / 9,6	102429,1
		Деловая графика	55,4 / 8,7	98963,6
3	LZO_X_999 уровень 6	Windows XP	67,7 / 9	106388,6
		Windows 7	240 / 55	695053,9
		Gnome 14	375 / 27	275627
		Текст	72,6 / 8,5	91106,8
		Деловая графика	65,4 / 8,6	95126,4
4	LZO_X_999 уровень 9	Windows XP	370,6 / 8,8	101859,9
		Windows 7	929 / 54	687317,9
		Gnome 14	1686 / 25	267276
		Текст	880 / 7,8	81288,4
		Деловая графика	293,5 / 7,6	91764,1
5	LZO_X_1	Windows XP	11,2 / 11,7	142159
		Windows 7	72 / 73	862583,9
		Gnome 14	62 / 35	368299
		Текст	11,8 / 11	138447,3
		Деловая графика	10,1 / 10,3	124248,7
6	Гибридный алгоритм	Windows XP	54,2 / 33,5	117388
		Windows 7	227 / 108	924176
		Gnome 14	136 / 48	362831
		Текст	62,4 / 33	98180,3
		Деловая графика	60,2 / 32	112324,1
7	АСПИ, использующий LZO_X_999 уровень 6	Windows XP	58,1 / 24,6	91029,1
		Windows 7	275 / 113	597407,6
		Gnome 14	129 / 47	307255
		Текст	62,5 / 26,7	89682,6
		Деловая графика	62,5 / 27,6	94742,1

№	Алгоритм / Параметр	Тип изображения	Время кодирования / декодирования, мс	Размер после сжатия, Б
8	АСПИ, использующий LZO_X_999 уровень 9	Windows XP	64,1 / 23,1	90679
		Windows 7	694 / 114	589924,3
		Gnome 14	213 / 47	306046
		Текст	65 / 26,8	89578
		Деловая графика	68,2 / 26,6	94522,5
9	Microsoft Deflate	Windows XP	48,6 / 12	157206,5
		Windows 7	167 / 56	976739
		Gnome 14	94 / 14	393325
		Текст	50,6 / 10,9	126848,4
		Деловая графика	47,1 / 10,9	145175,6
10	zlib уровень 1	Windows XP	15 / 5	135744
		Windows 7	103 / 16	1052147,6
		Gnome 14	29 / 6	339626
		Текст	15 / 5	122339,75
		Деловая графика	14 / 4	122351,1
11	zlib уровень 4	Windows XP	32 / 4	110946,3
		Windows 7	117 / 16	701493,1
		Gnome 14	62 / 5	277583
		Текст	32 / 4	94714,5
		Деловая графика	32 / 4	101396,2
12	zlib уровень 6	Windows XP	38 / 4	104116,6
		Windows 7	130 / 16	661786,1
		Gnome 14	74 / 5	260496
		Текст	43 / 4	83749,25
		Деловая графика	36 / 4	95471,7
13	zlib уровень 9	Windows XP	89 / 4	99733,9
		Windows 7	230 / 16	653241,8
		Gnome 14	246 / 5	252031
		Текст	249 / 4	76921,5
		Деловая графика	77 / 4	90687,5
14	АСПИ, использующий zlib уровень 6	Windows XP	56 / 22	80693,2
		Windows 7	266 / 119	614804,8
		Gnome 14	106 / 37	264709
		Текст	66 / 25	73976,5
		Деловая графика	62 / 27	82764,7
15	АСПИ, использующий zlib уровень 9	Windows XP	57 / 22	80672,7
		Windows 7	299 / 111	613965,7
		Gnome 14	109 / 36	264648
		Текст	69 / 25	73857,4
		Деловая графика	62 / 25	82762,6
16	Алгоритм Хаффмана	Windows XP	78 / 66	1389133,1
		Windows 7	215 / 192	3890489
		Gnome 14	151 / 78	3475561
		Текст	52 / 42	880449,8
		Деловая графика	70 / 59	1232763,2
17	Арифметическое сжатие	Windows XP	35 / 74	1378388,1

№	Алгоритм / Параметр	Тип изображения	Время кодирования / декодирования, мс	Размер после сжатия, Б
		Windows 7	95 / 206	3863098,3
		Gnome 14	68 / 88	3448677
		Текст	31 / 68	841289,5
		Деловая графика	34 / 73	1216817,4
18	JPEG 2000 (без потерь)	Windows XP	437 / 361	298352
		Windows 7	1234 / 954	1048049
		Gnome 14	848 / 428	746466
		Текст	481 / 390	384686
		Деловая графика	456 / 375	315728
19	JPEG Lossless	Windows XP	41 / 54	358629
		Windows 7	118 / 145	911078
		Gnome 14	80 / 64	897276
		Текст	38 / 50	475470
		Деловая графика	45 / 56	348097
20	Алгоритм группового кодирования	Windows XP	4 / 4	342721
		Windows 7	15 / 11	1575184
		Gnome 14	29 / 6	857475
		Текст	3 / 3	286454
		Деловая графика	4 / 4	316114
21	CCITT Group 4	Windows XP	25 / 12	238452
		Windows 7	103 / 35	1355626
		Gnome 14	51 / 26	641714
		Текст	20 / 10	140853
		Деловая графика	28 / 15	311048
22	JPEG-LS	Windows XP	25 / 23	210390
		Windows 7	82 / 75	762053
		Gnome 14	48 / 27	526388
		Текст	24 / 20	184841
		Деловая графика	27 / 24	239670

В таблицах 8 и 9 использовалась реализация кодека H.264, входящая в состав ПО для фиксации и локального сохранения GUI-видео Bandicam [48]. Степень загруженности ЦП приведена в расчёте на одно ядро процессора. Поэтому значения свыше 100 означают, что процесс в среднем использовал более одного ядра процессора.

Таблица В.8 – Кодеки сжатия GUI-видеоданных (набор данных «Работа пользователя с прикладными приложениями»). Разрешение экрана 1920 × 1080

№	Кодек	Средняя степень загруженности ЦП, %	Размер сжатого файла за 1 минуту, Б	Средний уровень использования ОП, Мбайт
1	Butterfly Screen Video	25	1114254	35,2

№	Кодек	Средняя степень загруженности ЦП, %	Размер сжатого файла за 1 минуту, Б	Средний уровень использования ОП, Мбайт
	Codec			
2	BB FlashBack 4.1.5 [48]	40	1912221	156,5
3.1	H.264 (без потерь)	51,6	10917022	65,1
3.2	H.264 (уровень потерь 20 %)	51,4	8989120	65
4.1	CamStudio Lossless Codec v1.5 (LZO) [53]	25	17790720	26,1
4.2	CamStudio Lossless Codec v1.5 (GZIB)	25	17424384	26,3
5	Microsoft Video 1 [103]	42,5	150574848	26,2
6	Cinepak [57]	97,5	16724736	31,2
7.1	MSU Screen Capture Lossless Codec v1.2 (with Fades detection) [108]	40	3308544	40,3
7.2	MSU Screen Capture Lossless Codec v1.2 (without Fades detection)	25	5017344	44,1
8.1	Microsoft Expression Encoder 4 (30 kbps) [101]	15	7699944	54,8
8.2	Microsoft Expression Encoder 4 (10 kbps)	12,5	7603944	55,3
9	Windows Media Encoder 9 [140]	150	1133380	304,7
10.1	Google VP8 Video Codec (VBR, 5000 kbps)[74]	137,5	1825536	72,1
10.2	Google VP8 Video Codec (CBR, 5000 kbps)	152,5	1629696	72,9
11	inno Screen Capture Codec 1.9 [84]	32,5	2462976	30,9
12	TechSmith Screen Codec 2 [127]	37,5	25229721	159,5

Таблица В.9 – Кодеки сжатия GUI-видеоданных (набор данных «Работа пользователя с системными приложениями»). Разрешение экрана 1280 × 1024

№	Кодек	Средняя степень загруженности ЦП, %	Размер сжатого файла за 1 минуту, Б	Средний уровень использования ОП, Мбайт
1	Butterfly Screen Video Codec	20	1751353	25,7
2	BB FlashBack 4.1.5	37	3032653	133,4
3.1	H.264 (без потерь)	29,2	7259012	52
3.2	H.264 (уровень потерь 20 %)	29	5977099	51,7

№	Кодек	Средняя степень загруженности ЦП, %	Размер сжатого файла за 1 минуту, Б	Средний уровень использования ОП, Мбайт
4.1	CamStudio Lossless Codec v1.5 (LZO)	18,9	17840017	20,4
4.2	CamStudio Lossless Codec v1.5 (GZIB)	18,9	17700531	20,5
5	Microsoft Video 1	35	145297297	25,1
6	Cinepak	83,8	14800397	26,4
7.1	MSU Screen Capture Lossless Codec v1.2 (with Fades detection)	29,7	3423204	36,8
7.2	MSU Screen Capture Lossless Codec v1.2 (without Fades detection)	21,6	3548575	41,1
8.1	Microsoft Expression Encoder 4 (30 kbps)	25,7	16630782	47,5
8.2	Microsoft Expression Encoder 4 (10 kbps)	17	16287915	52,5
9	Windows Media Encoder 9	60	1941591	201,7
10.1	Google VP8 Video Codec (VBR, 5000 kbps)	86,5	1566720	43,6
10.2	Google VP8 Video Codec (CBR, 5000 kbps)	94,6	2540627	53,1
11	inno Screen Capture Codec 1.9	24,3	2836203	17,1
12	TechSmith Screen Codec 2	22,8	33662098	105,1

Таблица В.10 – Алгоритмы отсечения неизменившихся областей кадра

№	Алгоритм / Параметр	Тип изменения между кадрами	Время сжатия, мс	Размер после сжатия, Б
1	Алгоритм отсечения неизменившихся строк и столбцов	Сворачивание окна	4	921600
		Перемещение окна по диагонали	4	673554
2	Алгоритм отсечения неизменившихся блоков	Сворачивание окна	4	997632
		Перемещение окна по диагонали	4	476928
3	Адаптивный алгоритм отсечения неизменившихся областей кадра	Сворачивание окна	8	921600
		Перемещение окна по диагонали	8	476928

Таблица В.11 – Алгоритмы отсечения неизменившихся областей кадра (вся последовательность алгоритмов обработки промежуточных кадров)

№	Алгоритм / Параметр	Тип изменения между кадрами	Время сжатия, мс	Размер после сжатия, Б
1	Алгоритм отсечения неизменившихся строк и столбцов	Сворачивание окна	43	30743
		Перемещение окна по диагонали	32	22560
2	Алгоритм отсечения неизменившихся блоков	Сворачивание окна	44	32817
		Перемещение окна по диагонали	30	16315
3	Адаптивный алгоритм отсечения неизменившихся областей кадра	Сворачивание окна	47	30743
		Перемещение окна по диагонали	34	16315

В таблицах 12 и 13 замеряемый параметр «Количество блоков» означает количество блоков, для которых найдено соответствие данной реализацией. Суммарное количество блоков равно 1024 × 768 / (16 × 16) = 3072. При этом надо учитывать, что часть блоков не изменились по сравнению с предыдущим кадром. Запись 4 → 6 означает, что текущий кадр сначала обрабатывается реализацией (3) алгоритма оценки движения, а затем реализацией (5).

Таблица В.12 – Алгоритмы оценки движения (набор данных «Движения первого типа»)

№	Алгоритм / Параметр	Время выполнения, мс	Количество блоков, шт
1	Алгоритм оценки движения, представленный в [99]	10049	1436
2	Алгоритм оценки движения, представленный в [99], с начальной проверкой диагональных элементов	4012	1436
3	Модификация (1) алгоритма оценки движения (поиск блока, соответствующего текущему блоку, осуществляется только по вертикали и по горизонтали)	65	1436
4	Модификация (1) алгоритма оценки движения с начальной проверкой диагональных элементов	24	1436
5	Модификация (2) алгоритма оценки движения (поиск блока, соответствующего текущему блоку, осуществляется в ближайшей окрестности текущего блока)	1094	697
6	Модификация (2) алгоритма оценки движения с начальной проверкой диагональных элементов	510	697
7	4 → 6	24 → 10	1436 → 0

№	Алгоритм / Параметр	Время выполнения, мс	Количество блоков, шт
8	6 → 4	510 → 15	396 → 830

Таблица В.13 – Алгоритмы оценки движения (набор данных «Движения второго типа»)

№	Алгоритм / Параметр	Время выполнения, мс	Количество блоков, шт
1	Алгоритм оценки движения, представленный в [99]	3561	1384
2	Алгоритм оценки движения, представленный в [99] с начальной проверкой диагональных элементов	2422	1384
3	Модификация (1) алгоритма оценки движения (поиск блока, соответствующего текущему блоку, осуществляется только по вертикали и по горизонтали)	140	450
4	Модификация (1) алгоритма оценки движения с начальной проверкой диагональных элементов	63	450
5	Модификация (2) алгоритма оценки движения (поиск блока, соответствующего текущему блоку, осуществляется в ближайшей окрестности текущего блока)	62	1384
6	Модификация (2) алгоритма оценки движения с начальной проверкой диагональных элементов	47	1384
7	4 → 6	62 → 47	463 → 962
8	6 → 4	47 → 16	1379 → 38

В таблице 14 приведены данные о производительности дисковой подсистемы при записи на жёсткий диск данных в режиме, эмулирующем работу ПО для фиксации и локального сохранения GUI-видео в сжатом и несжатом виде. Запись порции данных производилась 10 раз в секунду, эмулируя запись отдельных кадров. Размер порции данных в несжатом виде был рассчитан исходя из разрешения монитора 1920 × 1080 пикселей. Тестирование проводилось на платформе с характеристиками: Intel Core i5 4570 3,2 ГГц, RAM 8 Гб, жёсткий диск Seagate ST500DM002 7200 rpm с типом подключения SATA 6 Гбит/с.

Таблица В.14 – Производительность дисковой подсистемы при фоновой записи кадров на жёсткий диск

Эмулируемый тип ПО	**Размер порции данных, КБ**	**Время копирования файла размером 1 ГБ, с**
С оперативным сжатием	150	17
Без оперативного сжатия	6075	55

ПРИЛОЖЕНИЕ Г. ФУНКЦИОНАЛЬНЫЙ ИНТЕРФЕЙС КОДЕКА СЖАТИЯ GUI-ВИДЕОДАННЫХ.

Типы данных

```
1. struct EncodedFrame
{
  BYTE** arrays;
  int* counts;
  int count;
}
```

Хранит данные закодированного кадра. Значение такого типа выступает в качестве возвращаемого значения функции BSC_CallEncodeToMemory(), а также в качестве параметра функции BSC_CallDecoderReadNextFrameFromMemory().

Поля структуры:

arrays массив массивов, являющихся частями закодированного кадра.

counts массив, содержащий количество байтов в каждом массиве в *arrays*.

count количество элементов в *arrays*.

Функции кодера

```
1. HANDLE BSC_CreateEncoderInstance(
                                    int a_frameWidth,
                                    int a_frameHeight)
```

Функция используется для инициализации кодера и должна быть вызвана перед использованием прочих функций кодера.

Параметры:

a_frameWidth максимальная ширина видео кадров, которые будут подвергаться сжатию (в пикселях).

a_frameHeight максимальная высота видео кадров, которые будут подвергаться сжатию (в пикселях).

Возвращаемое значение:

Идентификатор (указатель на созданный в ходе работы функции экземпляр кодера).

```
2. bool BSC_CallEncodeToFile(
                             HANDLE a_handle,
                             BYTE* a_currentBits,
                             BYTE* a_previousBits,
                             int a_frameNumber,
                             int a_frameSize,
                             int a_frameWidth,
                             int a_frameHeight)
```

Функция выполняет сжатие переданного кадра и сохраняет сжатые данные в файл.

Параметры:

a_handle	идентификатор, полученный от функции *BSC_CreateEncoderInstance()*.
a_currentBits	указатель на массив, содержащий пиксели текущего кадра.
a_previousBits	указатель на массив, содержащий пиксели предыдущего кадра.
a_frameNumber	индекс текущего кадра (начиная с 0).
a_frameSize	размер текущего кадра (в байтах).
a_frameWidth	ширина текущего кадра (в пикселях).
a_frameHeight	высота текущего кадра (в пикселях).

Возвращаемое значение:

True, если кадр закодирован успешно и false в противном случае.

```
3. EncodedFrame BSC_CallEncodeToMemory(
                            HANDLE a_handle,
                            BYTE* a_currentBits,
                            BYTE* a_previousBits,
                            int a_frameNumber,
                            int a_frameSize,
                            int a_frameWidth,
                            int a_frameHeight)
```

Функция выполняет сжатие переданного кадра и заполняет возвращаемую структуру данных *EncodedFrame*.

Параметры:

Используются те же параметры, как в функции *BSC_CallEncodeToFile()*.

Возвращаемое значение:

Если в ходе кодирования произошла ошибка, то в результирующем экземпляре *EncodedFrame* поле *count* устанавливает в -1.

```
4. bool BSC_CallEncoderSetFileName(
                                HANDLE a_handle,
                                const char* a_fileName)
```

Функция используется для задания пути к результирующему файлу. Этот файл будет использоваться для записи закодированных данных функцией *BSC_CallEncodeToFile()*.

Параметры:

a_handle	идентификатор, полученный от функции *BSC_CreateEncoderInstance()*.
a_fileName	путь к файлу.

Возвращаемое значение:

True, если путь к файлу установлен успешно, false в противном случае.

```
5. bool BSC_CallEncoderSetKeyFrameFrequency(
                                HANDLE a_handle,
```

```
                                    unsigned int a_frequency)
```

Функция используется для задания частоты, с которой в формируемый видеопоток будет вставляться ключевой кадр (I-frame). Каждый *a_frequency кадр* будет закодирован как ключевой. Остальные кадра будут закодированы как промежуточные (P-frame).

Параметры:

a_handle	идентификатор, полученный от функции *BSC_CreateEncoderInstance()*.
a_frequency	частота вставки ключевого кадра в формируемый видеопоток.

Возвращаемое значение:

True, если частота ключевого кадра установлена успешно, false в противном случае.

```
6. bool BSC_CallEncoderSetMaxFrameProcessingTime(
                                HANDLE a_handle,
                                   unsigned int a_maxTimeMs)
```

Функция используется для задания максимального времени обработки кадра. Если время обработки кадра вышло за границы a_maxTimeMs, динамически подключаемые модули не используются.

Параметры:

a_handle	идентификатор, полученный от функции *BSC_CreateEncoderInstance()*.
a_frequency	максимальное время обработки кадра (в мс).

Возвращаемое значение:

True, если максимальное время обработки кадра установлена успешно, false в противном случае.

```
7. char* BSC_CallEncoderGetLastError(HANDLE a_handle)
```

Функция используется для получения информации о последней ошибке.

Параметры:

a_handle	идентификатор, полученный от функции *BSC_CreateEncoderInstance()*.

Возвращаемое значение:

Пустая строка, если не было ошибок, в противном случае – строка, содержащая информацию об ошибке.

```
8. bool BSC_DestroyEncoderInstance(HANDLE a_handle)
```

Функция используется для уничтожения экземпляра кодера и освобождения занимаемых им ресурсов. Эта функция должна быть вызвана после использования всех остальных функций кодера.

Параметры:

a_handle	идентификатор, полученный от функции *BSC_CreateEncoderInstance()*.

Возвращаемое значение:

True, если экземпляр кодера, на который указывает *a_handle*, уничтожен успешно, false в противном случае.

Функции декодера

```
1. HANDLE BSC_CreateDecoderInstance(
                        int a_frameWidth,
                        int a_frameHeight)
```

Функция используется для инициализации декодера и должна быть вызвана перед использованием прочих функций декодера.

Параметры:

a_frameWidth максимальная ширина видео кадров, которые будут подвергаться декодированию (в пикселях).

a_frameHeight максимальная высота видео кадров, которые будут подвергаться декодированию (в пикселях).

Возвращаемое значение:

Идентификатор (указатель на созданный в ходе работы функции экземпляр кодера).

```
2. BYTE* BSC_CallDecoderReadNextFrameFromFile(
                        HANDLE a_handle)
```

Функция используется для чтения из файла, заданного с помощью вызова функции *BSC_CallDecoderSetFileName()*, и декодирования очередного кадра.

Параметры:

a_handle идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

Возвращаемое значение:

Указатель на массив пикселей декодированного кадра (в формате RGB24), если кадр декодирован успешно. NULL, если достигнут конец файла. Если произошла ошибка, возвращаемое значение также NULL, но в этом случае функция *BSC_CallDecoderGetLastError()* вернёт непустую строку.

```
3. BYTE* BSC_CallDecoderReadNextFrameFromMemory(
                        HANDLE a_handle,
                        EncodedFrame* a_encodedFrame)
```

Функция используется для декодирования кадра, заданного параметром *a_encodedFrame*.

Параметры:

a_handle идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

a_encodedFrame указатель на структуру *EncodedFrame*, содержащую закодированный кадр.

Возвращаемое значение:

То же, что и у функции *BSC_CallDecoderReadNextFrameFromFile()*.

```
4. bool BSC_CallDecoderGoToFileBeginning(HANDLE a_handle)
```

Функция используется, чтобы установить текущую позицию в начало файла.

Параметры:

a_handle идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

Возвращаемое значение:

True, если текущая позиция установлена в начало файла успешно, false в противном случае.

```
5. int BSC_CallDecoderGetCurrentFrameID(HANDLE a_handle)
```

Функция используется для получения идентификатора текущего кадра. Идентификатор представляет собой индекс кадра в последовательности (считая с 0).

Параметры:

a_handle идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

Возвращаемое значение:

Идентификатор кадра при успешном завершении функции, -1 в противном случае.

```
6. DWORD BSC_CallDecoderGetCurrentFrameTimeStamp(
                              HANDLE a_handle)
```

Функция используется для получения временной метки кадра. Это количество миллисекунд, прошедших со времени, когда кадр с номером 0 был закодирован. Поэтому кадр под номером 0 имеет временную метку также равную 0.

Параметры:

a_handle идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

Возвращаемое значение:

Временная метка кадра, если выполнение метода прошло успешно, максимальное значение типа *unsigned int* (2^32 - 1) в противном случае.

```
7. BITMAPINFO* BSC_CallDecoderGetBitmapInfo(HANDLE
a_handle)
```

Функция используется для получения информации о текущем кадре в виде указателя на экземпляр структуры *BITMAPINFO*.

Параметры:

a_handle идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

Возвращаемое значение:

Указатель на экземпляр структуры *BITMAPINFO*, входящей в состав .NET framework, если функция завершила выполнение без ошибок. NULL, если в ходе

выполнения функции произошла ошибка, или если ещё не декодировано ни одного кадра из входного файла. Наличие ошибки следует проверять путём вызова функции *BSC_CallDecoderGetLastError()*.

```
8. bool BSC_CallDecoderSetFileName(
                            HANDLE a_handle,
                            const char* a_fileName)
```

Функция используется для задания пути к входному файлу. Этот файл будет использован в качестве входного функцией *BSC_CallDecoderReadNextFrameFromMemory()*.

Параметры:

a_handle	идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.
a_fileName	Путь к входному файлу.

Возвращаемое значение:

True, если путь к входному файлу установлен успешно, false в противном случае.

```
9. bool BSC_CallDecoderSetDrawCursorFlag(
                            HANDLE a_handle,
                            bool a_value)
```

Функция используется для установления флага рисования курсора мыши. Если значение *a_value* true, декодер будет рисовать курсор мыш на каждом кадре. В противном случае курсор мыши не будет нарисован на декодируемых кадрах.

***Параметры*:**

a_handle	идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.
a_value	флаг рисования курсора мыши.

Возвращаемое значение:

True, если флаг рисования курсора мыши установлен успешно, false в противном случае.

***Примечания*:**

По умолчанию этот флаг имеет значение true. Пользовать может отображать нестандартный рисунок курсора мыши после декодирования кадра. В этом случае может быть использована функция *BSC_CallDecoderGetCursorPosition()* для получения текущей позиции курсора мыши.

```
10. bool BSC_CallDecoderSetCursorFilePath(
                HANDLE a_handle,
                const char* a_fileName)
```

Функция используется для задания пути к файлу, из которого будет подгружен рисунок курсора мыши.

***Параметры*:**

a_handle	идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

a_fileName путь к файлу, содержащему рисунок курсора мыши. Рисунок должен быть в формате bmp.

Возвращаемое значение:

True, если путь к файлу, содержащему рисунок курсора мыши, задан успешно, false в противном случае.

***Примечания*:**

Курсоры загружаются из bmp-файлов. Необходимо вызвать эту функцию перед декодированием кадров, если вы хотите, чтобы декодер рисовал курсор мыши. Если файл курсора не задан, курсор не будет рисоваться.

Один из стандартных курсоров поставляется вместе с демонстрационным проектом DecoderWinAPIInterface – “SimpleCursor1.bmp”.

11. `POINT` **`BSC_CallDecoderGetCursorPosition`**`(HANDLE a_handle)`

Функция используется для получения позиции курсора мыши в текущем кадре.

***Параметры*:**

a_handle идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

Возвращаемое значение:

Координаты курсора мыши, если функция завершила выполнение успешно, false в противном случае.

```
12. bool BSC_CallDecoderGoToFrame(
                            HANDLE a_handle,
                            unsigned int a_frameNumber)
```

Функция используется для изменения текущей позиции во входном файле.

Параметры:

a_handle идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

a_frameNumber номер кадра (считая с 0), к которому будет осуществлён переход во входном файле.

Возвращаемое значение:

True, если переход к заданному кадру осуществлён успешно, false в противном случае.

13. `BYTE*` **`BSC_CallDecoderGetCurrentFrame`**`(HANDLE a_handle)`

Функция используется для получения последнего декодированного кадра.

Параметры:

a_handle идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

Возвращаемое значение:

Указатель на массив пикселей последнего декодированного кадра (текущего кадра) в формате RGB24, если предварительно декодирован хотя бы один кадр.

NULL в случае возникновения ошибки, или если ещё не декодировано ни одного кадра из входного файла. Наличие ошибки следует проверять путём вызова функции *BSC_CallDecoderGetLastError()*.

14. `char* BSC_CallDecoderGetLastError(HANDLE a_handle)`

Функция используется для получения информации о последней ошибке.

Параметры:

a_handle	идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

Возвращаемое значение:

Пустая строка, если не было ошибок, в противном случае – строка, содержащая информацию об ошибке.

15. `bool BSC_DestroyDecoderInstance(HANDLE a_handle)`

Функция используется для уничтожения экземпляра декодера и освобождения занимаемых им ресурсов. Эта функция должна быть вызвана после использования всех остальных функций декодера.

Параметры:

a_handle	идентификатор, полученный от функции *BSC_CreateDecoderInstance()*.

Возвращаемое значение:

True, если экземпляр декодера, на который указывает *a_handle*, уничтожен успешно, false в противном случае.